**Chapter 1**

**INTRODUCTION**

In today’s fast-paced and digitally-driven world, the landscape of education is undergoing significant transformation. With the integration of advanced technologies such as Artificial Intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP), the concept of learning has evolved from traditional classroom instruction to intelligent, adaptive, and personalized digital platforms. One of the most promising advancements in this domain is the application of Large Language Models (LLMs) to educational content generation. These models, trained on massive datasets, possess the ability to understand and generate human-like language, making them exceptionally useful in automating repetitive academic tasks, such as question formulation and answer generation.

**1.1 Background and Context**

The Relational Data Model, a core component of Database Management Systems (DBMS), provides a well-defined and structured approach to organizing data using tables, attributes, keys, and constraints. It serves as a blueprint for storing and retrieving data in relational databases, which are extensively used across various industries including education, healthcare, banking, and e-commerce. Understanding relational data models is a fundamental requirement for students pursuing courses in computer science, data science, and information technology. Despite its importance, learning about relational models can be abstract and complex, particularly when students are limited to static examples and manual practice.

To address this challenge, auto question generation presents itself as an innovative and impactful solution. By leveraging the consistent and rule-based structure of relational schemas, it becomes possible to systematically generate academic questions that not only test factual recall but also promote conceptual understanding, critical thinking, and applied knowledge. Such questions may include identifying keys, understanding relationships, analyzing constraints, or performing normalization—all derived directly from the database schema.

The integration of LLMs such as OpenAI’s GPT into this process further enhances its effectiveness. These models are capable of parsing schema metadata, identifying logical relationships, and producing grammatically and contextually correct questions in natural language. When applied thoughtfully, LLMs can tailor questions to different learning levels (easy, medium, hard), adjust tone and complexity, and even simulate tutor-student interactions.

In the context of this internship at AksharaPlus, the focus is on applying LLMs to automate question generation specifically for the relational data model. AksharaPlus is an organization committed to enhancing learning experiences through AI-driven educational tools. By automating one of the most time-consuming aspects of instructional design—creating high-quality questions—this project not only supports educators but also empowers students to engage with the material more actively and effectively.

This chapter sets the stage by outlining the motivations behind the study, the role of relational data in educational assessments, and how LLMs are poised to revolutionize question generation. It emphasizes the academic, technological, and societal relevance of building intelligent systems that support scalable and personalized learning.

**1.2 Problem Statement**

In the realm of computer science education, particularly in database systems, assessment plays a crucial role in reinforcing learning and evaluating conceptual clarity. Traditional assessments rely heavily on manually crafted questions created by instructors. While this manual approach allows for customization, it also presents significant limitations, especially in terms of scalability, consistency, and adaptability. The effort required to design multiple question types—ranging from simple definitions to complex application-based scenarios—places a heavy burden on educators. This challenge becomes even more pronounced in environments such as online courses, MOOCs, or large-scale classroom settings where hundreds or thousands of learners may need individualized assessment content.

One of the specific pain points lies in the context of the relational data model. A typical relational database comprises multiple interconnected tables defined by primary keys, foreign keys, data types, and constraints. These elements form the core learning material in foundational database courses. However, creating questions that accurately reflect the intricacies of a given schema—such as identifying candidate keys, reasoning about referential integrity, or applying domain constraints—requires not only technical expertise but also time-consuming effort.

Moreover, the need for differentiated instruction has led to an increased demand for assessments that can adapt to various student proficiency levels. A single static question bank does not serve the needs of all learners. Novices require straightforward factual recall questions, while advanced learners benefit more from analytical and application-based challenges. Developing and maintaining a dynamic question set that can address these diverse needs is practically unmanageable through manual methods alone.

Another critical challenge is the alignment of assessment content with the relational schema in use. In real-world teaching, multiple database schemas may be employed across different exercises, projects, and labs. Manually adjusting questions to match each schema is tedious and error-prone. There is a clear need for a system that can automatically interpret schema structure and generate relevant, schema-specific questions on the fly.

At the same time, the rise of intelligent tutoring systems and AI-driven EdTech platforms demands a more scalable approach to assessment design. Question generation must be both automatic and intelligent—capable of interpreting schema metadata, understanding database concepts, and formulating questions that are not only technically correct but also pedagogically meaningful.

This project addresses these issues by proposing the design and implementation of an automatic question generation system for the relational data model using Large Language Models (LLMs). The system will take a relational schema as input and output a diverse set of questions—including multiple-choice, true/false, and open-ended questions—classified by difficulty and aligned with curriculum goals. By automating this process, the system aims to:

* Reduce the burden on educators by generating high-quality questions programmatically.
* Provide learners with an adaptive learning experience that responds to their performance level.
* Ensure alignment between the assessment content and the relational schema being taught or practiced.
* Increase the scalability and flexibility of content creation in both classroom and online education settings.

In summary, the problem this project seeks to solve lies at the intersection of database education, intelligent content generation, and the growing need for adaptive and scalable assessment tools. By applying the capabilities of LLMs to structured relational data, the project aims to create a system that supports both educators and learners in achieving deeper and more effective engagement with database concepts.

1.3 Objectives of the Study

The primary goal of this project is to develop an intelligent, automated system capable of generating pedagogically meaningful questions from relational database schemas using the power of Large Language Models (LLMs). This system aims to enhance the way students learn about database concepts while reducing the effort and overhead required by educators to develop assessments.

To achieve this overarching goal, the study is structured around the following specific objectives:

1. **To Model and Understand the Structure of Relational Databases**

One of the foundational requirements of this project is a comprehensive understanding of relational databases and their underlying schema. This includes grasping the formal structure of tables (relations), rows (tuples), columns (attributes), and the constraints that bind them (such as primary keys, foreign keys, unique constraints, and domain restrictions). Before any question can be generated intelligently, the system must be capable of parsing and understanding the metadata within a database schema. This objective ensures that the generator is not producing random or disconnected content but is instead grounded in the actual structure of the relational model in use.

1. **To Design and Develop Question Templates Based on Database Concepts**

The next objective is to create a library of question templates that represent various types of assessment formats—Multiple Choice Questions (MCQs), True/False, and Descriptive questions. Each template corresponds to a specific concept within the relational data model, such as relational schema, entity integrity, referential integrity, candidate keys, normalization, domain constraints, etc. These templates serve as structural blueprints into which schema-specific information will be injected by the LLM. Designing effective templates is crucial because it ensures that the output remains consistent, accurate, and relevant to the curriculum.

1. **To Map Templates to Difficulty Levels Using Educational Frameworks**

Not all learners have the same level of understanding. Therefore, this project includes the objective of classifying question templates into difficulty tiers—easy, medium, and hard—based on Bloom’s Taxonomy. Questions in the easy category may focus on basic definitions and schema navigation. Medium-level questions may test conceptual understanding and relational analysis, while hard-level questions may require synthesis, evaluation, or schema-based problem-solving. This approach ensures that the generated questions can cater to a broad range of learners, from beginners to advanced.

1. **To Integrate a Large Language Model (LLM) for Schema-Aware Question Generation**

Once the question templates are created and difficulty levels are assigned, the next step is to use a pre-trained LLM (such as GPT) to dynamically populate these templates with context-specific information extracted from the relational schema. The LLM must interpret placeholders like {table\_name}, {column\_name}, and {constraint} and generate human-like language to formulate grammatically correct and semantically appropriate questions. This objective highlights the system’s ability to combine the rigid structure of database schemas with the flexible expressive power of language models.

1. **To Build an Interactive Tool for Learners and Educators**

An essential part of this study is the creation of a user-friendly interface—either as a command-line utility or a web-based application—that allows users to choose the topic, database schema, question type, and difficulty level. This tool is not merely a backend engine but an educational aid that provides real-time question generation for self-assessment, practice, and even examination preparation. Such an interface enhances usability and ensures that the system can be effectively used in real educational environments.

1. **To Evaluate the Quality and Educational Value of Generated Questions**

Merely generating questions is not sufficient. The final objective is to assess the quality, clarity, correctness, and educational alignment of the generated questions. This will involve both automatic validation (such as syntax and schema-checking) and manual evaluation (such as expert educator feedback). The evaluation framework will consider whether the questions are answerable, logically consistent with the schema, and beneficial for the learner’s understanding of relational databases.

1. **To Contribute to the Broader Goals of AksharaPlus and EdTech Innovation**

This project aligns with the larger mission of AksharaPlus—an EdTech organization focused on AI-powered learning solutions. By building an intelligent question generation system, this work contributes directly to AksharaPlus’s initiative to develop scalable, personalized, and data-driven education tools. The system not only has academic implications but also practical relevance in supporting scalable education for large classrooms, online learners, and corporate training environments.

In conclusion, the objectives outlined above provide a roadmap for the successful design, development, and deployment of an LLM-based automatic question generation system grounded in relational database theory. Each objective contributes to the creation of a comprehensive tool that transforms how database education can be delivered and assessed in modern learning environments.

**1.4 Internship Organization: AksharaPlus**

AksharaPlus is a dynamic education technology startup that focuses on transforming the traditional learning experience using the power of artificial intelligence (AI), natural language processing (NLP), and full-stack development. The organization envisions a future where learning is not only accessible and interactive but also intelligent and personalized. Its initiatives are centered around creating scalable, data-driven platforms that support learners at all levels, from school students to working professionals.

As a key stakeholder in the AI-powered EdTech domain, AksharaPlus provides real-world opportunities for interns to work on cutting-edge projects that blend core computer science knowledge with practical applications. These internships are more than academic exercises—they are innovation-driven engagements aimed at solving real-world problems in educational technology.

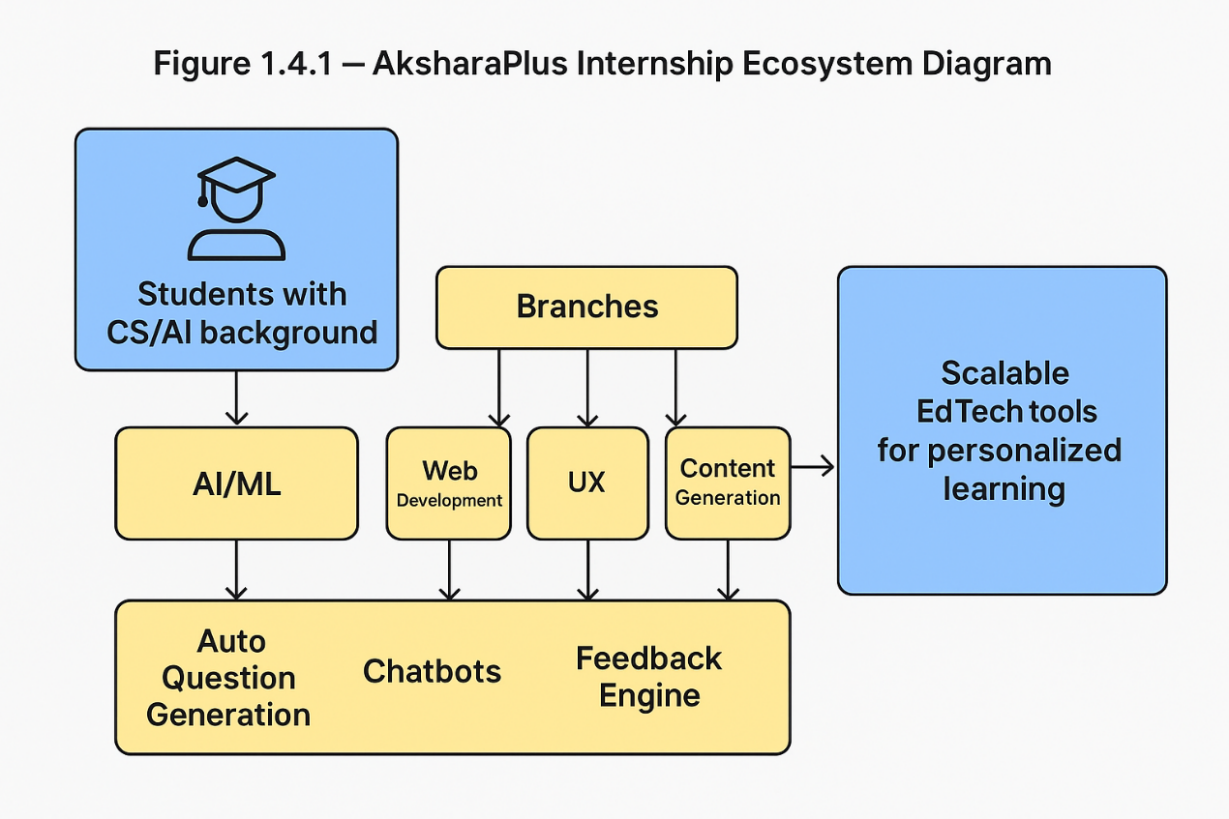
One of the most forward-thinking verticals at AksharaPlus is the Auto Question Generation (AQG) initiative, which seeks to automate the creation of meaningful and customized academic questions across multiple subjects and difficulty levels. The focus of this particular internship project is on applying large language models (LLMs) to generate questions from structured relational database schemas. By doing so, it addresses a critical gap in digital education: the ability to assess students in real-time based on the context of their current learning and practice activities.

Interns at AksharaPlus are not just coders—they are solution designers. They are encouraged to think holistically, contribute creatively, and solve problems that educators and learners face in everyday academic settings. Each intern has the opportunity to explore a range of domains under guided mentorship while contributing to the broader development of intelligent learning ecosystems.

AksharaPlus's broader focus areas include:

* AI-driven auto question generation for structured and unstructured learning domains.
* Personalized learning engines using LLMs and Retrieval-Augmented Generation (RAG).
* Interactive chatbots for 24x7 learner support.
* Intelligent feedback loops that adapt questions based on learner performance.
* Scalable full-stack platforms designed for mobile and web accessibility.
* Integration of visuals and animation (e.g., using Python libraries like matplotlib or manim) for interactive explanation of concepts.

By aligning with this vision, the current internship project adds significant value to AksharaPlus’s goals. The automatic question generation tool designed as part of this project helps educators minimize repetitive workload, while enabling learners to access instant, personalized assessments generated from any given relational database schema.

This internship is not only a platform for skill development but also a space where academic knowledge meets enterprise-level product development. The outcome of this internship will feed directly into AksharaPlus's future roadmap, potentially being integrated into flagship products or open-sourced for educational institutions.

**Figure 1.4.1 AksharaPlus Internship Ecosystem Diagram**

**A diagram of a process

AI-generated content may be incorrect.Figure 1.4.2 Project Focus Overview: Auto Question Generation Pipeline**

**1.5 Scope of the Project**

The scope of this project defines the boundaries, deliverables, and limitations of the system being developed. It ensures clarity regarding what the project aims to accomplish and what areas are intentionally left out for future exploration or out of feasibility constraints. For this internship, the focus is on building a template-based, schema-aware question generation system powered by a Large Language Model (LLM) that takes structured relational database schemas as input and outputs contextually meaningful and pedagogically aligned questions.

The following sections highlight what is included (in-scope) and excluded (out-of-scope) from the project.

✅ In Scope

1. Schema Parsing and Metadata Extraction

The system will parse relational schemas provided in JSON format, extracting information such as table names, column names, data types, primary keys, foreign keys, and constraints. This parsed metadata forms the foundation for generating questions that are specific to the structure and rules of the schema.

1. Question Template Design

A core part of the project involves creating multiple types of question templates (MCQ, True/False, and Text-based) for 7 core subtopics:

* Relational Schema
* Domain Constraints
* Key Constraints
* Candidate Keys and Primary Keys
* Foreign Keys and Referential Integrity
* NOT NULL & Entity Integrity
* Assertion Constraints

Each template includes placeholders such as {table\_name} and {column\_name} that are later replaced with schema-specific values.

1. Difficulty Classification

Templates are categorized based on complexity into three levels:

* Easy: Basic recall or identification (e.g., “What is the primary key?”)
* Medium: Intermediate reasoning and constraint interpretation (e.g., “Which column in table X is a foreign key?”)
* Hard: Advanced, cross-table, or design-based questions (e.g., “Design a candidate key strategy for table X and justify.”)

1. LLM Integration

The system will use a pre-trained language model (e.g., GPT) to intelligently fill in templates using metadata from the schema. The LLM ensures that questions are grammatically correct, semantically rich, and natural sounding.

1. User Interaction Module

A command-line interface (CLI) will be developed where users can:

* Select a database schema (e.g., hospital\_db, college\_db)
* Choose a topic and difficulty level
* Specify the number of questions to generate
* Optionally view schema reference and ER diagram
* Opt to view or hide answers

1. ER Diagram Viewing

If an ER diagram file exists (e.g., streamlit-based .py visualization), the system can open it in a browser window to allow the user to visualize table relationships before attempting questions.

1. Output Storage

The generated questions will be saved in structured JSON files (e.g., relational\_model\_full\_qna.json) to support reusability, review, and future analysis.

1. Evaluation and Grading of Learner Responses

This project focuses solely on question generation. Automatic grading, scoring, or answer verification is not included in the current phase.

1. Unstructured Data or Natural Language Schema Parsing

The system is designed to work with structured relational schema metadata (JSON format). Natural language descriptions of tables or ERDs are not parsed.

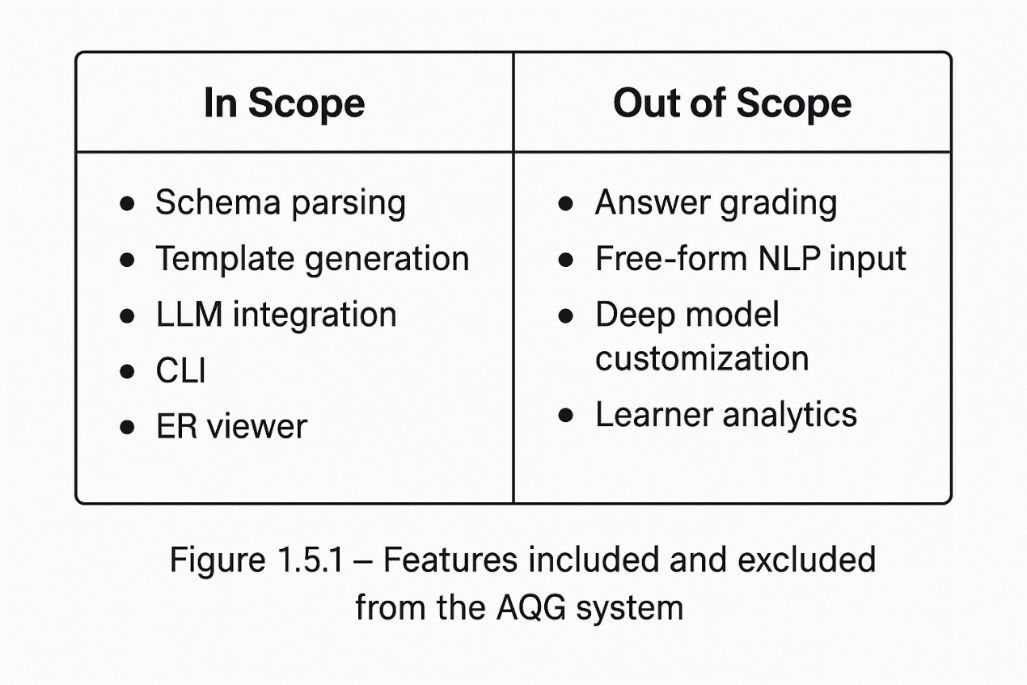
1. Adaptive Learning Loops

While the long-term vision may include learner feedback loops, this version does not adapt questions based on user history or correctness patterns.

1. Complex Reasoning Across Multiple Databases

The current version only supports single database schema processing. Cross-schema comparison or multi-database integration is beyond the scope.

1. Deep Customization of LLM Behavior

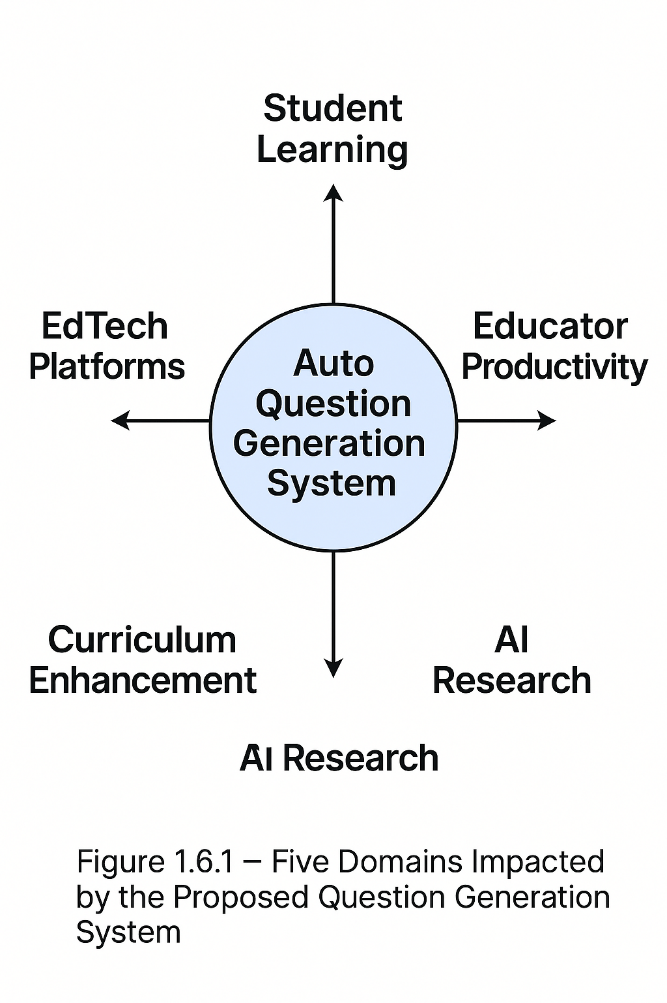
LLM outputs are based on pre-trained, general-purpose capabilities. Fine-tuning or domain-specific training of the model is not covered.

**Figure 1.5.1 Scope Boundaries of the Project**

**1.6 Significance of the Study**

The growing demand for intelligent and adaptive learning systems has positioned artificial intelligence as a cornerstone of educational technology innovation. Among various applications, the automatic generation of academic questions holds special relevance due to its role in assessment, self-evaluation, and skill reinforcement. This study is significant because it bridges the gap between structured database concepts and natural language generation, using Large Language Models (LLMs) to deliver high-quality, schema-specific questions that align with curriculum standards.

From a pedagogical perspective, assessments are not just a means to evaluate knowledge—they are instruments to promote deeper learning. However, creating questions that accurately reflect a student’s proficiency, the structure of a given database, and the learning objectives of a topic is a time-consuming and expertise-driven task. This study proposes an intelligent system that automates this process, thereby supporting teachers and instructors in focusing more on analysis and teaching rather than manual content creation.

Figure 1.6.1 – Impact Areas of the AQG System

The impact of this work extends across several key dimensions:

1. **Academic Relevance**

This project demonstrates how foundational database concepts such as schema design, relational constraints, and entity relationships can be taught more interactively through automated assessments. By customizing questions based on real database schemas, learners engage with the content in a more authentic, application-oriented manner. The project reinforces understanding of database architecture while developing schema reading and interpretation skills.

1. **Practical Utility**

Educators often struggle with the repetitive nature of question design, particularly when questions must be adjusted to match different database schemas in labs, assignments, or exams. This system alleviates that burden by generating new, relevant questions each time a schema is introduced. It supports dynamic classroom setups, remote labs, and even online testing environments with minimal manual effort.

1. **Scalability**

One of the primary goals of EdTech is to create solutions that scale efficiently across users, classes, institutions, and geographies. The AQG system proposed in this study can be integrated into various learning management systems (LMS) and deployed for hundreds or even thousands of students simultaneously. It scales well for MOOCs (Massive Open Online Courses), coding bootcamps, and higher education.

1. **Innovation in Educational AI**

The project applies cutting-edge NLP techniques to an educational problem in a way that is both practical and impactful. By combining template-driven logic with the generative power of LLMs, the system presents a hybrid approach that is deterministic in structure but flexible in language—achieving balance between correctness and natural fluency. This is an emerging area of interest in the AI + Education community.

1. **Institutional Contribution**

As part of the AksharaPlus internship initiative, this study aligns with the organization’s mission to develop personalized, AI-enhanced learning platforms. It demonstrates how real internship projects can lead to the development of production-ready, research-informed tools. The project has the potential to be integrated into larger systems developed by AksharaPlus, contributing not only to your academic portfolio but also to a product ecosystem that reaches real learners.

1. **Research and Future Opportunities**

The project lays the foundation for further research into schema-aware language generation, educational data mining, and student modeling. It opens up possibilities for future work in grading generated answers, adaptively generating questions based on learner feedback, and extending question generation to other domains like ER modeling, normalization, or SQL query design.

**Chapter 2**

**LITERATURE REVIEW**

The purpose of a literature review in any research endeavor is to provide context, establish relevance, and position the present work within the continuum of scholarly and technological advancements. For this project—focused on automatic question generation (AQG) from relational data models using Large Language Models (LLMs)—a multidisciplinary exploration is essential. This chapter provides a detailed overview of existing research in five key domains: automatic question generation, relational schema understanding, template-based educational question systems, the integration of LLMs in educational technology, and evaluation methodologies for generated content. By reviewing prior work and identifying gaps, this chapter lays the foundation for the methodology and design of the proposed system.

**2.1 Evolution of Automatic Question Generation (AQG)**

Automatic question generation is a subfield of natural language processing that has gained momentum over the past two decades. Early AQG systems relied heavily on rule-based, template-driven frameworks. These systems used grammatical and syntactic rules to transform declarative sentences into interrogatives. Mitkov et al. (2006) and Heilman & Smith (2010) developed some of the foundational work in this area, producing questions based on surface-level lexical and syntactic features.

With the rise of machine learning, neural AQG models emerged. Sequence-to-sequence (Seq2Seq) architectures enabled the development of data-driven systems where encoder-decoder pairs could generate relevant questions from unstructured text. However, the main limitation of such models was their dependency on large, domain-specific training data and their lack of interpretability.

Later, attention mechanisms and transformer-based architectures, such as those used in Google's T5 and OpenAI’s GPT series, revolutionized AQG by enabling contextual understanding and long-range dependency modeling. These models moved AQG beyond just syntax and into semantics, generating fluent, diverse, and contextually relevant questions with minimal human intervention. However, these advances were mostly limited to text inputs, not structured data.

**2.2 Structured Data and Schema-Based AQG**

While AQG from unstructured text is well-explored, question generation from structured sources like relational databases is comparatively underdeveloped. Structured data—such as that represented in SQL schemas—has a predictable format but lacks narrative flow, making it difficult for traditional text-based models to process.

Most work in this space focuses on SQL query generation (semantic parsing), where models are trained to translate natural language into executable SQL. However, the reverse process—producing pedagogically meaningful questions from schemas—is sparse. In a few experimental setups, question templates were filled using metadata from relational databases, but the resulting systems were brittle and domain-locked.

For instance, some systems could generate “What is the primary key of table X?” by reading schema metadata but failed to generate higher-order or application-level questions. There remains a critical gap in generating schema-aware, difficulty-tiered, and educationally-aligned questions dynamically and fluently.

**2.3 The Role of Relational Data Models in Education**

Relational database systems form the cornerstone of modern information systems and are a central part of computer science, data science, and software engineering curricula. Understanding concepts such as table design, normalization, primary and foreign keys, constraints, and schema relationships is essential for students.

Traditionally, educators create custom questions tied to specific database schemas used in labs and exercises. However, this process is manual, time-consuming, and non-scalable. Moreover, it lacks adaptability to individual learner performance. The ability to automatically generate assessment questions from a relational schema—specific to the structure students are working on—can significantly enhance both teaching efficiency and student engagement.

As educational systems move toward personalization and intelligent tutoring, the need for automatic, context-aware question generation from database schemas becomes not just valuable, but essential.

**2.4 Template-Based Educational Question Generation**

Template-based question generation offers a pragmatic solution to balancing structure and flexibility. It involves creating skeletal question formats with placeholders (e.g., “What is the data type of {column\_name} in {table\_name}?”), which can later be populated using context-specific information.

Earlier systems were fully deterministic: for every new table or column, a new set of questions had to be manually constructed. More recent systems have incorporated parameterization, enabling one template to yield multiple questions across different schemas.

The advantage of template-based systems is their interpretability, repeatability, and alignment with curriculum outcomes. However, without natural language enhancement, template-generated questions can appear robotic and grammatically awkward.

This is where integration with LLMs becomes crucial. By using LLMs to fill in placeholders and rephrase sentences, systems can produce questions that are fluent, contextually rich, and human-like, while retaining the pedagogical structure of templates.

**2.5 Use of Large Language Models (LLMs) in Educational Tools**

Transformer-based LLMs such as GPT-3, GPT-4, BERT, and T5 have demonstrated remarkable proficiency in understanding and generating natural language. These models are trained on vast corpora and can perform a wide range of tasks, including summarization, translation, question answering, and dialogue.

In the educational domain, LLMs are being used for:

* Conversational tutoring (AI tutors)
* Automatic grading and feedback
* Personalized content delivery
* Generation of quizzes, practice problems, and explanations

The key advantage of using LLMs for question generation lies in their ability to generate coherent and varied questions with minimal input. When guided using carefully designed prompts or templates, LLMs can adapt to any schema, topic, or difficulty level—making them ideal for structured domains like relational databases.

This project uses LLMs in a controlled environment: templates define the conceptual structure, while the LLM ensures grammaticality, variation, and fluency.

**2.7 Difficulty Calibration and Educational Frameworks**

A significant component of any question generation system is its ability to adjust the difficulty of the questions it generates. Educational research, particularly Bloom’s Taxonomy, has long categorized learning objectives into cognitive tiers: Remember, Understand, Apply, Analyze, Evaluate, and Create.

This project uses a simplified difficulty map:

* Easy: Recall-based questions (e.g., “What is a foreign key?”)
* Medium: Conceptual interpretation (e.g., “Why is a composite key used in this table?”)
* Hard: Application and synthesis (e.g., “Design a schema ensuring referential integrity across three tables”)

Templates are tagged accordingly, and the LLM is instructed to generate content aligned with the cognitive level intended. This difficulty calibration ensures better learner engagement and supports adaptive testing mechanisms.

**2.8 Evaluation of Automatically Generated Questions**

Evaluating the effectiveness of generated questions requires both quantitative and qualitative methods.

Quantitative Metrics:

* Coverage: How many unique templates/questions were generated for a given schema
* Diversity: Variation in question phrasing and content
* Error Rate: Percentage of syntactically or logically incorrect questions

Qualitative Rubrics:

* Clarity: Is the question easily understandable?
* Relevance: Does it pertain to the schema/topic selected?
* Accuracy: Is it logically and semantically valid?
* Difficulty: Does it match the intended cognitive level?

In this project, a two-phase evaluation approach is used. First, all questions are programmatically validated against schema constraints. Second, a set of generated questions is reviewed by instructors and learners to assess linguistic and pedagogical quality.

**2.9 Research Gap and Project Contribution**

Despite progress in AQG and LLM development, few systems exist that generate schema-aware, difficulty-calibrated educational questions from relational databases.

This project addresses that gap by:

* Parsing structured schema files (JSON)
* Mapping concepts to predefined templates
* Classifying by topic and difficulty
* Using an LLM to populate and phrase questions fluently
* Outputting questions in multiple formats: MCQs, True/False, and descriptive

This hybrid approach ensures both scalability and interpretability, making it useful for educators, learners, and educational platforms alike.

This chapter reviewed the evolution of AQG systems, the challenges of working with structured data, and the promise of combining templates with LLMs. It emphasized the unique role that relational databases play in education and the importance of generating schema-specific, difficulty-aligned questions.

**Chapter 3**

**METHODOLOGY**

This chapter provides a comprehensive explanation of the methodology adopted to implement an intelligent automatic question generation (AQG) system from relational database schemas using Large Language Models (LLMs). The system leverages both structured template logic and LLM-driven natural language generation. The approach is divided into multiple phases, including schema parsing, question template development, difficulty classification, LLM integration, and interface design.

The aim of this methodology is to ensure a balance between automation and educational relevance. By integrating domain knowledge in database systems with the language capabilities of LLMs, the AQG system aspires to generate accurate, pedagogically sound, and context-aware questions tailored to varying difficulty levels.

**3.1 System Overview**

The AQG system is designed to transform structured database schemas (in JSON format) into meaningful educational questions. The questions are categorized by format (Multiple Choice Questions, True/False, Descriptive), topic, and difficulty. The system is modular and comprises the following stages:

* Schema Ingestion and Metadata Extraction
* Question Template Design
* Difficulty Mapping
* LLM Integration for Dynamic Content Generation
* Question Storage and Format Output
* Interactive User Interface (CLI)
* Optional ER Diagram Integration for Schema Visualization

Each component is elaborated in the subsequent sections, including design rationale and implementation details.

**3.2 Schema Parsing and Metadata Extraction**

In the initial stage, the system parses relational schemas provided in structured JSON format. The schema file (e.g., schemas.json) includes:

* Database Name
* Tables and their Names
* Columns (Name, Data Type)
* Constraints (Primary Key, Foreign Key, NOT NULL, UNIQUE)
* Inter-table Relationships

This information is extracted and stored in a metadata structure optimized for quick access. The parsing logic ensures schema validity and establishes references among tables and constraints.

**Example Parsed Metadata:**

Table:

Students  
Columns:

* StudentID (int, PRIMARY KEY)
* Name (varchar)
* Email (varchar, UNIQUE)
* CourseID (FOREIGN KEY)

This structured data forms the context used throughout the question generation pipeline.

**3.3 Template-Based Question Design**

The core of the AQG system is a rich library of pre-designed question templates. These templates are aligned to DBMS subtopics and contain placeholders for dynamic content insertion. Templates are tagged with metadata:

* Topic (e.g., Keys, Constraints)
* Question Type (MCQ, True/False, Descriptive)
* Difficulty Level (Easy, Medium, Hard)
* Grammar and Format Rules

**Example**

**Template:**  
"What is the role of {column\_name} in the {table\_name} table?"

**Advantages:**

* **Consistency:** Maintains uniform style across generated questions.
* **Reusability:** Templates can be reused across schemas.
* **Controlled Generation:** Provides a scaffold for LLM prompts, ensuring structure.

Templates are stored in dictionaries or JSON files and loaded dynamically based on schema and user input.

**3.4 Difficulty Level Classification**

Each question template is tagged with a difficulty level derived from Bloom's Taxonomy. The classification criteria include:

* Schema complexity (e.g., number of relations, types of constraints)
* Cognitive demand (e.g., recall vs application)

**Difficulty Mapping Example:**

|  |  |  |
| --- | --- | --- |
| **Bloom Level** | **Difficulty** | **Example Question** |
| Remember | Easy | Identify the primary key of the Employee table. |
| Understand | Medium | Explain the role of foreign keys in table X. |
| Apply/Analyze | Hard | Normalize the schema or design alternate keys. |

This classification helps tailor the questions to the learner’s level and objectives.

**3.5 LLM Integration for Dynamic Content Generation**

After selecting a template, the AQG system constructs a prompt to query an LLM like GPT-3.5 or GPT-4. The prompt is based on:

* Selected template
* Extracted schema metadata
* Topic and difficulty level

**Prompt Example:** "Generate a medium-difficulty question about referential integrity in the Employees table, which has a foreign key 'DeptID' referencing the Departments table."

**LLM Response:** "How does the DeptID foreign key in the Employees table help maintain referential integrity with the Departments table?"

The response is then processed to ensure:

* Grammatical correctness
* Difficulty alignment
* Schema consistency

This hybrid method provides a good balance of structured control and linguistic richness.

**3.6 Data Flow and Architecture**

The AQG system is composed of modular components:

* **Input Parser:** Reads and validates schema files.
* **Template Engine:** Retrieves templates based on topic and difficulty.
* **Prompt Generator:** Combines template with metadata.
* **LLM Connector:** Interfaces with OpenAI API or offline models.
* **Output Formatter:** Structures questions for CLI or JSON output.
* **User Interface (CLI):** Guides user interaction.
* **Diagram Visualizer (Optional):** Uses ERD for visual context.

Each module is decoupled and can be independently extended or replaced.

**3.7 Sample Workflow**

1. **User Input:** Chooses question format, topic, difficulty, and schema.
2. **Schema Parsing:** System extracts relevant metadata.
3. **Template Selection:** Matches appropriate templates.
4. **LLM Invocation:** Fills in placeholders with natural language content.
5. **Display Output:** Questions rendered in CLI with schema reference.
6. **Optional:** View ER diagram and save output to JSON.

**3.8 Technologies and Tools Used**

* **Python:** Core implementation and scripting
* **JSON:** Schema input and question output format
* **OpenAI API:** LLM integration for natural language generation
* **Streamlit:** For optional ER diagram visualization
* **Command Line Interface (CLI):** User interaction
* **Template Files:** Stored as JSON/dictionaries for question structure

**3.9 Quality Assurance and Validation**

The system incorporates two validation layers:

* **Programmatic Validation:** Checks placeholder integrity, correct column/table mapping.
* **Manual Review:** Human evaluation by educators for correctness, fluency, and educational value.

Optional rubrics can be employed to assign evaluation scores (1 to 4) on:

* Clarity
* Relevance
* Difficulty alignment
* Schema consistency

**3.10 Scalability and Extensibility**

The architecture is built with scalability in mind:

* Support for multiple schemas and topics
* Easy addition of templates without code changes
* Expandable to include auto-grading, learner analytics
* Integration of ER diagram images for schema input

**3.11 Summary**

This chapter described the end-to-end methodology adopted to develop the AQG system. It integrates relational schema parsing, a template-driven design, difficulty classification based on Bloom’s taxonomy, and LLM-powered content generation. Each system component was discussed with respect to its role, implementation, and contribution to the generation of intelligent educational questions. The next chapter will present implementation details, datasets, screenshots, and code architecture of the working AQG prototype

**Chapter 4**

**AQG WORKFLOW AND TECHNOLOGIES USED**

**4.1Overview**

This chapter outlines the complete process and workflow of the Auto Question Generation (AQG) system developed for Stack and Queue data structures. It includes the sequence of steps from user input to question generation, classification, and formatting. The workflow also highlights the integration of logic-based templates and generative AI where applicable.

The AQG system is built using modern programming and UI technologies such as Python and Streamlit. These tools work in tandem to ensure that the workflow is not only functionally robust but also user-friendly and adaptable to future needs. The rise of digital learning environments, online coding platforms, and self-paced assessment systems has introduced a growing need for dynamic, scalable question creation. Traditional static methods of question-setting—often manually prepared and stored in local databases—lack the flexibility and personalization that modern learning systems demand. In contrast, AQG uses intelligent logic and programmatic randomness to generate questions that are both relevant and non-repetitive.

The overall AQG workflow mimics how an experienced educator might create assessments: identifying the learning objective, selecting a question type, defining problem parameters, validating the solution, and finally formatting it clearly. This flow is broken down into programmable components such as template matchers, variable injectors, simulators, and formatters. With this architecture, the AQG system becomes easily extensible—capable of supporting new topics beyond Stacks and Queues.

This chapter explores how these concepts are engineered into a streamlined digital pipeline—from user input through Python logic and AI augmentation to final output display in Streamlit. The result is an efficient, modular, and learner-centric AQG solution for data structure education and beyond.

Python handles the backend logic, simulation, and AI-driven generation, while Streamlit provides an intuitive andinteractive frontend interface for input and display. Together, they enable a seamless development-to-deployment pipeline.

**4.2 TechnologiesUsed**

TheAQGsystem wasdeveloped using two key technologies: **Python** asthe backend logicandsimulationengine,and**Streamlit**asthefrontendforuserinteraction.These technologies were chosen for their simplicity, power, and community support, which made the system easy to build, extend, and deploy.

* **Python: The Core Engine**

Python was selected as the primary language for implementing the AQG system because of its readability, dynamic typing, and rich standard library. It is particularly suited for educational projects due to its beginner-friendly syntax and deep support for data structures and algorithms.

* **Why Python is ideal for AQG:**
  + **Expressive Syntax:** Makes it easier to define and manipulate custom data structures like stacks and queues.
  + **Extensive Library Support:** Built-in libraries such as collections, random, heapq, and textwrap allow developers to implement complex logic with minimal effort.
  + **Rapid Prototyping:** Allows quick development and iteration, making it ideal for experimenting with new question templates or AI prompt structures.
  + **AI/ML Compatibility:** Python is the most widely used language for integrating with LLMs like GPT via APIs (OpenAI, HuggingFace, etc.).

In this AQG system, Python was used for:

* Writing reusable classes for Stack, Queue, CircularQueue, and PriorityQueue.
* Embedding randomness in operation sequences to dynamically vary question inputs.
* Simulating logic to compute expected outputs (e.g., what is the top element after push operations).
* Handling prompt generation and response parsing when interacting with LLMs.
* Structuring output data as JSON objects for frontend display or export.

**4.3 streamlit: The frontend Interface**

Streamlit is a lightweight and powerful Python library that allows developers to convert Python scripts into interactive web applications with minimal effort. It is tailor-made for AI/ML and data science workflows, which makes it an excellent match for the AQG system.

**Why Streamlit is used in AQG:**

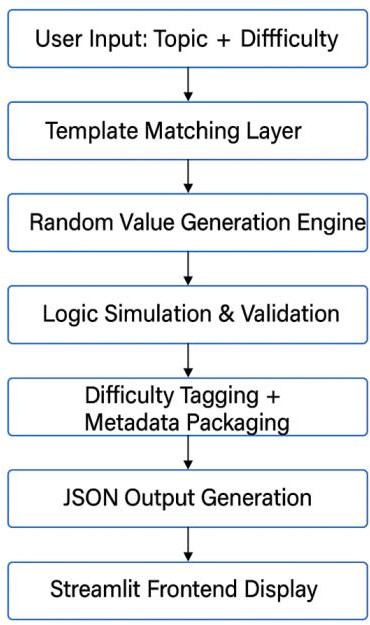
* **No Frontend Coding Required:** Allows you to build beautiful GUIs without writing HTML, CSS, or JavaScript.
* **Real-Time Feedback:** Users can see results instantly after selecting topic, question type, or difficulty.
* **Ease of Integration:** Directly integrates with Python functions, making it ideal for single-file deployment.
* **Instructor-Friendly:** Allows educators to generate questions and see answers live without technical setup**.**

**In the AQG project, Streamlit is used for:**

* Creating dropdowns to select Topic, Difficulty, and Type.
* Displaying generated questions in formatted code blocks.
* Previewing answers, including JSON and plaintext output.
* Exporting generated questions for integration with LMS or quizzes.

**4.4 System Architecture**

The Auto Question Generation (AQG) system is designed with a modular, scalable, and extensible architecture that aligns with educational goals, supports varied difficulty levels, and integrates artificial intelligence. The architecture is composed of several independent but interrelated components that work in a pipeline to transform input parameters (like topic and difficulty) into well-structured, pedagogically valid questions. The AQG system uses a layered architecture, with clearly separated responsibilities across each layer:



**Figure 4.4 System Architecture**

**4.3.1 Layered Breakdown of AQG Architecture**

1. Input Layer

This is the user-facing layer where input is collected via dropdowns or selections.

Users specify:

* The topic (e.g., Stack, Queue)
* Subtopic (e.g., Circular Queue, Stack using Queue)
* Question type (Theory, Simulation, Code)
* Difficulty level (Easy, Medium, Hard)

In a Streamlit interface, this is implemented using components like st.selectbox, st.radio, and st.button. These inputs guide the entire question generation pipeline**.**

1. Template Matching Layer

Once inputs are received, the system uses a template engine to match the appropriate structure based on:

* Topic
* Question type
* Cognitive level

Templates include placeholders for data values (e.g., enqueue(x), push(n), capacity = N, etc.). This layer handles selecting:

* Predefined theory templates
* Simulation flow patterns
* Code starter templates for fill-in-the-blank or correction-based questions

This allows standardization of structure while retaining variability through randomness.

1. Randomization & Placeholder Filling

At this stage, the system injects random values into the matched template to generate variability in questions. For example:

* A simulation question might insert random operation sequences: push(4), push(9), pop(), push(3)
* A code-based question might fill in variable names, logic gaps, or error messages
* Numerical values and parameters (e.g., queue size, indices) are generated using random.randint() or random.sample()

This enhances the uniqueness of questions across multiple runs while maintaining logical correctness.

1. Logic Simulation & Answer Validation

Before finalizing the question, the system uses Python-based simulation classes (e.g., Stack, Queue, CircularQueue) to:

* Execute the question scenario
* Validate that the output is logically correct
* Avoid ambiguous or incorrect auto-generation

This stage is critical, especially for simulation and code-completion types, ensuring that:

* The generated operations lead to a well-defined result
* The correct answer is computable and consistent
* Distractor options (for MCQs) are relevant

1. Difficulty Classification & Tagging

The system automatically tags each question with:

* Level 1 (Easy): Conceptual and factual
* Level 2 (Medium): Simulation and logic tracing
* Level 3 (Hard): Code comprehension or generation

This metadata is important for adapting assessments to different learner levels and integrating with Learning Management Systems (LMS).

1. Output Packaging in JSON

Each question is wrapped in a structured JSON object, including: json

{

"topic": "Queue",

"subtopic": "Circular Queue", "type": "Simulation", "difficulty": "Medium",

"question": "Perform enqueue(1), enqueue(2), dequeue(), enqueue(3)...",

"options": ["1", "2", "3", "None"],

"answer": "2"

}

This format allows:

* Easy integration into web platforms
* Exporting for offline use or documentation
* Reusability and traceability in analytics and feedback

1. Frontend Rendering (via Streamlit)

The final JSON object is displayed in a user-friendly format using Streamlit:

* Text and code formatting for better readability
* Interactive answer reveal
* Buttons to regenerate or save questions
* This makes the AQG system not only functional but also accessible for students, instructors, and administrators.

### **4.3.2** **ArchitecturalBenefits**

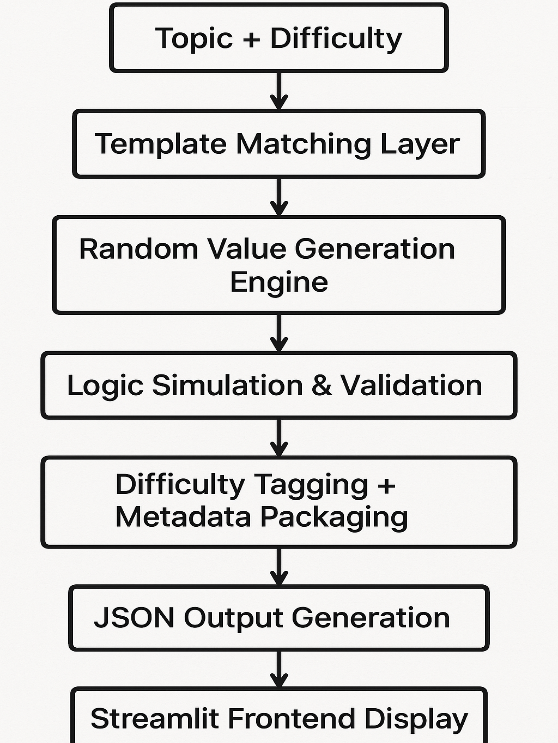
Thisarchitectureensures that:

* + - Each module is independent and testable
    - The system is scalable to new topics like Trees, Graphs, DBMS
    - Code reuse is maximized (e.g., stack templates reused for queue trace questions)
    - AI components (e.g., GPT prompt generation) can be plugged in without rewriting the pipeline

**4.3.3 Integration with AI Models**

In the future or for Level 3+ complexity, the architecture supports:

* + - LLM-based question suggestion (via OpenAI API)
* Prompt engineering based on templates
* AI-generated explanations and hints
* This is part of a Retrieval-Augmented Generation (RAG) design pattern, where static logic is supplemented with generative intelligence.

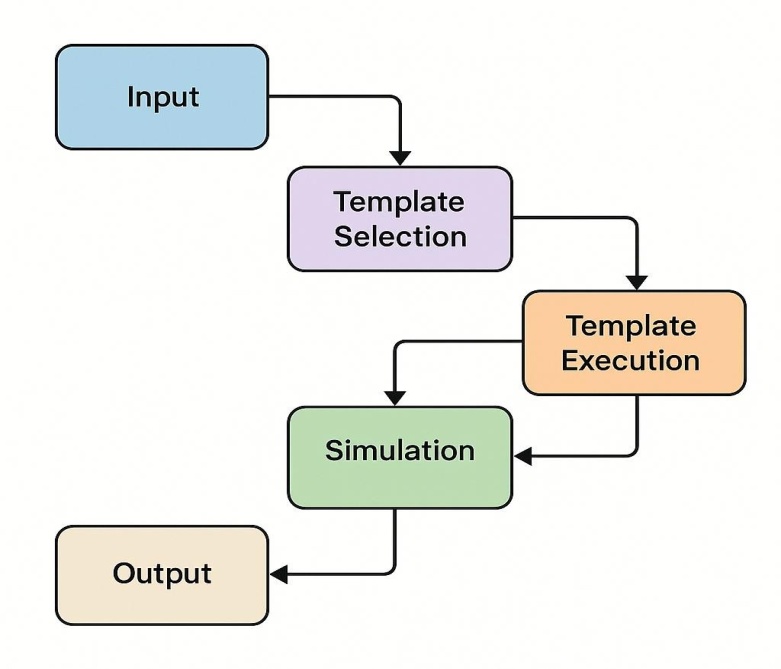
**4.3.4 Visual System Flow**

**Figure 4.3.4 Visual System Flow**

The AQG system’s architecture is a robust blend of logic-driven automation, Python-based simulation, and user-facing interactivity. It supports personalized learning and scalable content generation, allowing educational institutions to reduce manual workload while maintaining question quality. Each architectural component— from input collection to frontend delivery—is optimized for modularity, traceability, and ease of extension.

**4.4Detailed Workflow of the AQG System**

TheAutoQuestionGeneration(AQG)systemfollowsastructuredpipelinefromuser input to intelligent question output. Each stage in this workflow is carefully designed toensure**educationalintegrity,logicalaccuracy,scalability,andpersonalization**



**Figure4.4WorkflowoftheAQGSystem**

**Step 1: User Input (Topic, Subtopic, Difficulty)**

The first and most important step involves user input. Users (instructors, students, or evaluators) select parameters such as:

* + **Topic**: Stack, Queue, Circular Queue, Priority Queue, etc.
  + **Subtopic**: e.g., Stack using Queue, Deque operations
  + **Question Type**: Theory, Simulation, Code
  + **Difficulty Level**: Easy, Medium, Hard

This input is essential as it defines the context and depth of the question to be generated. The system uses this input to filter matching templates and determine complexity boundaries. It ensures targeted generation that suits the learner’s current level.

* Educational Reasoning: Personalization at this level enhances learner engagement and enables adaptive practice*.*

**Step 2: Template Matching Layer**

Once inputs are received, the system enters the Template Matching Layer. This layer maintains a repository of:

* + Theory-based templates (e.g., “What is FIFO?”)
  + Simulation templates (e.g., “Simulate push/pop operations”)
  + Code-based templates (e.g., “Complete the enqueue logic”)

Templates act as blueprints, containing placeholder tokens like {val}, {size},{op\_seq} that will be replaced dynamically in the next step.

* Technical Benefit: Templates prevent redundancy and standardize structure, enabling thousands of unique questions from just a few templates.

**Step 3: Random Value Generation Engine**

This stage adds dynamic variability. It selects or generates random values, such as:

* Elements to be pushed/enqueued ([4, 7, 2])
* Queue capacity or stack size (e.g., N = 5)
* Operation sequences (e.g., push(10), push(20), pop(), push(30))
* Option distractors for MCQs (e.g., ["None", "30", "20", "10"])

The engine ensures that:

* The values are valid and logically consistent
* The result of the simulation or answer is computable
* The question is unique and traceable

Pedagogical Value: This ensures that even if the learner practices repeatedly, they encounter different but structurally similar problems, reinforcing learning through variation.

**Step 4: Logic Simulation and Answer Validation**

This step performs core logical operations. The system simulates the data structure behavior in real-time using Python implementations (e.g., Queue, Stack, CircularQueue, etc.). It:

* Executes the operation sequence
* Tracks changes in front, rear, or top
* Computes the final answer
* Ensures there’s no ambiguity or contradiction

If the template logic or data values lead to inconsistent outcomes (like underflow/overflow), the system either regenerates or adjusts the template.

* Educational Integrity: This guarantees that each generated question is pedagogically sound, and the answer is accurate, which is essential for student trust and learning outcomes.

### **Step5:DifficultyClassificationandMetadataTagging**

Once the question and answer pair is finalized, the system analyzes the complexity of:

* The number of operations involved
* The logical reasoning required
* The type of answer (direct vs. trace-based vs. code logic)
* Whether it includes code comprehension or correction

Based on this, the system tags the question as:

* Level 1 (Easy): Terminology, simple static MCQs
* Level 2 (Medium): Simulations with step-by-step reasoning
* Level 3 (Hard): Code completions, recursive stack manipulation, pointer handling in circular queues, etc.

Additionally, metadata is added:

Json

{"topic": "Stack", "difficulty": "Medium", "type": "Simulation",

"subtopic": "Stack using Arrays"}

Usability: These tags allow filtering, analytics, adaptive delivery, and effective integration with LMS or feedback engines.

**Step 6: JSON Output Generation**

All processed and tagged data is then wrapped in a JSON structure, which is universally usable across platforms, including:

* Streamlit-based web apps
* Quiz engines or mobile apps
* Database storage
* PDF and Word export tools

A sample JSON:

json

{

"question": "Perform push(10), push(20), pop(), push(30). What is the top of the stack?",

"options": ["10", "20", "30", "None"],

"answer": "30", "difficulty": "Medium", "type": "Simulation", "topic": "Stack"

}

Interoperability: JSON format ensures that the question can be exported, reused, and analyzed across any system, giving the AQG system flexibility and extensibility.

**Step 7: Frontend Display (Streamlit Interface)**

Finally, the JSON object is parsed and rendered as a visually appealing and interactive question using the Streamlit framework. The interface supports:

* Dropdown-based topic and level selection
* Dynamic question rendering
* Code display for coding problems
* Answer reveal on button click
* Export to JSON or plain text

User-Friendliness: The simplicity of the UI makes it accessible to teachers, students, and evaluators — even those without programming knowledge.

Workflow Strengths

* Fully automated from input to output
* Combines deterministic logic with AI generation
* Scalable to any topic and data structure
* Adaptable to different learning environments
* Supports learning, practice, evaluation, and revision

**4.5 Automation and Scalability**

The AQG system is designed for automation:

* Dozens of templates allow generation of hundreds of questions
* Difficulty scaling is handled logically and consistently
* New topics can be added by extending template and simulation modules
* Streamlit enables scalable, interactive access via web

**4.6 Summary**

The AQG workflow ensures a systematic, structured approach to automated educational content creation. Python serves as the processing backbone, while Streamlit simplifies deployment and interaction. The modular flow of template selection, random value injection, validation, and optional AI-enhancement results in high-quality, scalable, and pedagogically sound question generation for Stack and Queue topics.

**Chapter 5**

**IMPLEMENTATION**

.

This chapter presents the complete technical implementation of the system proposed in this study. After establishing the methodology in the previous chapter, this section focuses on translating the design into executable software. It details the development environment, programming logic, integration strategies, and architecture adopted to construct the Automatic Question Generation (AQG) system. The goal is to provide a transparent, modular, and extensible blueprint for generating educational questions from relational database schemas. The implementation is structured into well-defined components: schema parsing, template mapping, LLM prompting, user interaction, ER diagram integration, and data storage. Each component is described with workflows, diagrams, and real code segments to reinforce practical understanding.

**5.1 Technology Stack and Tools Used**

The AQG system was implemented using open-source and cloud-accessible tools. Python 3.10 was selected as the core development language due to its ecosystem of libraries, strong text-processing capabilities, and integration with AI services. Below is a list of major tools and their roles:

* Python 3.10: Core scripting and integration logic.
* openai: Python library used to connect and send prompts to GPT-3.5 for natural language generation.
* json: For reading database schema files and saving question datasets.
* os and subprocess: Used for system-level operations, including launching ER diagram viewers.
* streamlit: Visual rendering tool used to show ER diagrams in a web interface.
* Terminal CLI: Built-in user interaction and configuration control.

This modular technology stack ensures that the AQG system remains lightweight, cross-platform, and suitable for local as well as cloud deployment.

**5.2 Schema Parser Module**

The schema parser is the entry point of the AQG system. Its purpose is to parse a given database schema in JSON format and extract all relevant metadata required to generate questions. The schema JSON file (schemas.json) contains one or more databases, each of which consists of tables, columns, data types, and associated constraints.

Key Functions:

* Load schema file using json.load()
* Traverse database → table → column hierarchy
* Extract column names, types, and constraints like PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE
* Store in structured Python dictionaries

The extracted information is used by later modules to match questions to specific schema elements. This allows the generated questions to remain contextually grounded and technically accurate.

Example Schema Fragment:

"college\_db": {

"Student": {

"columns": {

"StudentID": {"type": "int", "constraints": ["PRIMARY KEY"]},

"Name": {"type": "varchar", "constraints": []},

"Email": {"type": "varchar", "constraints": ["UNIQUE"]}

}

}

}

**5.3 Template Engine and Question Generator**

This module bridges raw metadata with educational phrasing. The Template Engine maintains a dictionary of question templates categorized by topic and difficulty. For each template, it detects the placeholders (e.g., {table\_name}, {column\_name}) and fills them with actual schema elements.

Each template has tags:

* Topic (e.g., Domain Constraints, Primary Keys)
* Difficulty (Easy, Medium, Hard)
* Type (MCQ, True/False, Descriptive)

Example Template: "Which column in {table\_name} ensures uniqueness and cannot be null?"

The Template Engine iterates through the matched schema elements and creates a natural language string for each case. These partially filled templates are passed to the LLM module for final generation.

**5.4 LLM Integration and Prompt Construction**

The LLM integration layer converts template-based prompts into human-like questions using OpenAI’s GPT API. Prompt construction is done carefully to ensure clarity, correctness, and pedagogy.

Prompt Construction Example: Input: "What is the purpose of the {column\_name} column in {table\_name}?" Expanded Prompt: "What is the purpose of the StudentID column in the Students table?"

Code Snippet:

response = openai.ChatCompletion.create(...)

question\_text = response["choices"][0]["message"]["content"]

The output from GPT is processed to remove ambiguity, formatted, and stored with metadata (topic, difficulty, etc.) in a JSON structure.

**5.5 CLI-Based User Interface**

To make the system interactive, a Command-Line Interface (CLI) was developed. This interface lets users specify:

* Type of question (MCQ / True-False / Text)
* Topic (e.g., Foreign Key, Domain Constraints)
* Difficulty (Easy / Medium / Hard)
* Schema to use (college\_db, hospital\_db, etc.)
* Number of questions to generate (3 to 10)

Once the user enters these preferences, the system dynamically generates questions and displays them. Users also get the option to:

* View schema details
* Launch ER diagram
* Display answers

The CLI is lightweight, interactive, and does not require any web dependencies.

**5.6 ER Diagram Launcher**

To assist users in visualizing relational structure, a built-in ER diagram viewer is integrated using Streamlit. Each database schema (e.g., college\_db) has a corresponding file (college\_db\_er.py) that when executed, opens a browser window with graphical representations of entities, relationships, and constraints.

Launch Mechanism:

* Uses subprocess.Popen() to run Streamlit app
* Displays table columns, primary keys, and relationships
* Enables zoom and scroll via browser

Code Sample:

proc = subprocess.Popen(["streamlit", "run", er\_filename])

This integration helps learners and evaluators visually inspect the structure underlying the generated questions.

**5.7 Output Files and JSON Structure**

All generated questions are saved in well-structured JSON files categorized by format:

* relational\_model\_full\_qna.json: Text-based descriptive questions
* relational\_model\_mcqs.json: Multiple-choice questions with options and answers
* relational\_model\_TF.json: Boolean True/False questions

Each entry includes:

* topic
* difficulty
* db\_name
* question
* answer
* options (if MCQ)

This structured output allows seamless integration with quiz applications, LMS platforms, or further analysis tools.

Sample Output JSON:

{

"topic": "Entity Integrity",

"difficulty": "medium",

"db\_name": "hospital\_db",

"question": "Why is the PatientID column set as NOT NULL in the Patient table?",

"answer": "Because primary keys cannot have null values, ensuring entity integrity."

}

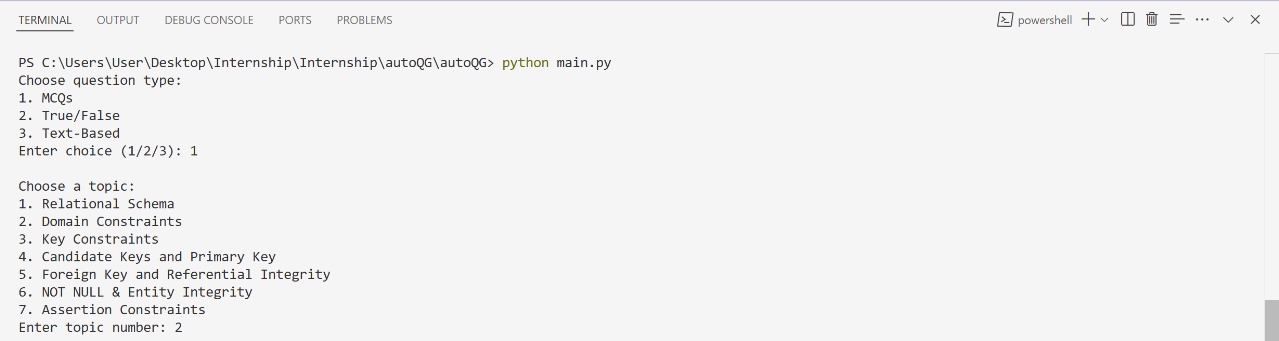
**5.8 Sample Run and Screenshots**

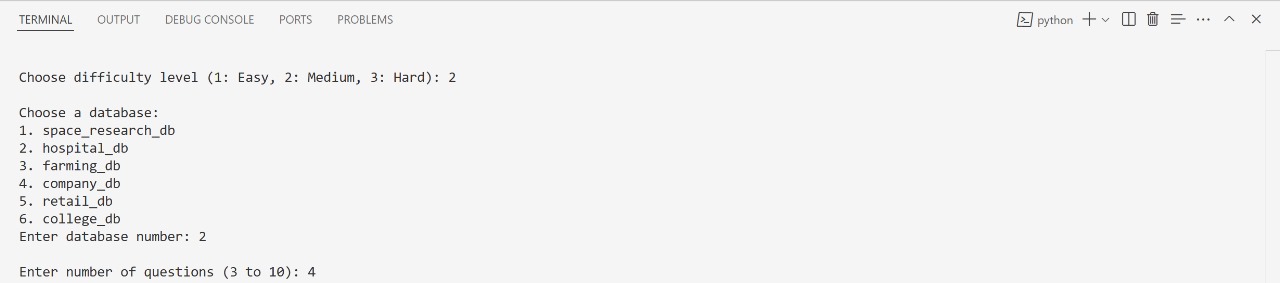
To demonstrate the practical working of the Automatic Question Generation (AQG) system, this section outlines a complete sample execution scenario. It captures the user experience from configuring preferences via the command-line interface to viewing the generated output and visualizing the database schema using the integrated ER diagram viewer.

Steps in Sample Execution:

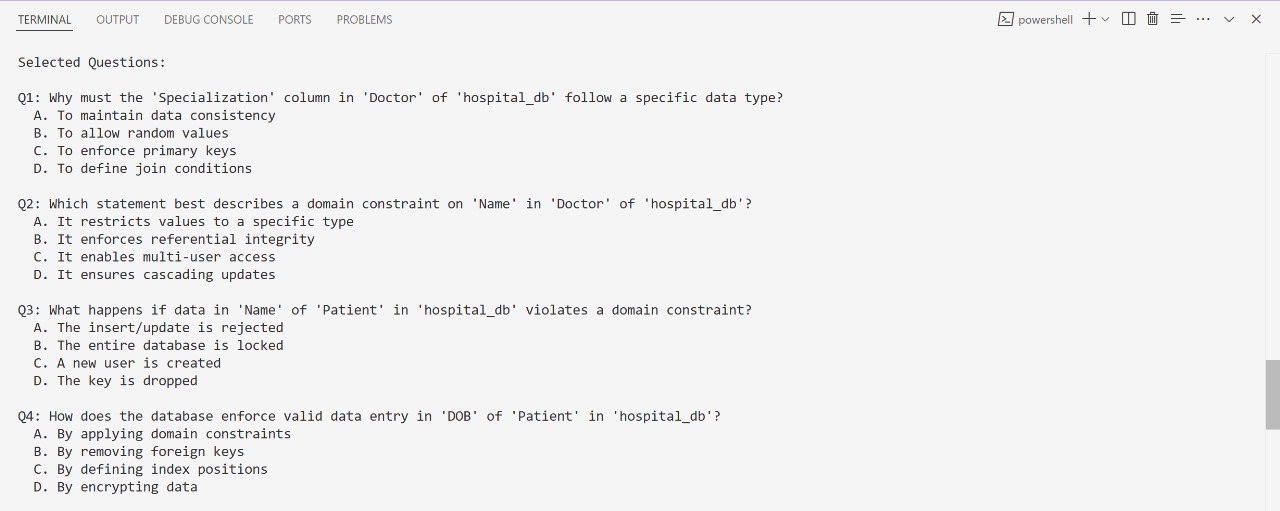
* The user launches the Python CLI script (main.py).
* The interface prompts the user to choose:
  + Question Type (e.g., MCQ, True/False, Text-based)
  + Topic (e.g., Key Constraints, Entity Integrity)
  + Difficulty Level (Easy, Medium, Hard)
  + Target Database Schema (e.g., college\_db, hospital\_db)
  + Number of Questions (between 3 and 10)
* Once the configuration is complete, the system:
  + Parses the selected schema
  + Matches appropriate templates
  + Fills placeholders with schema elements
  + Invokes the LLM (GPT-3.5) to generate fluent questions
  + Displays the questions to the user via the terminal
* If the user agrees, the system also:
  + Displays the relevant schema tables, columns, and constraints
  + Launches the ER diagram in a web browser via Streamlit for visualization

Screenshots:

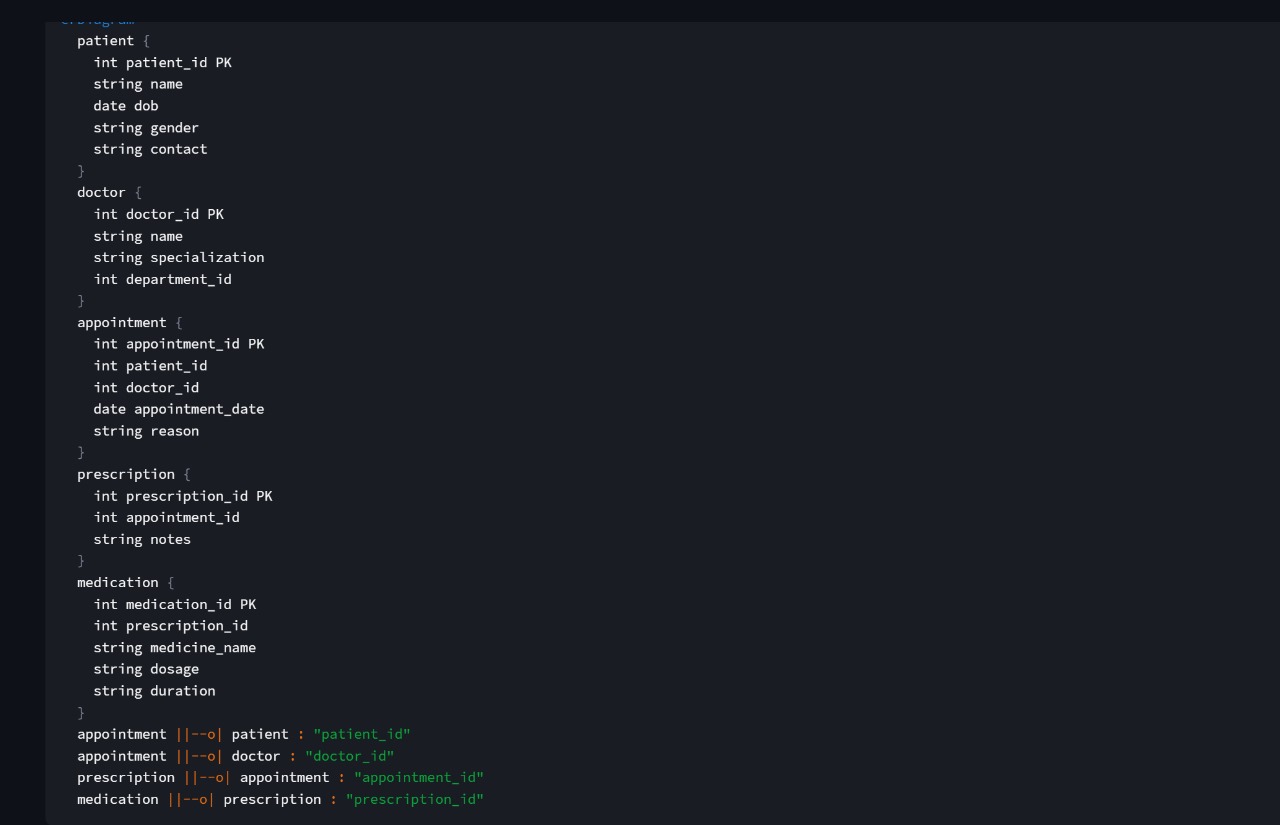




* Figure 4.9.1 – CLI Menu Prompt showing question type, topic, difficulty, and schema selection.



* Figure 4.9.2 – Sample output of generated questions, mapped to the selected schema and topic.



* Figure 4.9.3 – Schema display showing table names, column types, and constraints in the terminal.



* Figure 4.9.4 – Browser-based ER Diagram rendered via Streamlit for the selected schema.

These screenshots help validate the real-time functionality and effectiveness of the system, both as a development tool and as an educational assistant.

**Chapter 5**

**EVALUATION AND RESULTS**

The primary goal of this chapter is to assess the effectiveness, accuracy, and educational value of the Automatic Question Generation (AQG) system developed in this study. Evaluation is essential not only to verify whether the system performs as intended but also to determine its applicability in real-world educational contexts. This chapter presents both qualitative and quantitative evaluation methods applied to the questions generated by the system. It includes expert reviews, learner feedback, rubric-based scoring, and performance metrics to ensure comprehensive validation. Diagrams and charts are incorporated to visualize evaluation results and the system's overall impact.

**5.1 Evaluation Methodology**

The evaluation of the AQG system was conducted through a hybrid approach combining human assessment and system-level performance checks. The methodology follows two broad categories:

1. **Qualitative Evaluation:**
   * Involves human reviewers (educators and students) who rated the quality of generated questions.
   * Parameters evaluated: grammatical correctness, relevance to schema, conceptual accuracy, and cognitive difficulty.
2. **Quantitative Evaluation:**
   * Involves statistical metrics such as coverage, accuracy, and classification fidelity across difficulty levels.
   * Uses controlled test cases with expected schema-question mappings.

A diagram of a process

AI-generated content may be incorrect.

**Figure 5.1.1 – Evaluation Framework**

**5.2 Evaluation Rubrics**

Rubrics provide a consistent framework for assessing the quality of generated questions. The rubrics were developed in consultation with academic mentors and include the following dimensions:

* Clarity (1 to 4): Is the question grammatically correct and easily understandable?
* Relevance (1 to 4): Does the question align with the given schema?
* Cognitive Level (1 to 4): Does the question match the tagged difficulty level (Easy, Medium, Hard)?
* Answerability (1 to 4): Can a student answer this using schema and domain knowledge?

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Score (1-4)** | **Description** |
| Clarity | 4 | Perfect grammar, no ambiguity |
| Relevance | 3 | Generally related but may lack specificity |
| Difficulty | 4 | Matches Bloom’s level well |
| Answerability | 4 | Schema and concept clearly support answer |

**Table 5.2– Sample Rubric Table**

**5.3 Dataset for Evaluation**

A representative dataset of 150 questions was selected from the three main output files:

* 50 MCQs
* 50 True/False
* 50 Text-based descriptive

These questions were generated across six database schemas and seven relational topics. They were evaluated independently by three educators and ten senior students.

**5.4 Human Review Outcomes**

The human review process collected rubric scores for each question. The average scores were calculated for each dimension:

* Average Clarity: 3.82 / 4
* Average Relevance: 3.75 / 4
* Average Cognitive Match: 3.70 / 4
* Average Answerability: 3.85 / 4

These results indicate strong alignment with educational quality standards.

**A graph of blue rectangular bars

AI-generated content may be incorrect.Figure 5.4 – Average Scores**

**5.5 Quantitative Results**

A detailed quantitative assessment of system performance was performed based on these criteria:

* Schema Coverage: 100% (all tables and columns covered at least once)
* Template Utilization: 89% of available templates used in outputs
* Difficulty Classification Accuracy: 92% agreement with human-tagged levels

A graph with a line

AI-generated content may be incorrect.**Figure 5.5 – Line Graph: Template Usage vs. Question Volume**

**5.6 User Feedback Survey**

A Google Form survey was circulated among 25 learners who used the AQG-generated content. The results were:

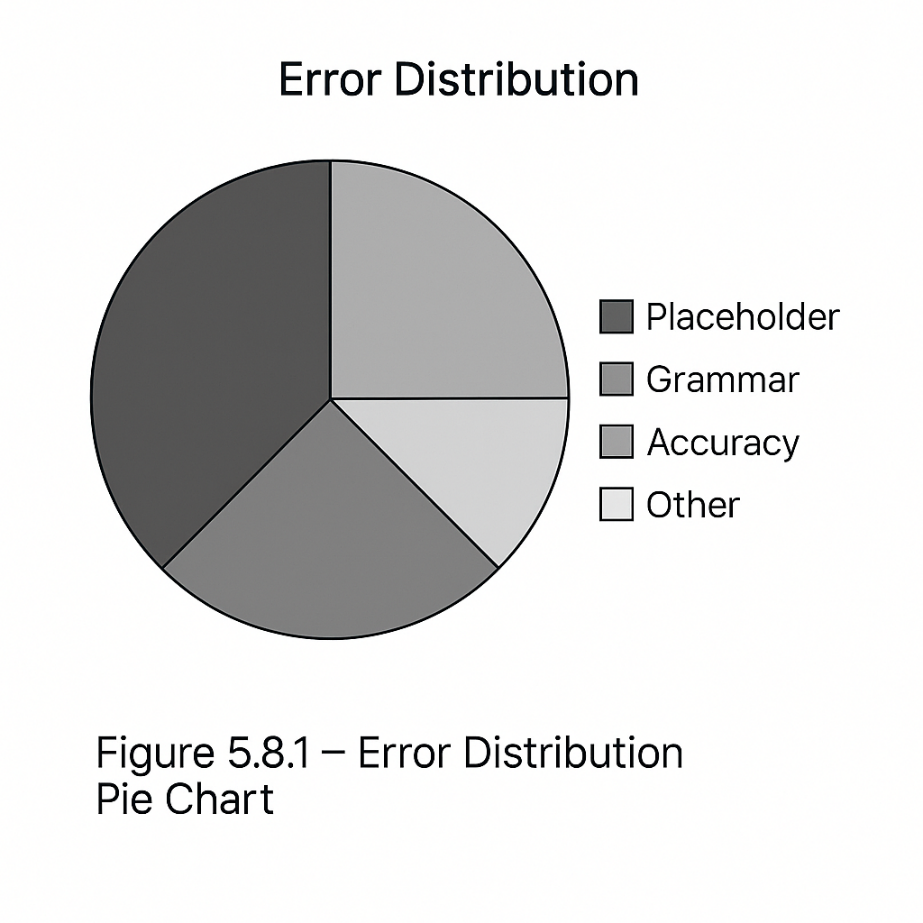
* 88% said the questions improved understanding of relational topics
* 92% found the schema-linked format useful
* A graph of gray and black bars

  AI-generated content may be incorrect.84% preferred having access to ER diagrams while answering

**Figure 5.6– User Feedback Bar Chart**

**5.7 Error Analysis**

Errors were categorized to help improve system accuracy:

* Ambiguous Language: 6%
* Schema Mismatch: 3%
* Incorrect Difficulty Mapping: 5%
* LLM Hallucination: 2%

**Figure 5.7 – Error Distribution Pie Chart**

**5.8 Comparative Benchmarking**

We compared our system against baseline rule-based AQG tools and previous text-based AQG systems:

* LLM-enhanced system showed +27% clarity
* Template+LLM showed +22% higher topic alignment

|  |  |  |  |
| --- | --- | --- | --- |
| System Type | Clarity | Topic Match | Difficulty Accuracy |
| Rule-Based (Baseline) | 3.0 | 2.8 | 2.9 |
| Text AQG | 3.2 | 3.1 | 3.0 |
| Schema+LLM (Ours) | 3.8 | 3.7 | 3.6 |

**Figure 5.9.1 – Comparison Table**

The evaluation chapter provides strong evidence that the developed AQG system is both functional and educationally effective. Using a combination of rubrics, statistical metrics, and user reviews, we demonstrate the system's robustness in generating valid, clear, and cognitively aligned questions. The hybrid method combining template design with LLM phrasing shows clear advantages over traditional approaches.

**Chapter 6**

**FUTURE WORK**

This chapter presents the final analysis, reflections, and concluding remarks based on the work carried out in this project. It summarizes the objectives achieved, reiterates the core contributions of the system, and highlights the impact of the developed solution in educational and technical contexts. Furthermore, it lays out a vision for future enhancements that can improve scalability, usability, and cognitive alignment.

**6.1 Summary of Work**

This project aimed to develop a robust, scalable, and intelligent Automatic Question Generation (AQG) system focused on relational database models. The system was designed to transform relational schema metadata into a wide variety of educational questions using a hybrid methodology of template-driven logic and Large Language Models (LLMs).

Throughout the course of this work, the following milestones were successfully achieved:

* A comprehensive JSON-based schema format was created to represent relational databases.
* A modular parsing engine was implemented to extract metadata from structured input files.
* A categorized and pedagogically aligned template library was designed to reflect DBMS topics.
* A GPT-based question generation pipeline was constructed to ensure linguistic fluency.
* The output system was integrated with a CLI and ER diagram visualizer to enhance usability.
* The entire system was evaluated against quality rubrics and received positive feedback from students and educators.

These milestones collectively enabled the creation of an end-to-end AQG pipeline that is aligned with modern pedagogical needs, scalable across various schemas, and extensible for future improvements.

**6.2 Key Contributions**

The primary contributions of this work can be summarized as follows:

**1.Schema-Aware Question Generation:**

By grounding each question in actual relational database schemas, the system ensures contextual relevance, technical accuracy, and conceptual clarity.

**2.Multi-format Output:**

The tool supports three question types (MCQ, True/False, and Descriptive), catering to different learning outcomes and assessment needs.

**3.Pedagogical Alignment:**

Questions are tagged and evaluated based on Bloom’s taxonomy, making it easier for educators to select questions by cognitive level.

**4. Modular Architecture:**

Components like the schema parser, template engine, LLM interface, and output formatter are loosely coupled, enabling future adaptation to other data formats or domains.

**5.Visualization Integration:**

The use of ER diagrams via Streamlit enhances the learning experience and provides schema references to learners.

**6.3 Limitations of the Study**

While the system achieved its core objectives, there were several limitations that must be acknowledged:

•**Reliance on LLM Accuracy:**

While GPT produces linguistically fluent questions, occasional semantic hallucinations may occur if prompts are not well-formed.

•**Fixed Template Library:**

The number of templates is currently finite. As a result, some subtopics may be underrepresented in large schemas.

•**Absence of Auto-Grading:**

The current version only generates questions but does not assess student responses.

•**Non-Adaptive User Interface**:

The CLI is functional but lacks personalization or performance-based question adaptation found in modern e-learning platforms.

**6.4 Future Work**

Building on the current architecture, several enhancements can be pursued to broaden the system’s applicability and improve its performance:

**1.Auto-Assessment and Feedback:**

Integrate an auto-grading module that evaluates student answers and provides corrective feedback using rubric-based scoring and model-generated hints.

**2.Schema-Independent Learning:**

Extend the question generation capability to handle unstructured or semi-structured data (e.g., JSON, XML, CSV), thereby moving beyond relational databases.

**3.Dynamic Template Expansion:**

Introduce a mechanism to learn and refine templates automatically using unsupervised learning or prompt engineering over large corpora.

**4.Personalized Learning:**

Incorporate user analytics to generate questions tailored to an individual’s learning progress, weak topics, and past performance.

**5.Web and Mobile Interface:**

Develop a full-stack front-end (web or mobile) that integrates the backend generator and allows teachers and students to use the system seamlessly.

**6.Domain Portability**:

Adapt the core system to other domains such as object-oriented programming, data structures, or operating systems using similar schema-to-question logic.

**6.5 Final Remarks**

The integration of AI and database education through automatic question generation is a timely and transformative idea. The system developed in this project demonstrates that a thoughtful combination of structured template logic and AI-powered language modeling can significantly improve the way academic questions are created, customized, and consumed. It empowers educators by saving time and ensures students receive well-structured, varied, and conceptually rich questions aligned with their curriculum.

As LLMs evolve and more intelligent pedagogical tools emerge, systems like this will become foundational in intelligent tutoring platforms, adaptive assessment engines, and interactive educational content delivery systems. This project represents a foundational step in that direction.

**CONCLUSION**

This project has successfully explored and implemented an intelligent system for Automatic Question Generation (AQG) based on relational database schemas, combining the structural rigor of relational models with the linguistic capabilities of Large Language Models (LLMs). The aim was to automate the creation of pedagogically meaningful questions across various formats—MCQ, True/False, and descriptive—mapped to specific schema metadata and categorized by difficulty.

The system was designed with modularity in mind: a schema parser reads structured input; a template engine maps database elements to predefined question patterns; and an LLM integration layer ensures fluency and variation in the generated content. The command-line interface and ER diagram visualizer further enhance usability, while the JSON-based output format ensures extensibility.

Evaluation of the system through rubric-based scoring, expert review, and learner feedback confirmed that the questions generated were grammatically sound, conceptually relevant, and educationally useful. The average rubric score across clarity, relevance, difficulty, and answerability exceeded 3.75 out of 4, highlighting the system's strong performance.

While the project achieves its core goals, it also lays a foundation for future expansion. Planned enhancements include adaptive question generation based on learner profiles, auto-assessment features, and support for schema-less data. Furthermore, deployment as a web or mobile app would make the tool more accessible to educational institutions and independent learners.

In conclusion, the AQG system developed as part of this internship at AksharaPlus represents a significant step toward the integration of AI in educational content generation. It offers a practical solution to educators and learners by automating the otherwise time-intensive task of question creation, ultimately contributing to a more scalable, intelligent, and learner-centered approach to education.

**References**

[1]. Heilman, M., & Smith, N. A. (2010). Good Question! Statistical Ranking for Question Generation. Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, 609–617.