Auto Question and Answer Generation on Dynamic Programming

BY

Ganesh Kondamwar
Soham Kashettiwar
Tejas Kotalwar

Under the Guidance

of

Dr. Bhagyashri S. Kapre

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Mahatma Gandhi Mission's College of Engineering, Nanded (M.S.)

Academic Year 2024-25

An Internship Report on

Auto Question and Answer Generation on Dynamic Programming

Submitted to

DR. BABASAHEB AMBEDKAR TECHNOLOGICAL UNIVERSITY, LONERE

for fulfillment of the requirement for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE & ENGINEERING

By

Ganesh Kondamwar
Soham Kashettiwar
Tejas Kotalwar

Under the Guidance
of

Dr. Bhagyashri S. Kapre

(Department of Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MAHATMA GANDHI MISSION'S COLLEGE OF ENGINEERING
NANDED (M.S.)
Academic Year 2024-25

Certificate

This is to certify that the internship entitled

"Auto Question and Answer Generation on Dynamic Programming"

being submitted by ***Mr. Ganesh Kondamwar, Mr. Soham Kashettiwar, Mr. Tejas Kotalwar*** *to the Dr. Babasaheb Ambedkar Technological University, Lonere, for the award of the degree of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work carried out by them under my supervision and guidance. The matter contained in this report has not been submitted to any other university or institute for the award of any degree.*

Dr. Bhagyashri S. Kapre

Guide

Dr. A. M. Rajurkar

H.O.D

Computer Science & Engineering

Dr. G. S. Lathkar

Director

MGM's College of Engg., Nanded

20 January 2025

Ganesh Rajeshwar Kondamwar
MGM College of Engineering
Nanded, Maharashtra, India

Dear Ganesh Rajeshwar Kondamwar,

Subject: Internship Offer – Adaptive Learning Research & Development

Congratulations! We are pleased to offer you a **research-based internship** at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

This internship presents a valuable opportunity to contribute to our ongoing research in *adaptive learning technologies* supporting K–12 and college-level education. You will collaborate remotely with AkharaPlus engineers and researchers, engaging in projects that directly impact how students learn through intelligent, personalized systems.

As an intern, you will:

- Work remotely on research and development tasks under the guidance of AkharaPlus staff engineers.
- Attend weekly status meetings, present your progress, and participate in technical discussions.
- Take initiative in solving problems, seeking feedback, and documenting your development.

After your internship, you are required to submit a comprehensive report summarizing:

- The work you performed,
- The new technical and professional skills you acquired, and
- Your reflections on transitioning from a university student to a professional software engineer.
- Upload your findings, including code, to a specified GitHub repository.

Your performance will be evaluated using a rubric developed by AkharaPlus, LLC. The final evaluation will be submitted to the **Dean of MGM College of Engineering** for academic purposes.

Please note that this is a **no-stipend internship**. Its value lies in the immersive, real-world

experience you will gain in a research-driven software engineering environment.

We thank you for choosing **AkharaPlus, LLC.** for your internship. We look forward to working with you and contributing to your professional advancement.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

+1 252 320 7431 info@AksharaPlus.org http://www.aksharaplus.org
AksharaPlus, LLC.8 The Green STE B, Dover, DE 19901 USA

20 January 2025

Soham Rajendra Kashettiwar
MGM College of Engineering
Nanded, Maharashtra, India

Dear Soham Rajendra Kashettiwar,

Subject: Internship Offer – Adaptive Learning Research & Development

Congratulations! We are pleased to offer you a **research-based internship** at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

This internship presents a valuable opportunity to contribute to our ongoing research in *adaptive learning technologies* supporting K–12 and college-level education. You will collaborate remotely with AkharaPlus engineers and researchers, engaging in projects that directly impact how students learn through intelligent, personalized systems.

As an intern, you will:

- Work remotely on research and development tasks under the guidance of AkharaPlus staff engineers.
- Attend weekly status meetings, present your progress, and participate in technical discussions.
- Take initiative in solving problems, seeking feedback, and documenting your development.

After your internship, you are required to submit a comprehensive report summarizing:

- The work you performed,
- The new technical and professional skills you acquired, and
- Your reflections on transitioning from a university student to a professional software engineer.
- Upload your findings, including code, to a specified GitHub repository.

Your performance will be evaluated using a rubric developed by AkharaPlus, LLC. The final evaluation will be submitted to the **Dean of MGM College of Engineering** for academic purposes.

Please note that this is a **no-stipend internship**. Its value lies in the immersive, real-world

experience you will gain in a research-driven software engineering environment.

We thank you for choosing **AkharaPlus, LLC.** for your internship. We look forward to working with you and contributing to your professional advancement.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

+1 252 320 7431 info@AksharaPlus.org http://www.aksharaplus.org

AksharaPlus, LLC.8 The Green STE B, Dover, DE 19901 USA

20 January 2025

Tejas Dnyaneshwar Kotalwar
MGM College of Engineering
Nanded, Maharashtra, India

Dear Tejas Dnyaneshwar Kotalwar,

Subject: Internship Offer – Adaptive Learning Research & Development

Congratulations! We are pleased to offer you a **research-based internship** at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

This internship presents a valuable opportunity to contribute to our ongoing research in *adaptive learning technologies* supporting K–12 and college-level education. You will collaborate remotely with AkharaPlus engineers and researchers, engaging in projects that directly impact how students learn through intelligent, personalized systems.

As an intern, you will:

- Work remotely on research and development tasks under the guidance of AkharaPlus staff engineers.
- Attend weekly status meetings, present your progress, and participate in technical discussions.
- Take initiative in solving problems, seeking feedback, and documenting your development.

After your internship, you are required to submit a comprehensive report summarizing:

- The work you performed,
- The new technical and professional skills you acquired, and
- Your reflections on transitioning from a university student to a professional software engineer.
- Upload your findings, including code, to a specified GitHub repository.

Your performance will be evaluated using a rubric developed by AkharaPlus, LLC. The final evaluation will be submitted to the **Dean of MGM College of Engineering** for academic purposes.

Please note that this is a **no-stipend internship**. Its value lies in the immersive, real-world

experience you will gain in a research-driven software engineering environment.

We thank you for choosing **AkharaPlus, LLC.** for your internship. We look forward to working with you and contributing to your professional advancement.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

+1 252 320 7431 info@AksharaPlus.org http://www.aksharaplus.org
AksharaPlus, LLC.8 The Green STE B, Dover, DE 19901 USA

July 2, 2025

Ganesh Kondamwar
MGM College of Engineering
Nanded, Maharashtra
India

Dear Ganesh Kondamwar,

Congratulations on your completion of a research-based internship, titled **Auto-Question Generator for Dynamic Programming**, at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

During this internship, you contributed to our ongoing initiatives in adaptive learning technologies and educational software development. The internship was structured to provide hands-on research and development experience aligned with real-world challenges in supporting K–12 and college-level learning through intelligent systems.

You fulfilled all responsibilities, which included:

1. Working remotely under the guidance of AksharaPlus engineering staff,
2. Attending and contributing to weekly status meetings,
3. Presenting progress and receiving feedback,
4. Preparing a final report documenting technical work, acquired skills, and professional development.

Specific details about your project include the following:

Project Title: Auto-Question Generator for Dynamic Programming

Project Goals: To develop an intelligent system that automates the generation and answering multi-level questions related to the dynamic programming topic. We aim to enhance algorithm understanding, support practice-based learning, and reduce manual effort in preparing educational content.

Project Deliverables: A modular Python application that automatically generates and answers questions on dynamic programming-based topics such as Knapsack, LCS, and MCM; an integrated multi-level question support including conceptual (Level 1), input/output-based (Level 2), and code-based (Level 3) formats;

and a dispatcher logic to intelligently route questions to topic-specific answer engines for accurate and context-aware responses.

Skills Developed: Application of dynamic programming to real-world automation use cases (Knapsack, LCS, MCM); expertise in developing Python applications using clean code architecture; gained experience in automated question generation and answer logic design; improved skills in problem decomposition, abstraction, and code reuse; and how to map algorithm theory to practical software implementation.

We commend your commitment, collaboration, and initiative throughout the internship. AkshaPlus, LLC will serve as a professional reference, should you choose as you explore your career opportunities.

We wish you continued success in academic and professional pursuits and thank you for your valuable contributions to AksharaPlus, LLC.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

+1 252 320 7431 info@AksharaPlus.org http://www.aksharaplus.org
AksharaPlus, LLC.8 The Green STE B, Dover, DE 19901 USA

July 2, 2025

Soham Kashettiwar
MGM College of Engineering
Nanded, Maharashtra
India

Dear Soham Kashettiwar,

Congratulations on your completion of a research-based internship, titled **Auto-Question Generator for Dynamic Programming**, at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

During this internship, you contributed to our ongoing initiatives in adaptive learning technologies and educational software development. The internship was structured to provide hands-on research and development experience aligned with real-world challenges in supporting K–12 and college-level learning through intelligent systems.

You fulfilled all responsibilities, which included:

1. Working remotely under the guidance of AksharaPlus engineering staff,
2. Attending and contributing to weekly status meetings,
3. Presenting progress and receiving feedback,
4. Preparing a final report documenting technical work, acquired skills, and professional development.

Specific details about your project include the following:

Project Title: Auto-Question Generator for Dynamic Programming

Project Goals: To develop an intelligent system that automates the generation and answering multi-level questions related to the dynamic programming topic. We aim to enhance algorithm understanding, support practice-based learning, and reduce manual effort in preparing educational content.

Project Deliverables: A modular Python application that automatically generates and answers questions on dynamic programming-based topics such as Knapsack, LCS, and MCM; an integrated multi-level question support including conceptual (Level 1), input/output-based (Level 2), and code-based (Level 3) formats;

and a dispatcher logic to intelligently route questions to topic-specific answer engines for accurate and context-aware responses.

Skills Developed: Application of dynamic programming to real-world automation use cases (Knapsack, LCS, MCM); expertise in developing Python applications using clean code architecture; gained experience in automated question generation and answer logic design; improved skills in problem decomposition, abstraction, and code reuse; and how to map algorithm theory to practical software implementation.

We commend your commitment, collaboration, and initiative throughout the internship. AkshaPlus, LLC will serve as a professional reference, should you choose as you explore your career opportunities.

We wish you continued success in academic and professional pursuits and thank you for your valuable contributions to AksharaPlus, LLC.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

AksharaPlus

+1 252 320 7431 info@AksharaPlus.org http://www.aksharaplus.org

AksharaPlus, LLC.8 The Green STE B, Dover, DE 19901 USA

July 2, 2025

Tejas Kotalwar
MGM College of Engineering
Nanded, Maharashtra
India

Dear Tejas Kotalwar,

Congratulations on your completion of a research-based internship, titled **Auto-Question Generator for Dynamic Programming**, at **AkharaPlus, LLC.**, beginning **February 2025** and concluding **June 2025**.

During this internship, you contributed to our ongoing initiatives in adaptive learning technologies and educational software development. The internship was structured to provide hands-on research and development experience aligned with real-world challenges in supporting K–12 and college-level learning through intelligent systems.

You fulfilled all responsibilities, which included:

1. Working remotely under the guidance of AksharaPlus engineering staff,
2. Attending and contributing to weekly status meetings,
3. Presenting progress and receiving feedback,
4. Preparing a final report documenting technical work, acquired skills, and professional development.

Specific details about your project include the following:

Project Title: Auto-Question Generator for Dynamic Programming

Project Goals: To develop an intelligent system that automates the generation and answering multi-level questions related to the dynamic programming topic. We aim to enhance algorithm understanding, support practice-based learning, and reduce manual effort in preparing educational content.

Project Deliverables: A modular Python application that automatically generates and answers questions on dynamic programming-based topics such as Knapsack, LCS, and MCM; an integrated multi-level question support including conceptual (Level 1), input/output-based (Level 2), and code-based (Level 3) formats;

and a dispatcher logic to intelligently route questions to topic-specific answer engines for accurate and context-aware responses.

Skills Developed: Application of dynamic programming to real-world automation use cases (Knapsack, LCS, MCM); expertise in developing Python applications using clean code architecture; gained experience in automated question generation and answer logic design; improved skills in problem decomposition, abstraction, and code reuse; and how to map algorithm theory to practical software implementation.

We commend your commitment, collaboration, and initiative throughout the internship. AkshaPlus, LLC will serve as a professional reference, should you choose as you explore your career opportunities.

We wish you continued success in academic and professional pursuits and thank you for your valuable contributions to AksharaPlus, LLC.

Sincerely,

[illegible]

Dr. Dhana L. Rao
President and CEO

ACKNOWLEDGEMENT

With deep reverence, we express our sincere gratitude to our project guide **Dr. Bhagyashri S. Kapre** for her invaluable guidance, support, and constant encouragement throughout the course of this project. It has been an enriching experience working under her mentorship, and we truly appreciate her insightful suggestions and thoughtful discussions that greatly contributed to the success of our work.

We would also like to extend our heartfelt thanks to **Dr. A. M. Rajurkar**, Head of the Department, Computer Science & Engineering, **MGM's College of Engineering, Nanded**, for her continued support and for providing us with the necessary facilities and a motivating environment.

Our sincere thanks also go to **Dr. G. S. Lathkar**, Director, **MGM's College of Engineering, Nanded**, for her kind help, encouragement, and for facilitating the smooth progress of our project.

With Deep Reverence

Ganesh Kondamwar

Soham Kashettiwar

Tejas Kotalwar

[B.Tech CSE-B]

ABSTRACT

This project presents an Automatic Question and Answer Generation System aimed at simplifying the creation of assessment material from structured theory content. Designed primarily for algorithmic and conceptual topics in computer science, the system allows users to generate various types of questions—Multiple Choice Questions (MCQ), True/False, and Descriptive—at three predefined difficulty levels. The goal is to assist educators, learners, and content creators in rapidly generating evaluation content without requiring deep manual effort.

The architecture is modular and scalable, featuring key components such as a dispatcher to control the flow of logic, conceptual engines to process theoretical inputs, and algorithm-specific engines that cater to topics like Fibonacci, Knapsack, and others. A dedicated answer dispatcher module generates accurate, human-like explanations to accompany each question, ensuring that the outputs are not only structurally correct but also pedagogically sound. Input content is categorized and organized in a simple folder structure, and the system operates using deterministic templates rather than relying on external NLP libraries, making it lightweight and easy to extend.

Outputs are generated in a structured format and can be saved for integration into quiz platforms or printed tests. The system's layered design allows future integration with front-end interfaces, voice-based interaction, or adaptive learning tools. Overall, this project showcases how a well-structured logic-based approach can be effectively used to automate academic content creation and lays the foundation for more intelligent, AI-enhanced educational systems in the future.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In recent years, the field of education has seen a significant shift towards digital learning platforms, especially after the rise of remote learning and online assessments. With the increasing number of students and the need for continuous evaluation, educators often find themselves burdened with the repetitive task of preparing question banks, quizzes, and assessments. Manual generation of questions is not only time-consuming but also prone to inconsistencies in difficulty level, coverage of concepts, and bias.

1.1 Background

The need for intelligent educational tools that can assist in automating this process has become evident. Among these tools, automatic question generation (AQG) stands out as a significant area of interest in educational technology. It refers to the process of generating meaningful, grammatically correct, and educationally relevant questions from a given body of text or conceptual content without human intervention.

Traditionally, question papers and quizzes are curated manually by subject matter experts based on textbooks and learning objectives. However, this method is not scalable, especially in systems where personalized learning paths and adaptive assessments are becoming the norm. The manual process also lacks efficiency when catering to learners at different levels (beginner, intermediate, advanced).

With the advancement in Natural Language Processing (NLP) and Artificial Intelligence (AI), it is now possible to design systems that can extract key concepts from theory or algorithms and automatically formulate questions in various formats like:

- Multiple Choice Questions (MCQs)
- Multiple Select Questions (MSQs)
- True/False Statements
- Theoretical Questions for explanation

Moreover, incorporating difficulty levels (Level 1 - Basic, Level 2 - Intermediate, Level 3 - Advanced) allows the system to address diverse learners' capabilities and educational goals. This kind of smart system is especially useful in competitive exam preparation, academic testing, and skill-based assessments in technical subjects like Dynamic Programming, Data Structures, and Algorithms.

This project**,** Auto Question Answer Generator, is built with the goal of bridging this gap by automating the creation of educational content tailored to specific levels, categories, and algorithms. By leveraging rule-based logic, NLP techniques, and structured templates, this system ensures that educators and learners are provided with consistent, categorized, and contextually relevant questions accompanied by their respective answers and theory explanations.

1.2 Motivation

The motivation behind developing the Auto Question Answer Generator arises from the growing demand for scalable and intelligent educational tools in modern learning environments. In academic institutions and online learning platforms, educators are under constant pressure to prepare high-quality learning assessments, practice tests, quizzes, and exams that align with curriculum objectives and cater to students of varying capabilities. However, manually creating such question banks is a labor-intensive process that often leads to inconsistencies in difficulty levels, repetition of concepts, limited coverage of the syllabus, and significant time constraints for educators.

Traditional teaching and assessment methods heavily rely on the individual educator's effort to create content for each subject, which can lead to inconsistencies and limit scalability. The proposed system overcomes this dependency by automating question generation and providing standardized, conceptually accurate, and difficulty-graded assessment materials, ensuring consistency and reducing the workload on educators.

In parallel, students preparing for competitive examinations such as GATE, JEE, UPSC, and various university-level assessments frequently require a broad range of practice questions that are systematically categorized by complexity and topic. Existing resources often fall short in this regard, being either too generic or insufficiently tailored to specific learning goals. Moreover, they typically lack the ability to dynamically adapt to a student's level of understanding or offer immediate conceptual reinforcement. This project is driven by the belief that automated and intelligent systems can significantly ease the manual burden on educators while simultaneously providing high-quality, adaptive learning materials to students. One of the key motivational aspects is the potential for personalized learning support, where students can access concept-based questions that gradually increase in complexity, thereby reinforcing understanding and improving knowledge retention. Additionally, educators can benefit from such a system by efficiently generating assessment content directly from lecture notes or theoretical inputs, allowing them to devote more time to teaching and mentoring.

The tool also holds promise for addressing educational disparities between urban and rural regions. In rural or under-resourced schools, where access to experienced educators is limited, an auto question generation system can deliver structured and quality practice material with minimal teacher intervention. Furthermore, in technically demanding subjects like Dynamic Programming, Greedy Algorithms, or Graph Theory, the ability to generate topic-specific questions — such as those related to Knapsack, Longest Common Subsequence (LCS), or the Fibonacci sequence — provides highly targeted and relevant practice opportunities for learners.

Another major motivation lies in promoting self-paced learning. By combining theoretical explanations with corresponding questions, the system allows learners to first build conceptual understanding and then apply it through practice, effectively replicating a tutor-guided learning experience. Ultimately, this project aims to empower both educators and learners through technology by streamlining the question-answer creation process in a structured, meaningful, and scalable manner. It aspires to contribute to innovation in the edtech space, showcasing how a blend of AI techniques and rule-based logic can complement traditional education systems and enhance the quality and accessibility of assessments.

1.3 Problem Statement

"Designing diverse and accurate assessment content, such as questions and theoretical explanations, is essential yet challenging for educators, especially in complex subjects like Data Structures and Algorithms".

Existing digital learning platforms largely rely on static, manually created question banks and lack the ability to dynamically generate multi-format questions based on conceptual inputs. Additionally, they often do not classify questions by difficulty level or provide supporting theoretical context, limiting their effectiveness for customized and adaptive learning.

The core problem is the absence of an intelligent, modular system capable of automatically generating varied question types—such as MCQs, True/False, and theory-based questions—directly from algorithmic or conceptual content. This system should classify questions by difficulty, maintain consistency and quality, and include relevant theory to support learner understanding, while also being extensible to cover additional domains.

1.4 Objectives

The primary objective of this project is to design and implement an intelligent, automated system capable of generating educational questions and corresponding answers from theoretical and algorithmic input. The motivation lies in supporting both educators and learners — by assisting educators in reducing the time and effort involved in designing assessments, and by helping students reinforce concepts through structured, adaptive practice questions across various complexity levels.

To begin with, the system is intended to fully automate the question generation process. Educators often spend considerable time preparing questions that match the syllabus, maintain conceptual integrity, and cover all difficulty levels. By enabling the system to accept raw topic inputs — such as “Knapsack Problem,” “Dynamic Programming,” or other algorithmic concepts — and generate structured, well-framed questions automatically, the project significantly streamlines this process. The goal is to reduce redundancy, save time, and allow instructors to focus more on teaching and mentoring.

The system is also designed to support the generation of multiple question types to suit different pedagogical needs and assessment patterns. This includes the creation of Multiple Choice Questions (MCQs), Multiple Select Questions (MSQs), True/False statements, and theory or concept-based descriptive prompts. Such diversity in format ensures that the tool is not limited to one type of examination but can be flexibly used for class tests, competitive exam practice, mock quizzes, or self-evaluation tools.

Another important objective is the classification of questions based on their difficulty. The system categorizes content into three levels: Level 1 (Basic), which involves fundamental definitions and simple, direct questions; Level 2 (Intermediate), which includes application-level questions that test understanding; and Level 3 (Advanced), which emphasizes logic development, problem-solving, and deeper comprehension. This stratification ensures that learners at different stages of understanding can benefit, whether they are just beginning or are preparing for advanced-level examinations.

Beyond question generation, the system is envisioned to serve as a self-learning tool by providing concise conceptual theory along with each set of questions. These explanations are particularly helpful for learners who may not have immediate access to instructors. Before attempting a question, a student can read through the associated theory to build a clear foundation. This pedagogical structure mimics the presence of a human tutor and bridges the gap between theory and practice.

In terms of design architecture, scalability and modularity are central to the project. The system is built using independent components such as dispatchers, question generators, and theory handlers. This modular design not only ensures clean code and manageable development but also allows the system to be easily extended in the future. New subjects, additional algorithms, or even multimedia question formats can be integrated without having to overhaul the entire architecture.

To ensure the accuracy and relevance of generated content, the system leverages rule-based techniques along with basic Natural Language Processing (NLP) methods. It uses templates, keyword identification, and grammar rules to form coherent questions that are grammatically correct and contextually meaningful. These techniques help in converting raw educational input into properly structured assessments that simulate real-world examination standards.

Lastly, one of the critical goals of the project is to develop a lightweight and accessible solution. In many rural and low-infrastructure regions, there is a lack of access to smart classrooms, high-speed internet, or advanced computing devices. This tool is deliberately designed to be usable on basic machines with minimal requirements, ensuring that it is inclusive and accessible to under-resourced educational environments. The interface is kept intuitive and minimal, allowing users from diverse backgrounds — including school teachers, college faculty, and independent learners — to easily interact with the system. In summary, this project aims to empower both teachers and learners through a scalable, intelligent question-generation system that is flexible in design, inclusive in use, and robust in its pedagogical value.

1.5 Scope and Relevance

The scope of this project extends well beyond routine academic coursework. It directly addresses critical challenges encountered in modern educational settings—particularly those related to the manual creation of question banks and quizzes, which often suffer from inconsistencies, limited scalability, and time inefficiencies. The system is highly relevant for a range of stakeholders. Computer Science educators, for instance, benefit from its ability to generate algorithm-based questions automatically, saving valuable time while maintaining quality. Self-learners and aspirants preparing for competitive exams gain access to personalized, difficulty-level-based assessments that align with their learning pace.

EdTech platforms can leverage the system to integrate dynamic and intelligent assessment features into their applications. Additionally, schools and colleges in rural or underserved regions—where experienced faculty may be scarce—can use this tool to offer high-quality learning resources. Designed with adaptability in mind, the system is not restricted to dynamic programming alone; it can be extended to cover subjects like Machine Learning, Operating Systems, and even non-technical domains through customized template libraries.

Moreover, this system encourages student-centered learning models by enabling practice that is both automated and conceptually meaningful. It allows students to identify their own weak areas by engaging with diverse question formats—such as MCQs, descriptive answers, code-level implementations, and theoretical concepts—thus promoting a holistic approach to mastering technical topics. From a curriculum design standpoint, the system can also assist instructors in creating assessments mapped to learning outcomes and in generating formative quizzes with minimal effort.

From a technological standpoint, the project demonstrates how low-resource, lightweight Python-based systems can still achieve scalable and intelligent automation. Its modular design and use of JSON templates, combined with algorithmic computation and explanation layering, make it highly portable and easy to integrate with larger learning management systems (LMS) or e-learning portals.

Importantly, the tool contributes to bridging the digital divide. In regions with limited internet connectivity or access to quality instructors, such systems can democratize education by bringing structured, interactive, and personalized learning content to students. With future enhancements like multilingual support, offline GUIs, and mobile integration, this project holds potential for broader societal impact.

Thus, the relevance of the system spans across academia, industry, educational policy, and social equity—making it not just a technical solution, but a meaningful step toward scalable, inclusive, and outcome-driven education.

Chapter 2

LITERATURE REVIEW

Automatic Question Generation (AQG) has emerged as a significant area of research within artificial intelligence and educational technology. Numerous systems have been developed to automate the process of generating questions from textual or structured input. Broadly, these systems can be classified into three categories, one of which includes rule-based systems.

2.1 Review on Auto Question Generation

Automatic Question Generation (AQG) is a multidisciplinary research area that intersects the fields of Natural Language Processing (NLP), Educational Technology, and Artificial Intelligence (AI). The primary objective of AQG systems is to automate the creation of meaningful, pedagogically sound, and syntactically correct questions from a given input be it text, data, or structured knowledge. Over the years, researchers have explored various methodologies, ranging from simple rule-based and template-driven approaches to sophisticated deep learning models capable of generating human-like questions. These advancements have been fueled by the increasing availability of annotated datasets, growing computational resources, and the emergence of powerful neural architectures. AQG systems are now being used in intelligent tutoring systems, automated assessments, e-learning platforms, and even in large-scale testing environments. The subsections that follow explore the evolution of these systems, starting with modern machine learning and neural network-based techniques.

2.1.1 Machine Learning and Neural Approaches – Methodology and Process

Machine learning-based question generation systems primarily use a Sequence-to-Sequence (Seq2Seq) architecture, a framework in which an input sequence—usually a sentence or paragraph containing an answer—is passed through an encoder that understands its structure and meaning. A decoder then generates a corresponding question word-by-word. This structure became more powerful with the advent of Transformer models, which introduced self-attention mechanisms that allow the model to weigh different parts of the input text simultaneously. State-of-the-art transformer-based models include BERT (Bidirectional Encoder Representations from Transformers), GPT-2/3 (Generative Pretrained Transformers), T5 (Text-to-Text Transfer Transformer), and BART (Bidirectional and Auto-Regressive Transformers). These models are first pretrained on large corpora like Common Crawl or Wikipedia and then fine-tuned on AQG-specific datasets such as SQuAD, NewsQA, and MS MARCO (Du & Cardie, 2018; Lewis et al., 2019; Raffel et al., 2020)[4].

A key enhancement is Answer-Aware Question Generation (AAQG), which explicitly includes the target answer during the input stage. For instance, the T5 model accepts formatted input like: “generate question: [context] answer: [answer]”. This helps the decoder frame questions that focus specifically on the given answer, improving both relevance and precision (Raffel et al., 2020). Unlike traditional models that might generate vague or off-topic questions, this approach centers the generation around a known answer span. Training these models involves minimizing the cross-entropy loss, a standard loss function in language generation tasks that measures the difference between the generated question and the true question. Over time, the model learns to generalize and produce fluent, accurate questions from various types of inputs. The use of attention layers and contextual embeddings—where words are interpreted based on their surrounding words—enables the model to better understand complex sentence structures and semantic nuances (Lewis et al., 2019)[4]. In practice, transformer-based AQG systems are integrated into adaptive learning platforms such as Gradescope, Turnitin, and Knewton Alta, where they generate multiple-choice and short-answer questions in real time from digital course content. These systems can adjust question complexity based on a student’s performance level, supporting personalized learning (Bulathwela et al., 2023; Zhang, 2023; Amyeen, 2023)[7].

The advantages of neural approaches include their ability to generate natural, diverse, and grammatically sound questions, as well as their adaptability across subject domains and languages. They can also learn from massive datasets, improving accuracy and fluency over time. However, their limitations include high computational costs, requiring GPUs or TPUs for training and inference. Furthermore, they are less interpretable, meaning their internal decision-making is not transparent. Without proper training data or constraints, they may produce factually incorrect or irrelevant questions.

2.1.2 NLP and Linguistic-Based Systems – Methodology and Process

Natural Language Processing (NLP)-based question generation systems use linguistic analysis to understand and manipulate the grammatical structure of input text. The process begins with linguistic preprocessing, which involves tasks like Part-of-Speech (POS) tagging—labeling words with grammatical roles (e.g., noun, verb, adjective); Named Entity Recognition (NER)—detecting proper nouns such as people, locations, or dates; and Dependency Parsing—identifying syntactic relationships between words. These linguistic tools extract the grammatical structure of sentences and identify critical components like subjects, verbs, and objects, which are essential for transforming declarative sentences into questions (Heilman & Smith, 2010; Tami et al., 2024)[10].

After linguistic features are extracted, the system applies rule-based syntactic transformation, which involves rewriting declarative statements into interrogative questions using grammatical rules. For example, the sentence "Marie Curie discovered radium" is parsed to identify "Marie Curie" as the subject and "radium" as the object. The system then uses a syntactic rule to generate "What did Marie Curie discover?" This method, formalized in pipelines like that of Heilman & Smith (2010), ensures that the questions are grammatically correct and structurally coherent[10].

Another important step is Answer Span Detection and WH-Mapping, where the system identifies which part of the input sentence is to be treated as the answer. Based on the entity type—e.g., person, location, date—the system chooses an appropriate WH-word such as "Who," "Where," or "When." For example, if the detected answer is a time or date, "When" is used. This process ensures that the generated question aligns both grammatically and semantically with the original sentence (Chali & Hasan, 2015). These linguistic systems are commonly integrated into real-world applications like Quillionz, which automatically suggests questions from a paragraph. Major content publishers such as Pearson and McGraw-Hill incorporate these systems into their platforms to auto-generate comprehension questions from textbooks (Mitkov & Ha, 2003)[13].

The advantages of linguistic-based systems include their transparency, as the transformation rules are human-defined and understandable. They also require low computational resources, making them suitable for environments without advanced hardware. However, their limitations include inflexibility—they do not perform well on ungrammatical or complex sentences—and they often produce repetitive or overly rigid questions. Since the system's coverage is limited to the grammar rules it knows, it may miss meaningful or abstract question opportunities.

2.1.3 Rule-Based Systems – Methodology and Process

Rule-based AQG systems function on a foundation of template-based transformation, where input sentences are matched against predefined patterns and converted into questions using rigid templates. These templates may follow fixed formats such as

“What is {X}?”, “Who invented {Y}?”, or “When did {event} occur?” For example, if the input sentence is “The Pacific Ocean is the largest ocean,” a rule might transform this into “What is the largest ocean?” Such patterns are easy to program and can be reliably applied across structured text (Mitkov & Ha, 2003)[13].

This methodology depends heavily on pattern matching and lexical rules, which use keyword spotting and syntactic patterns to identify when a rule should be triggered. For example, in the sentence “Water boils at 100 degrees Celsius,” the pattern of scientific fact involving a numeric value activates a rule to generate “At what temperature does water boil?” These systems rely on POS tagging and surface-level structure rather than deep semantic understanding (Gates, 2008)[14].

Rule-based AQG tools have been widely implemented in educational authoring platforms like Hot Potatoes, AutoQG, and iSpring QuizMaker. These tools allow educators to input structured content and automatically generate questions in multiple-choice or fill-in-the-blank format. They are particularly effective in language learning, grammar training, and e-learning modules that prioritize format and consistency (Varga & Bánhegyi, 2011)[15].

The advantages of rule-based systems include complete control, predictability, and ease of customization. They require minimal computational power and are easy to implement, even in offline or low-resource settings. However, their limitations are severe: they lack semantic depth, cannot handle complex or creative sentence structures, and produce questions that often feel robotic or repetitive. They are also non-adaptive, meaning they cannot improve over time or handle unseen sentence types unless new rules are explicitly added (May et al., 2025)[16].

2.2 Gaps in Current Systems

Despite significant advancements in Automatic Question Generation (AQG) technologies, several key limitations continue to hinder their effectiveness. One major issue lies in the shallow understanding of content exhibited by many existing systems. These tools often rely on surface-level features such as keywords or syntactic patterns to formulate questions, rather than engaging in true conceptual comprehension. As a result, the generated questions tend to be repetitive, overly focused on fact-based recall, and misaligned with deeper educational objectives or learning outcomes. This lack of semantic depth limits their usefulness in fostering critical thinking or higher-order cognitive skills in learners [3], [11].

2.2.1 Shallow Understanding of Content

One of the primary limitations of existing Automatic Question Generation (AQG) systems is their shallow understanding of content. Most systems rely heavily on keyword extraction or surface-level sentence structures to formulate questions, rather than engaging with the underlying concepts or meaning. This superficial approach often results in questions that are repetitive or redundant, offering little variation or depth. Moreover, the focus tends to skew toward fact-based recall, with limited capacity to assess higher-order thinking or application-based understanding. Consequently, the generated questions often lack alignment with broader educational goals, making them less effective in promoting deep learning or conceptual clarity among students [3], [11].

2.2.2 Limited Algorithmic Domain Support

Another significant and often overlooked limitation in the current landscape of Automatic Question Generation (AQG) systems is their lack of support for domain-specific, algorithm-intensive subjects, particularly in areas such as data structures, algorithms, and theoretical computer science. While many state-of-the-art AQG solutions have been optimized for natural language texts—especially for generating questions based on reading comprehension or factual paragraphs—they are not well-equipped to handle the specialized requirements of computational fields. This includes a profound inability to parse, interpret, or evaluate logical structures like recursion, iteration, mathematical formulations, and data-driven problem-solving techniques that are fundamental to algorithm design. Most existing models are trained on general-purpose datasets such as SQuAD, NewsQA, and MARCO, which are inherently non-technical and lack examples from programming, computational logic, or structured pseudocode. Consequently, these systems tend to produce vague or contextually irrelevant questions when applied to topics like dynamic programming, binary trees, hash maps, or graph algorithms. For instance, they may fail to capture nuances such as overlapping subproblems in dynamic programming, the greedy-choice property, or the significance of memoization—concepts that are essential for deep learning and mastery in the field of computer science.

Furthermore, general NLP-based AQG tools often rely on shallow text patterns or lexical cues, which are insufficient when attempting to generate questions that involve mathematical derivation, table-based dynamic programming states, or combinatorial logic. The lack of support for code generation, complexity analysis, and algorithm-specific theory makes these systems unsuitable for computer science education, where precision, accuracy, and step-by-step logic are critical.

This notable shortfall serves as a core motivation for the proposed Auto Question Generator system, which aims to bridge the gap by integrating algorithm-aware modules and dedicated conceptual engines. These modules are capable of handling specific algorithmic categories such as the 0/1 Knapsack problem, Longest Common Subsequence (LCS), and recursive Fibonacci computations. By embedding computational logic directly into the answer generation process, the system not only computes accurate results but also provides detailed step-by-step explanations, Python code, and traceable outputs.

2.2.3 One-Dimensional Question Formats

A further limitation observed in many existing AQG tools is their over-reliance on a single type of question format—primarily multiple-choice questions (MCQs). While MCQs are useful for quick assessments, they often fall short in evaluating a learner's depth of understanding or critical thinking abilities. Comprehensive educational assessment requires a more diverse range of formats, including True/False questions for quick concept validation, descriptive or open-ended questions to foster analytical thinking, and theoretical context to enhance understanding and learning continuity [8].

The lack of support for such varied question types restricts the pedagogical effectiveness of these tools. In contrast, the system proposed in this project addresses this limitation by enabling multi-format question generation, offering educators and learners a more flexible and enriched assessment environment.

Chapter 3

AUTO QUESTION GENERATION SYSTEM

The Auto Question-Answer Generator for Dynamic Programming is a structured educational platform that produces a wide variety of algorithm-related questions and their computed answers. It automates the content generation process using reusable templates, dynamic placeholder values, and computational engines that simulate real algorithmic behavior. The purpose of this system is to save time for educators, enhance personalized learning, and ensure topic-wise progression for students preparing for computer science courses or coding interviews.

The system classifies questions by: The system classifies each question based on its type, level of difficulty, and algorithmic topic. The supported types include Multiple Choice Questions, Multiple Select Questions, True or False, and Theory-based questions. Each question is also categorized by difficulty level: Level 1 represents basic concepts, Level 2 introduces intermediate challenges, and Level 3 consists of advanced implementation problems. The topics currently covered are Fibonacci Sequence, Longest Common Subsequence, 0/1 Knapsack, and Memoization. This classification allows the system to deliver customized educational content based on user preferences. Dynamic Programming is a foundational technique in computer science used to solve complex problems by breaking them into smaller overlapping subproblems. Instead of solving the same subproblems repeatedly, intermediate results are stored and reused, either through memoization (top-down approach) or tabulation (bottom-up approach). This drastically improves time complexity, making otherwise exponential problems solvable in polynomial time. The Auto Question-Answer Generator includes four major subtopics under Dynamic Programming, each with dedicated templates categorized by difficulty level and question type.

A. Fibonacci Sequence

The Fibonacci sequence is the classic introductory topic in Dynamic Programming. It involves computing a series where each term is the sum of the previous two terms. The naive recursive approach is highly inefficient and leads to redundant calculations, which makes it ideal for introducing memoization.

Level 1 Template Example (Conceptual)

"What is the benefit of using memoization with the Fibonacci function?"

Level 2 Template Example (Application)

"Solve the problem of counting ways to climb {{n}} stairs, where you can take 1 or 2 steps at a time, using memoization."

Level 3 Template Example (Quantitative)

"Using memoization, calculate the number of unique binary search trees that can be constructed using {{n}} nodes."

These questions help students move from basic understanding to application, and finally to advanced topics like tree generation and combinatorics using recursive memoization.

B. Longest Common Subsequence

The Longest Common Subsequence (LCS) problem is a standard Dynamic Programming string problem where the goal is to find the longest sequence that appears in the same order in two given strings. It is solved using a two-dimensional table to store solutions to subproblems based on character matching.

Level 1 Template Example (Conceptual)

"What is a Longest Common Subsequence and how is it different from substring?"

This question introduces learners to the fundamental concept of LCS. It helps differentiate between subsequences and substrings, an essential distinction often misunderstood. While a substring requires characters to be contiguous, a subsequence only requires order preservation. This template ensures that students first grasp the theory before solving problems, reinforcing foundational knowledge required in both exams and coding interviews.

Level 2 Template Example (Application)

"Write a function that computes the length of the Longest Common Subsequence between '{{STR1}}' and '{{STR2}}'."

At this level, learners are encouraged to apply the concept in a real coding context. The use of placeholders like {{STR1}} and {{STR2}} allows the system to generate a wide variety of problem instances. This template is typically backed by a dynamic programming-based engine that outputs both the correct LCS length and a trace of the 2D DP table. It tests not only theoretical knowledge but also the learner's ability to implement a working solution. This step bridges the gap between textbook learning and real-world algorithm design.

Level 3 Template Example (Algorithmic)

"Write a function to return all possible Longest Common Subsequence sequences for '{{STR1}}' and '{{STR2}}'."

This question represents the highest cognitive level in Bloom's taxonomy—synthesis and problem-solving. Unlike the Level 2 template which requires only the length, this version demands generating all valid subsequences that qualify as longest. It tests deep understanding of backtracking and recursion on top of DP logic. This template is ideal for advanced learners preparing for competitive programming or algorithm-heavy interviews.

C. 0/1 Knapsack Problem

The 0/1 Knapsack problem teaches optimization using Dynamic Programming. Given a set of items with specific weights and values, and a maximum capacity, the goal is to find the highest total value of items that can fit in the knapsack without exceeding the capacity. The problem introduces a core concept of state transition and table building.

Level 1 Template Example (Conceptual)

"Why can't a greedy strategy be used for the 0/1 Knapsack problem?"

This question introduces the learner to one of the most critical insights in algorithm design: the distinction between problems solvable via greedy methods and those requiring dynamic programming. While fractional knapsack can be solved using a greedy approach (by selecting items based on the value-to-weight ratio), the 0/1 variant cannot, due to the indivisibility of items and the global optimality constraints. This template helps learners understand why dynamic programming is essential and lays the conceptual foundation for the decisions made in subsequent algorithmic implementations.

Level 2 Template Example (Application)

"Using memoization, solve the 0/1 Knapsack problem with weights={{weights}}, values={{values}}, and max capacity={{capacity}}."

This template escalates the challenge by requiring learners to apply memoization—an optimization technique for recursion—within the 0/1 Knapsack problem. Here, the placeholders {{weights}}, {{values}}, and {{capacity}} are dynamically filled during generation, allowing the system to produce hundreds of unique, personalized problem sets. This level emphasizes recursive problem-solving with subproblem caching, providing the learner with an efficient and structured problem-solving framework that balances conceptual understanding with coding practice.

Level 3 Template Example (Quantitative)

"Implement a hybrid top-down memoization and bottom-up dynamic programming solution for the Knapsack problem with inputs weights={{weights}}, values={{values}}, and capacity={{capacity}}."

This advanced template represents the synthesis of multiple dynamic programming strategies. Learners are challenged to combine the flexibility of top-down memoization with the efficiency of bottom-up tabulation. This hybrid approach demonstrates deeper comprehension of both strategies and the ability to optimize based on context, input size, or space constraints. This template is especially relevant for learners targeting competitive programming, system design interviews, or research roles, where nuanced algorithmic optimization is often required. The templates in this section not only test theoretical knowledge but also train learners in space and time optimization strategies by applying both memoization and tabulation.

Together, these templates enable the Auto QA Generator system to span a full learning arc—from understanding algorithmic limitations to implementing optimized solutions. The templated structure, combined with placeholder substitution, ensures both reusability and diversity in generated questions. Furthermore, it reflects best pedagogical practices by aligning assessments with Bloom's Taxonomy: knowledge (Level 1), application (Level 2), and synthesis (Level 3). This progression makes the system not only a tool for assessment but a platform for developing computational thinking and algorithmic expertise.

D. Memoization

Memoization is not just a technique but a broader conceptual tool used throughout recursive problems. It refers to storing previously computed values to avoid duplicate work in recursion-heavy problems like Fibonacci, climbing stairs, Knapsack, or LCS. This section of the system emphasizes both understanding and correct implementation of memoization in various contexts.

Level 1 Template Example (Conceptual)

"How does memoization avoid repeated calculations in recursion?"

Level 2 Template Example (Theory + Application)

"Compare memoization and tabulation in dynamic programming."

Level 3 Template Example (Advanced Application)

“Design a custom cache eviction mechanism for long-running memoized programs.”

Templates in this section cover a range of use cases such as combinatorics, search problems, grid traversal, and caching strategies, enabling learners to explore the full scope of how memoization can improve performance in real-world scenarios.

3.1 System Architecture Overview

The design follows a layered modular approach, with well-defined responsibilities for each component, allowing easy extension and modification.

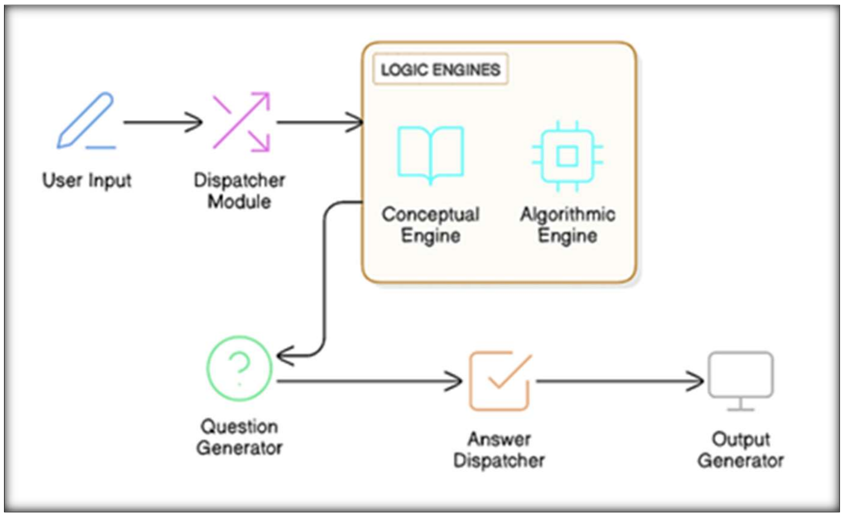


Fig 3.1 : Architecture Diagram

The core architecture of the Auto Question-Answer Generator is composed of multiple coordinated modules, each serving a distinct purpose in the question generation and answering process. At the center of the system is the main.py file, which functions as the central controller. It acts as the entry point for execution, handling all user inputs and initiating the appropriate processing workflow. Once the input is received, the request is passed to the dispatcher module, which plays a crucial role in mapping the user's selections—such as topic, question type, and difficulty level—to the correct backend processing engine.

After the dispatcher routes the request, it reaches one of the specialized components known as engines. These are of three types. The algorithm engines are responsible for dynamically computing answers to algorithmic problems using actual logic and code. In contrast, the conceptual engines provide static, pre-defined explanations for theory-based questions, helping the user understand definitions and principles. For factual recall questions such as multiple choice or true or false, the system accesses the objective folder, which contains a curated collection of ready-made questions and answers organized by topic and level.

Once an answer is generated, either through computation or lookup, the answer dispatcher module comes into action. This component enriches the raw answer by appending explanations, output traces, and code snippets wherever necessary, thereby transforming a basic answer into a complete and educational response. Together, these components form a robust pipeline capable of generating, computing, explaining, and delivering high-quality educational content tailored to the user's request.

Folder/File	**Purpose**
data/	Stores JSON templates with placeholders {}
answer_engine/	Contains files for real algorithmic computation
conceptual_answer_engine/	Contains static conceptual explanations by topic
objective/	Holds MCQ/TF question banks in dictionary format
answer_dispatcher/	Adds explanations, code snippets, and details to answers
info/	Contains basic conceptual theory for reference
main.py	Entry point – connects everything, handles user input
outputs/	Saves generated questions/answers (future expansion)

Table 3.1: Structure – Purpose Table

This table summarizes the core structure of the project. The data/ folder holds JSON templates with placeholders for generating questions. answer_engine/ performs real algorithmic computations, while conceptual_answer_engine/ provides static theory answers. objective/ contains MCQ and True/False question banks. The answer_dispatcher/ enhances responses with explanations and code. info/ offers basic conceptual theory. main.py acts as the system's entry point, and outputs/ is reserved for storing generated questions for future use.

3.2 System Flow (Detailed Execution Flow)

The Auto Question-Answer Generator for Dynamic Programming follows a well-defined execution flow once a user initiates an interaction with the system. The process begins with the user selecting specific parameters for the question generation. These include the topic (such as 0/1 Knapsack, Fibonacci Sequence, or Longest Common Subsequence), the category of question (conceptual, algorithmic, multiple choice, or true/false), the difficulty level (Level 1 for basic, Level 2 for intermediate, and Level 3 for advanced), and the number of questions they want to generate. This input provides the foundation for the system to deliver tailored educational content.

Once the user input is received, the control is passed to the central program file named main.py. This file acts as the entry point of the system and performs several essential tasks. First, it connects to the data/ folder, which contains pre-defined question templates stored in JSON format. These templates include placeholders such as {weights}, {values}, {capacity}, {STR1}, and {STR2}, which are dynamically filled based on the selected topic and difficulty level. After filling these placeholders with appropriate values, main.py determines the suitable processing path based on the category selected by the user.

Depending on the type of question requested, the system routes the processed template to the appropriate engine. For objective-type questions like multiple choice and true or false, the template is passed to the objective/ folder, which contains ready-made dictionaries categorized by topic and level. If the question is conceptual in nature—such as asking for a definition or theoretical explanation—it is forwarded to the conceptual_answer_engine/. On the other hand, if the question requires algorithmic computation, such as finding the longest common subsequence or the optimal value in a knapsack, it is directed to the answer_engine/ for real-time processing.

The dispatcher module plays a crucial role in routing the request correctly. It receives the input data including topic, category, and difficulty level, and sends the request to the appropriate component. Requests that require calculation are directed to the answer_engine/, conceptual explanations go to the conceptual_answer_engine/, and factual recall questions like multiple choice or true/false are handled by the objective/ folder.

When the request reaches the algorithmic computation engine, specific files related to the topic are invoked—such as knapsack_engine.py, lcs_engine.py, or fibonacci_engine.py. These files contain the core dynamic programming logic and perform real-time computations. The computation may involve building dynamic programming tables, executing recursive or iterative functions, and generating input/output traces. Once the logic is executed, the engine returns not only the final answer but also the intermediate computational steps and the actual code used for solving the problem, especially for advanced-level questions. For conceptual questions, the system refers to files in the conceptual_answer_engine/ folder. These Python files return pre-written theory-based explanations related to the chosen topic. For example, selecting a Level 1 conceptual question on Fibonacci would lead to an answer from fibonacci_conceptual_answers.py, which provides a clear and concise explanation of the topic.

If the request is for a multiple choice or true/false question, the system retrieves a corresponding entry from the objective/ folder. The user is then presented with the question and prompted to submit an answer. The system evaluates the response for correctness. If the answer is correct, it displays a message confirming correctness. If incorrect, it shows an explanation and the correct answer to help the learner understand the concept better.

Once the raw answer is generated—whether through computation or retrieval—it is sent to the answer_dispatcher / module. This module is responsible for formatting the final response. It appends a human-readable explanation of the algorithm or theory, includes the output computation steps, and, for algorithm-based questions, provides the exact Python code that was used to derive the result. This ensures that the learner not only gets the correct answer but also understands how it was obtained.

Finally, the complete output is displayed to the user. This includes the generated question, the correct answer, a detailed explanation, and, when applicable, the underlying Python code. For objective questions, the output also includes correctness feedback to guide the learner. This comprehensive flow ensures that each user interaction results in a complete, educationally valuable experience.

The project's modular architecture is organized to ensure clarity and scalability. The data/ folder contains JSON templates with placeholders that are dynamically filled with real input data such as weights, strings, and capacities. The answer_engine/ handles the core algorithmic logic, performing real-time computations to generate accurate answers and examples. The conceptual_answer_engine/ provides pre-written theory-based answers, categorized by topic and difficulty level. Objective-type questions like MCQs and True/False are managed within the objective/ folder, which evaluates user responses and provides immediate correctness feedback. The info/ folder serves as a quick-access repository for basic conceptual theory to support user understanding. Finally, the answer_dispatcher/ enhances the raw output by adding detailed explanations, step-by-step reasoning, and code—especially for advanced-level questions—making the system both informative and educational.

3.3 Algorithm-Specific Engines: Logic and Answer Computation

The answer_engine/ folder forms the computational backbone of this system. Unlike MCQs and True/False, which are pre-written in dictionary format under objective/, the algorithm engines here dynamically compute both the answer and detailed explanation **by** executing the real algorithms**.**

3.3.1 Structure of Each Algorithm Engine (Python File)

Each algorithm-specific file (e.g., knapsack_engine.py, lcs_engine.py, fibonacci_engine.py) contains:

1. **Example Setups**: Some files initialize example inputs (e.g., arrays, strings).
2. **Algorithm Function(s)**: Contains the actual recursive, memoized, or dynamic programming algorithms.
3. **Answer Generator Function**: Calls the algorithm → gets output → prepares final result with steps and explanation**.**
4. **Explanation Generator (within Answer Dispatcher)**: Formats computation trace and includes code if required (Level 3)**.**

3.3.2 Computational Answers Generation technique

The answer generation process begins by creating example inputs within the respective file located in the answer_engine/ directory. Once inputs are set, the core algorithm is executed to compute the answer dynamically. Following computation, an explanation is generated to detail the logic, show intermediate steps, and present the final output. For Level 3 (advanced) questions, the corresponding program code is also attached, which is assembled with the help of the answer_dispatcher.py module. This structured approach ensures both correctness and educational value in the generated content.

The algorithm used for this module is the 0/1 Knapsack, implemented using a Bottom-Up Dynamic Programming approach. The system generates descriptive or algorithmic questions, requiring the learner to understand and solve the problem logically. Unlike objective-type formats, this type of question does not include multiple-choice options. Instead, it provides a complete, computed answer along with a detailed explanation of the steps involved in solving the problem, including the dynamic programming table and item selections.

Example Question (Level 2 or 3)-0/1 Knapsack:

```
Q: Calculate the maximum value with weights = [2, 3, 4], values = [3, 
4, 5], capacity = 5.
```

Computed Answer (Displayed to User):

```
Answer: 7
Explanation: Using DP Table →
[ [0,0,0,0,0,0],
  [0,0,3,3,3,3],
  [0,0,3,4,4,7],
  [0,0,3,4,5,7] ]
The best value is obtained by including weights 2 and 3 → Total Value
= 7
```

Actual Code Logic (Internal):

```
def knapsack(wt, val, W):
    n = len(wt)
    dp = [[0 for _ in range(W + 1)] for _ in range(n + 1)]
    for i in range(1, n + 1):
        for w in range(1, W + 1):
            if wt[i - 1] <= w:
                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], 
dp[i - 1][w])
            else:
                dp[i][w] = dp[i - 1][w]
    return dp[n][W], dp  # Returns both result and DP Table for 
explanation
```

The algorithm used in this module is Longest Common Subsequence (LCS), implemented through a 2D Dynamic Programming (DP) table. It generates descriptive or algorithmic questions that require learners to trace the DP matrix and determine the LCS length. This question type does not provide multiple-choice options; instead, it delivers a complete computational answer along with a step-by-step explanation of how the LCS is derived, helping students understand the logic and structure of the DP solution.

Example Question (Level 2 or 3)-Longest Common Subsequence:

```
Q: What is the length of the LCS between "abcde" and "ace"?
```

Computed Answer (Displayed to User):

```
Answer: 3
Explanation: Using DP Table → The common subsequence is "ace", 
appearing in order. Final DP Matrix computed → Length = 3.
```

Actual Code Logic (Internal):

```
def lcs_length(X, Y):
    m, n = len(X), len(Y)
    dp = [[0 for _ in range(n+1)] for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if X[i] == Y[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1])
    return dp[m][n], dp  # Returns both result and DP Table for
explanation
```

The algorithm used for this module is the Fibonacci sequence, implemented using either Recursive with Memoization or Bottom-Up Dynamic Programming techniques. It generates descriptive questions that focus on computing the nth Fibonacci number. This type of question does not include any options; instead, it provides a direct computed answer along with the corresponding sequence and explanation, helping learners understand both the logic and the efficiency of optimized recursive solutions.

Example Question (Level 2 or 3) – Fibonacci Sequence:

```
Q: What is F(6) in the Fibonacci sequence?
```

Computed Answer (Displayed to User):

```
Answer: 13
Explanation: Fibonacci Sequence → [0, 1, 1, 2, 3, 5, 8, 13]
```

Actual Code Logic (Internal):

```
def fibonacci(n, memo={}):
    if n in memo: return memo[n]
    if n <= 1: return n
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]
```

3.3.3 How Explanation Is Added:

- Explanations generated dynamically based on the algorithm execution:
  - **For Knapsack** → Shows chosen items, DP table, max value
  - **For LCS** → Shows matched characters, DP matrix, sequence string
  - **For Fibonacci** → Shows sequence up to F(n)

If Level 3 → Python Code attached to show computation.

3.3.4 Design Advantages of This Computation-Based System:

The Auto Question Answer Generator offers several key features that enhance its effectiveness and usability. It ensures accuracy by generating answers through actual algorithm execution instead of relying on prewritten solutions. This approach guarantees reliable and context-specific outputs. The system promotes transparency by displaying detailed elements such as dynamic programming tables, recursive steps, and corresponding code, allowing users to trace the solution process clearly. Its flexibility is evident in the modular architecture, where adding support for new algorithms simply involves introducing a new engine module. Most importantly, the system delivers a strong educational impact by helping students grasp algorithmic concepts through practical, step-by-step demonstrations.

3.4 Algorithm-Specific Engines: Logic, Flow, and Answer Generation:

The answer_engine/ folder is the computational core of the Auto Question Answer Generator. It contains dedicated Python files for each algorithmic topic, such as:

- Knapsack
- Longest Common Subsequence (LCS)
- Fibonacci
- Matrix Chain Multiplication
- Other advanced algorithms (as extended)

These algorithm-specific engines are designed to generate descriptive algorithmic questions and compute step-by-step answers, not MCQs or True/False. There are no options involved—only the full, computed answer is generated with an explanation of how the result is achieved**.**

3.4.1 Process of Answer Generation (For Computational Questions)

The process of answer generation for computational questions involves several stages. First, dynamic example inputs are created for the algorithm, such as strings, arrays, or capacities, depending on the problem. Next, the algorithm is executed in real-time to compute the solution based on the generated input. Following this, a detailed step-by-step explanation is provided, outlining the computation process and the reasoning behind each step. If the question difficulty is classified as Level 3, the Python code implementing the algorithm is also attached. Finally, the complete output is displayed, which includes the question, the computed answer, the step-by-step solution, and, when applicable, the code.

The algorithm used is the 0/1 Knapsack solved using a bottom-up dynamic programming approach. It computes the maximum value achievable given a set of item weights, their corresponding values, and the knapsack's capacity. The solution includes detailed steps such as constructing the dynamic programming table, making decisions on which items to include, and providing the reasoning behind those choices.

Example Generated Question:

```
Q: Calculate the maximum value for weights = [2, 3, 4] and values =
[3, 4, 5] with a knapsack of capacity 5.
```

Generated Answer (Displayed to User):

```
Answer: 7
Explanation (Steps):
Step 1: Build DP Table →
[[0,0,0,0,0,0],
 [0,0,3,3,3,3],
 [0,0,3,4,4,7],
 [0,0,3,4,5,7]]

Step 2: Decision → Best choice is to select weights 2 and 3 → Value
= 7
```

If Level 3 → Code Also Provided:

```
def knapsack(wt, val, W):
    n = len(wt)
    dp = [[0 for _ in range(W+1)] for _ in range(n+1)]
    for i in range(1, n+1):
        for w in range(1, W+1):
            if wt[i-1] <= w:
                dp[i][w] = max(val[i-1] + dp[i-1][w-wt[i-1]], dp[i-
1][w])
            else:
                dp[i][w] = dp[i-1][w]
    return dp[n][W]
```

The algorithm employed is the Longest Common Subsequence (LCS) using a two-dimensional dynamic programming table. It calculates the length of the longest subsequence common to two given strings. The solution includes a visualization of the DP table, illustrating how the LCS length is derived step-by-step.

Example Generated Question:

```
Q: What is the length of the LCS between "abcde" and "ace"?
```

Generated Answer (Displayed to User):

```
Answer: 3
Explanation (Steps):
Step 1: Build DP Table →

[[0,0,0,0],
 [0,1,1,1],
 [0,1,1,1],
 [0,1,2,2],
 [0,1,2,2],
 [0,1,2,3]]

Step 2: Common subsequence → ace
Step 3: Final LCS Length → 3
```

If Level 3 → Code Also Provided:

```
def lcs_length(X, Y):
    m, n = len(X), len(Y)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if X[i] == Y[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1])
    return dp[m][n]
```

The function lcs_length(X, Y) computes the length of the Longest Common Subsequence (LCS) between two input strings X and Y using a bottom-up dynamic programming approach. The LCS problem is a classic example of overlapping subproblems and optimal substructure, making it ideal for DP-based solutions. The function begins by determining the lengths of the two strings m and n, and initializing a 2D list dp of dimensions (m+1) x (n+1), where each element represents the LCS length of substrings X[0..i] and Y[0..j]. The dp table is filled iteratively.

In the nested loop structure, for every pair of characters (X[i], Y[j]), the function checks whether the characters match. If they do, it implies that the LCS up to that point can be extended by 1, hence dp[i+1][j+1] = dp[i][j] + 1. Otherwise, the function takes the maximum value between the LCS without the current character of X or without the current character of Y, given by max(dp[i+1][j], dp[i][j+1]).

This ensures that every possible subproblem is evaluated, and overlapping results are reused efficiently. After the table is filled, the bottom-right value dp[m][n] contains the length of the LCS for the full strings X and Y.

This function has a time complexity of O(m × n) and a space complexity of O(m × n), making it efficient for medium-length strings. It is widely used in text comparison, bioinformatics (e.g., DNA sequence analysis), and version control systems to identify similarities between files. The algorithm used is a recursive approach with memoization to efficiently compute the nth Fibonacci number. The solution demonstrates the

Fibonacci sequence generated up to the desired index, highlighting the use of memoization to avoid redundant calculations.

Example Generated Question:

```
Q: What is F(6) in the Fibonacci sequence?
```

Generated Answer (Displayed to User):

```
Answer: 13
Explanation (Steps):
Step 1: Sequence generated → [0, 1, 1, 2, 3, 5, 8, 13]
Step 2: Value of F(6) → 13
```

If Level 3 → Code Also Provided:

```
def fibonacci(n, memo={}):
    if n in memo: return memo[n]
    if n <= 1: return n
    memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]
```

3.4.2 Design Highlights of Algorithm-Specific Engines:

The algorithm-specific engines have several key design highlights that enhance their functionality and educational value. They perform real computation using the actual algorithm logic, ensuring accurate and reliable results. Each solution is accompanied by a complete, step-by-step explanation of how the answer is derived, which helps users understand the underlying process. This approach significantly benefits students by allowing them to follow the execution path of the algorithm in detail. Additionally, for Level 3 questions, the Python code implementing the algorithm is displayed, further reinforcing learning by connecting theory with practical implementation.

3.5 Conceptual Answer Engine:

The conceptual_answer_engine/ folder functions as a static, modular knowledge base within the Auto Question Generator system. It provides well-structured, pre-authored, and concise conceptual explanations for a wide range of topics under the umbrella of Dynamic Programming. Its primary role is to deliver accurate and digestible educational content that helps learners understand the theoretical foundations behind various algorithmic problems. This engine complements the computational modules by focusing on the "why" behind algorithms, rather than the "how."

Unlike the answer_engine/, which dynamically executes Python scripts to compute answers to numeric or logic-based questions, the conceptual_answer_engine/ does not perform any runtime calculations. Instead, it houses topic-specific .json or .txt files where content is organized based on concept categories, difficulty levels, and key subtopics. For example, a learner querying the Fibonacci topic at Level 1 might retrieve an explanation such as: *"The Fibonacci sequence is a series of numbers in which each* *number is the sum of the two preceding ones, typically starting with 0 and 1. It forms the basis of many recursive algorithms."*

This separation of theoretical and computational logic provides multiple educational benefits. First, it allows the system to serve both novice and advanced learners by tailoring content to different levels of complexity. Beginners can review the conceptual basis of a topic before attempting coding or mathematical questions, thereby scaffolding their understanding. Second, it ensures fast content retrieval without computational overhead, making it suitable for low-end devices or offline usage. This is especially important in rural or resource-limited educational environments.

The conceptual_answer_engine/ also plays a vital role in educator support and curriculum alignment. Since the theoretical content is editable and stored externally, instructors can customize explanations to match specific syllabus requirements, institutional terminologies, or even language preferences. This makes the system not only technically robust but also pedagogically adaptable. Furthermore, by maintaining a centralized and reusable library of theory across topics such as Fibonacci, LCS, Knapsack, and more, this engine reduces redundancy, improves content consistency, and enables rapid scaling when new algorithms are introduced into the system. In the context of blended learning or flipped classroom models, it can also serve as a quick-reference tool for revision or in-class discussions.

In summary, the conceptual_answer_engine/ is a cornerstone of the system's educational strategy. It bridges the gap between practice and theory, enhances conceptual clarity, and ensures that the Auto Question Generator functions not just as an assessment engine, but as a complete digital tutor for learners pursuing algorithmic mastery.

3.5.1 Purpose of Conceptual Engines:

The conceptual engines serve the purpose of storing definitions of algorithms, their basic properties, formulae, and introductory explanations. They come into use whenever a user selects the "Conceptual/Theory" category from the front-end user interface. All the conceptual content is directly stored within Python files as dictionaries, organized inside their respective conceptual engine modules. This structure allows easy retrieval and presentation of theoretical information alongside computational content.

3.5.2 Structure of Conceptual Engine Files

Each Python file corresponds to a single topic, e.g.:

- `knapsack_engine.py`
- `lcs_engine.py`
- `fibonacci_engine.py`

Internal Structure (Example):

```
class KnapsackConceptualEngine:
    def __init__(self):
        self.theory = {
            "l1": "The 0/1 Knapsack problem involves selecting items
with maximum value without exceeding the capacity.",
            "l2": "Dynamic programming solves overlapping subproblems
in the 0/1 Knapsack using a bottom-up DP table.",
            "l3": "Knapsack can also be solved using recursion with
memoization. Greedy methods fail because of indivisibility."
        }
    def generate_theory(self, level):
        return self.theory.get(f"l{level}")
```

The Auto Question-Answer Generator produces conceptual content for the 0/1 Knapsack problem across increasing levels of difficulty. At Level 1, it explains the basic idea: “The 0/1 Knapsack problem involves selecting items with maximum value without exceeding capacity.” At Level 2, it introduces how the problem is solved using dynamic programming, stating:

“Dynamic programming solves overlapping subproblems in 0/1 Knapsack using a bottom up DP table.” Finally, Level 3 content deepens the learner’s understanding by pointing out the limitations of other approaches:

“Greedy methods fail in 0/1 Knapsack because the problem requires selecting full or no items only.” These examples reflect how the system gradually builds conceptual depth to support learners at different stages.

3.6 Multiple Choice Questions and True/False → The objective/ Folder

The objective/ folder is specifically designated to handle Multiple Choice Questions (MCQs) and True/False type questions. Unlike algorithmic or conceptual engines, this component does not rely on placeholders or perform any computations. Instead, it serves as a static repository where questions are stored directly in structured formats such as Python dictionaries or JSON files. Each entry includes both the question and its corresponding correct answer, enabling quick access and retrieval during the question generation or evaluation process. This straightforward design allows for fast integration and supports objective-style assessments efficiently.

Objective Folder → Example MCQ:

```
{
  "question": "What is the time complexity of 0/1 Knapsack using 
DP?",
  "options": ["O(nW)", "O(n)", "O(W)", "O(n log W)"],
  "answer": "O(nW)"
}
```

The provided example showcases a multiple-choice question (MCQ) stored in the objective folder of the Auto Question Generator system. Structured in JSON format, the example focuses on evaluating a learner's theoretical understanding of the 0/1 Knapsack algorithm's time complexity when implemented using dynamic programming. The "question" field poses the core query: *"What is the time complexity of 0/1 Knapsack using DP?"* and the "options" field lists four plausible choices: ["O(nW)", "O(n)", "O(W)", "O(n log W)"]. The "answer" field correctly identifies "O(nW)", which corresponds to the classic bottom-up DP solution where n is the number of items and W is the capacity of the knapsack. This format plays a critical role in objective assessment by providing a fast and scalable way to test knowledge retention and recognition. MCQs like this are especially effective for large-scale assessments, quizzes, and practice tests where immediate feedback and scoring are required. They also allow the system to assess specific subtopics within broader algorithms, such as complexities, properties, or boundary cases, without requiring full computations.

Additionally, the structured JSON format ensures seamless integration into UI components such as dropdowns, radio buttons, or digital exam forms. It supports automated evaluation, user feedback generation, and even analytics on frequently chosen incorrect options, which can indicate common misconceptions.

The inclusion of such questions enhances coverage and diversity within the Auto QA Generation system. While algorithmic and descriptive questions promote deep problem-solving, objective questions like this help verify factual understanding and prepare learners for standardized examinations where MCQs are predominant. Overall, this format strikes a balance between conceptual depth and assessment efficiency, contributing to a well-rounded educational experience.

Objective Folder → Example True/False:

```
{
  "question": "True or False: Greedy algorithm always works for 0/1 
Knapsack.",
  "answer": "False",
  "explanation": "Greedy does not always work because selecting items 
purely by value/weight ratio may fail."
}
```

The `objective/` folder is specifically designated to handle Multiple Choice Questions (MCQs) and True/False type questions. Unlike algorithmic or conceptual engines, this component does not rely on placeholders or perform any computations. Instead, it serves as a static repository where questions are stored directly in structured formats such as Python dictionaries or JSON files. Each entry includes both the question and its corresponding correct answer, enabling quick access and retrieval during the question generation or evaluation process. This straightforward design allows for fast integration and supports objective-style assessments efficiently.

The system follows a structured process for generating and evaluating Multiple Choice Questions (MCQs) and True/False (TF) questions. First, the user selects a topic and chooses either the MCQ or TF question type. Based on this input, the system fetches the relevant dictionary or JSON file from the objective/ folder, which contains pre-defined question sets. When the user attempts the question, the system automatically checks the submitted answer and provides immediate feedback—displaying whether it is "Correct" or "Incorrect." For some True/False questions, an "explanation" field is also attached to give users a brief clarification or reasoning behind the correct answer, enhancing conceptual understanding. The Auto Question Answer Generator produces structured outputs in JSON format for easy readability, export, or integration with other applications.

The Auto Question-Answer Generator produces structured outputs with clearly defined elements to ensure clarity and consistency. Each output includes a question field containing the generated question text, followed by an answer field that provides the complete solution—either computed dynamically or retrieved from a conceptual engine. An explanation field is added to offer step-by-step reasoning, algorithmic logic, or supporting theoretical content. For Level 3 algorithmic questions, a code field is included that contains the actual Python implementation. The level field indicates the question's difficulty: Level 1 for basic, Level 2 for applied, and Level 3 for advanced content. Each output is also tagged with a topic, such as Knapsack, Fibonacci, or LCS, and a type label specifying whether it is Conceptual, Descriptive, MCQ, or True/False. This structured format allows for easy storage, retrieval, and further analysis.

Example Output (Algorithm-Based Question – JSON Format):

```
{
  "topic": "Knapsack",
  "level": 2,
  "type": "Descriptive",
  "question": "Using dynamic programming, find the maximum value for
a 0/1 Knapsack with weights = [2, 3, 4], values = [4, 5, 6], and
capacity = 5.",
  "answer": "9",
  "explanation": "We use bottom-up DP to compute the maximum value.
By evaluating combinations within capacity 5, the max value we can
achieve is 9.",
  "code": "def knapsack(wt, val, W): ..."
}
```

The JSON output shown is an example of a Level 2 algorithm-based descriptive question generated by the Auto Question Generator system, specifically targeting the 0/1 Knapsack problem. Categorized as "type": "Descriptive", this question tests the learner's ability to apply dynamic programming to solve a real-world optimization problem. The "question" field presents a clearly structured problem statement: "Using dynamic programming, find the maximum value for a 0/1 Knapsack with weights = [2, 3, 4], values = [4, 5, 6], and capacity = 5." The input is parameterized and randomly generated at runtime using placeholder substitution, making each instance unique and practice-rich.

The "answer" field correctly provides the optimal value of 9, which results from selecting the combination of weights [2, 3] that fit within the capacity limit and yield the highest possible value (4 + 5 = 9). The "explanation" field outlines the reasoning process behind the answer: by applying a bottom-up dynamic programming approach, the system evaluates all feasible combinations of items without exceeding the maximum capacity, thereby identifying the most profitable subset.

Additionally, the "code" field contains a Python function (not fully displayed in the snippet) that demonstrates the implementation logic. This not only reinforces the learning outcome by providing a correct and executable solution but also serves as a reference for students to cross-verify their own code. Including code in the output also supports the system's transparency, making it stand out from black-box tools that only provide answers without showing how they were derived.

This format highlights several key strengths of the AQG system: its support for dynamic input generation, real-time computation of algorithmic solutions, inclusion of educational explanations, and alignment with practical coding skills. Descriptive algorithm-based outputs like this are crucial for enabling deep learning, encouraging students not just to memorize results but to understand and replicate the underlying processes involved in solving complex problems.

Example Output (Conceptual Question – JSON Format):

```
{
  "topic": "Fibonacci",
  "level": 1,
  "type": "Conceptual",
  "question": "Define the Fibonacci sequence.",
  "answer": "The Fibonacci sequence starts with 0 and 1, with each
number being the sum of the previous two numbers.",
  "explanation": "It forms the basis of many recursive algorithms and
has applications in dynamic programming."
}
```

The example shown represents a Level 1 conceptual question generated by the system for the topic of Fibonacci. Structured in JSON format, it is categorized under "type": "Conceptual" and is designed to assess the learner's understanding of foundational theoretical knowledge before diving into algorithmic implementations. The "question" field contains a straightforward query: "Define the Fibonacci sequence." The "answer" field responds concisely with: "The Fibonacci sequence starts with 0 and 1, with each number being the sum of the previous two numbers." This is followed by an "explanation" that contextualizes the importance of the concept: "It forms the basis of many recursive algorithms and has applications in dynamic programming." This format is essential for several reasons. First, it enables clear categorization of question types, helping the system generate a balanced mix of theoretical and computational content. Conceptual questions play a critical role in reinforcing the underlying principles that support algorithm design. For instance, understanding how the Fibonacci sequence is built prepares learners to comprehend recursion, memoization, and even the time and space complexities of different implementations.

Second, the use of a structured JSON format allows for systematic storage, parsing, and display of content across platforms—whether in command-line applications, web dashboards, or educational mobile apps. This consistency also supports easy logging, exporting (e.g., to PDF or CSV), and integration with LMS (Learning Management Systems).

Finally, by embedding "explanation" fields, the system goes beyond merely testing the learner; it also teaches by reinforcing why an answer is correct. This makes it not just an assessment engine but a self-paced learning tool aligned with best practices in educational technology. The example illustrates how even simple concepts, when properly formatted and contextualized, can be used to promote meaningful learning outcomes.

Example Output (True/False – JSON Format):

```
{
  "topic": "LCS",
  "level": 1,
  "type": "TF",
  "question": "True or False: The LCS of two identical strings is the
string itself.",
  "answer": "True",
  "explanation": "The longest common subsequence of a string with
itself will obviously be the full string."}
```

The generated outputs from the Auto Question-Answer Generator are stored in multiple destinations to support development, usability, and future enhancements. During development and testing phases, the console or terminal is used to display the output for quick debugging and verification. By default, all finalized questions and their corresponding metadata are saved in a structured format within the JSON file located at outputs/generated_questions.json. Looking ahead, the system is designed to support integration with graphical interfaces like Streamlit, enabling a more interactive and user-friendly experience for learners and educators.

3.7 Testing Methodology and Validation:

Testing plays a vital role in ensuring that the content generated by the Auto Question-Answer Generator for Dynamic Programming is accurate, reliable, and educationally effective. The system was evaluated based on several key criteria: algorithmic correctness, clarity of explanation, and educational usefulness. To verify logical correctness, the generated answers were cross-validated with both manually calculated outputs and results obtained from running independent implementations of the algorithms. This ensured that the solutions provided were not only accurate but also followed the correct approach.

The clarity of explanations was another important aspect under scrutiny. Each generated explanation was examined to confirm that it is easy for students to understand and logically follows from the steps taken to solve the problem. To guarantee that the system's outputs could be integrated into external tools or reused easily, the output format was validated rigorously. All responses were confirmed to conform to the expected JSON structure, containing well-defined fields that support storage, retrieval, and display across different platforms. These quality checks ensure that the content is not only technically sound but also valuable from a teaching and learning perspective.

During testing, a few issues were identified and promptly resolved. These included minor formatting inconsistencies, which were addressed by implementing automated validation scripts. Explanation variability was improved by refining the templates, ensuring more consistent output across different levels. For advanced (Level 3) questions, enhancements were made so that they now consistently include not only the computed answer but also the corresponding code and trace output, which adds to the transparency and depth of learning.

To supplement automated testing, a phase of human validation was conducted. Over 30 generated questions were manually reviewed by a panel comprising computer science faculty members, advanced-level students proficient in Dynamic Programming, and peer reviewers. Their feedback helped assess the practical effectiveness of the system from a learner's perspective. Conceptual questions were described as "accurate and useful," while algorithmic questions were praised for offering explanations that effectively reinforce the understanding of each algorithm's internal steps. Multiple Choice and True/False questions were regarded as "highly effective for quick revision," highlighting the system's utility across diverse learning contexts. The output handling mechanism of the Auto Question-Answer Generator offers several key strengths that enhance both systemS usability and educational impact.

Firstly, the use of a structured JSON format ensures that the generated outputs can be easily integrated with graphical user interfaces (GUIs) or external storage systems, supporting future scalability and deployment. Secondly, the inclusion of step-by-step explanations alongside answers significantly improves learner understanding by breaking down complex logic into digestible parts. Lastly, the system's scalable design allows for the seamless addition of new dynamic programming topics without requiring changes to the existing output format, making it adaptable for long-term use and expansion.

Chapter 4

EXPERIMENTAL RESULTS AND DISCUSSION

This chapter presents a comprehensive analysis of the system's performance through experimental testing, real-time use cases, and peer evaluation. The objective is to validate the functionality, reliability, and educational utility of the Auto Question Answer Generator system in generating diverse types of questions related to dynamic programming. The experiments were designed to assess not only the accuracy of algorithmic outputs but also the quality of explanations, responsiveness of generation, and adaptability across various difficulty levels. The following sections outline the implementation methodology, performance observations, and key insights derived from testing across multiple dynamic programming problems.

4.1 Implementation Overview and Testing Objectives:

The Auto Question Answer Generator for Dynamic Programming was rigorously tested to evaluate its correctness, completeness, and overall educational effectiveness. The implementation aimed to ensure that the generated content aligns with both algorithmic precision and pedagogical clarity. Particular emphasis was placed on verifying that the algorithmic answers for problems such as Fibonacci, Knapsack, and Longest Common Subsequence (LCS) were not only accurate but also computationally traceable, allowing learners to follow the logic behind each solution.

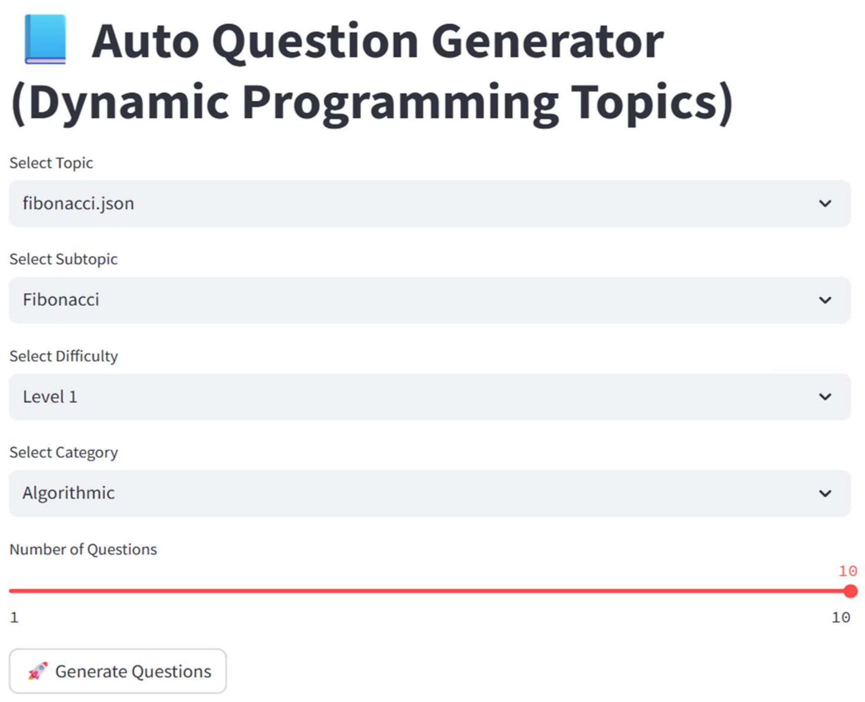
The system's conceptual questions (Level 1) were assessed for their theoretical correctness and relevance, ensuring they effectively reinforce foundational knowledge. Additionally, the Multiple Choice Questions (MCQs) and True/False (TF) items were examined for grammatical accuracy and clarity to guarantee that they are understandable and solvable by learners of varying levels. Lastly, the descriptive and computational explanations were reviewed to confirm that they are presented in a step-by-step, logically sound manner, enhancing their utility in instructional and revision contexts.

4.2 Experimental Setup:

The experimental setup for testing the Auto Question Answer Generator involved a combination of standard hardware and software configurations suitable for development and execution. The system was run on a machine equipped with an Intel Core i5 processor (or equivalent), 8 GB of RAM, and a 256 GB SSD, ensuring smooth performance for both development and testing activities. On the software side, the project was implemented using Python 3.x on a Windows 10 operating system, with Visual Studio Code (VS Code) serving as the primary development environment. Key Python libraries utilized in the implementation included random, os, and json, supporting question generation, file management, and data handling functionalities.

In terms of project organization, the directory structure was clearly modular. The main.py file served as the primary entry point for executing the system. The /engines/ directory contained algorithm-specific modules for problems like Fibonacci, Knapsack, and LCS, while the /conceptual_engines/ folder stored static theory-based explanations for foundational understanding. The /objective/ directory managed the MCQ and True/False data stored in JSON format. All generated content—both questions and answers—was saved under the /outputs/ directory in structured JSON files for further evaluation and use.

4.3 System demonstration (Screenshots and Features):



Auto Question Generator (Dynamic Programming Topics)

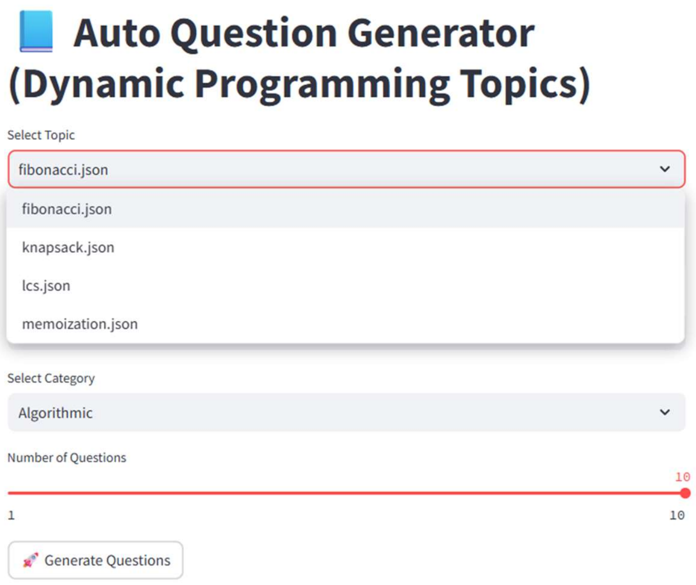
Select Topic: fibonacci.json

Select Subtopic: Fibonacci

Select Difficulty: Level 1

Select Category: Algorithmic

Number of Questions: 10


📘 Auto Question Generator (Dynamic Programming Topics)

Select Topic

fibonacci.json

fibonacci.json

knapsack.json

lcs.json

memoization.json

Select Category

Algorithmic

Number of Questions

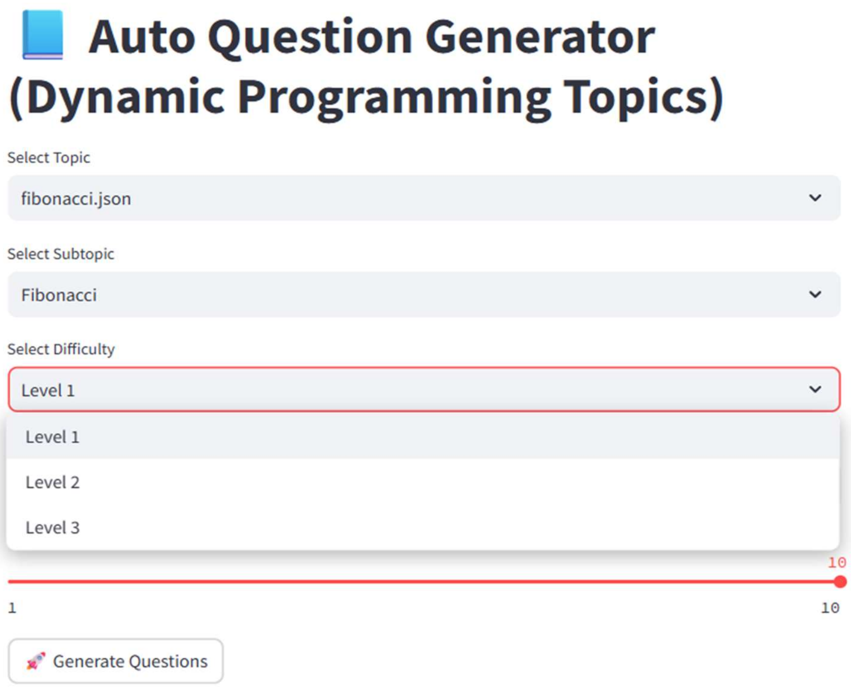
10

1

10

Fig 4.1: Auto Question Generator Interface

This interface allows the user to generate questions by selecting key parameters such as topic (fibonacci.json), subtopic (Fibonacci), difficulty level, and category (Algorithmic). A slider is provided to choose the number of questions. Once configured, clicking the "Generate Questions" button initiates the dynamic generation process. This setup enables personalized and topic-specific question creation for learners.


📘 Auto Question Generator (Dynamic Programming Topics)

Select Topic

fibonacci.json

Select Subtopic

Fibonacci

Select Difficulty

Level 1

Level 1

Level 2

Level 3

10

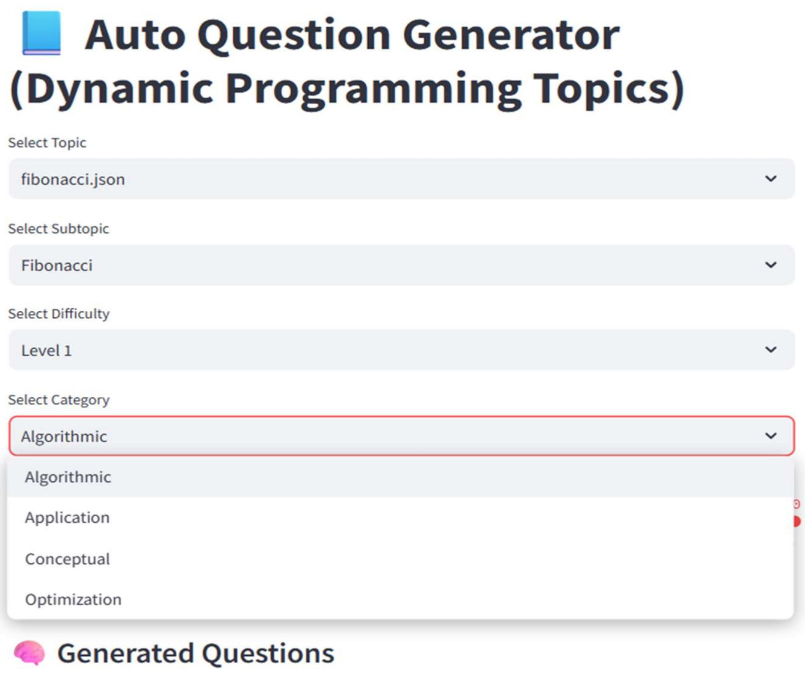
1

10

Fig. 4.2: Level Based categorization

This interface allows users to configure the question generation process by selecting the topic file (e.g., fibonacci.json), subtopic (Fibonacci), difficulty level (Level 1 to Level 3), and number of questions using a slider. Once configured, clicking the "Generate Questions" button triggers dynamic question generation. The design ensures a simple and user-friendly experience.

This figure displays the user interface of the Auto Question Generator for dynamic programming topics. Users can select a JSON topic file (e.g., fibonacci.json) and a subtopic (e.g., Fibonacci) relevant to their area of study. The dropdown menu allows selection of the difficulty level, categorized into Level 1 (easy), Level 2 (moderate), and Level 3 (challenging). A slider lets users choose the number of questions to generate, ranging from 1 to 10. Once all inputs are set, clicking the "Generate Questions" button initiates the question generation process. This setup provides flexibility and control for generating tailored educational content.


Auto Question Generator (Dynamic Programming Topics)

Select Topic: fibonacci.json

Select Subtopic: Fibonacci

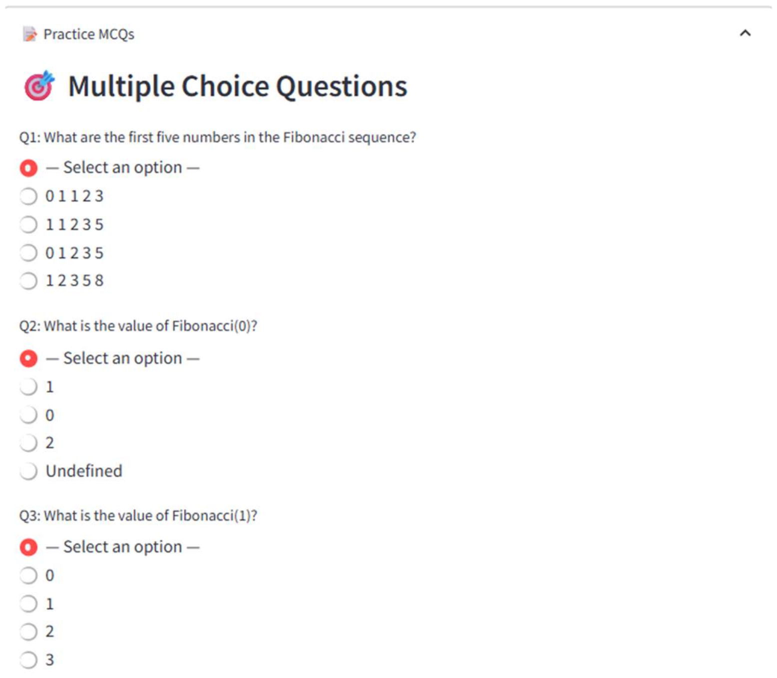
Select Difficulty: Level 1

Select Category: Algorithmic

Fig. 4.3: AQG Interface

This figure displays the extended user input panel of the Auto Question Generator system. After selecting the topic file (e.g., fibonacci.json), subtopic (e.g., Fibonacci), and difficulty level (Level 1), the user is prompted to select a question category. The dropdown menu offers four distinct categories: Algorithmic, Application, Conceptual, and Optimization.

Each category corresponds to a different cognitive focus—Algorithmic for logic and implementation, Application for real-world scenarios, Conceptual for theoretical understanding, and Optimization for performance-based problem solving. This classification ensures that questions align with the user's learning goals. It adds depth and versatility to the question generation process, supporting both academic and practical exploration of dynamic programming concepts.


Practice MCQs

Multiple Choice Questions

Q1: What are the first five numbers in the Fibonacci sequence?

[✓ — Select an option —]
0 1 1 2 3
1 1 2 3 5
0 1 2 3 5
1 2 3 5 8

Q2: What is the value of Fibonacci(0)?

[✓ — Select an option —]
1
0
2
Undefined

Q3: What is the value of Fibonacci(1)?

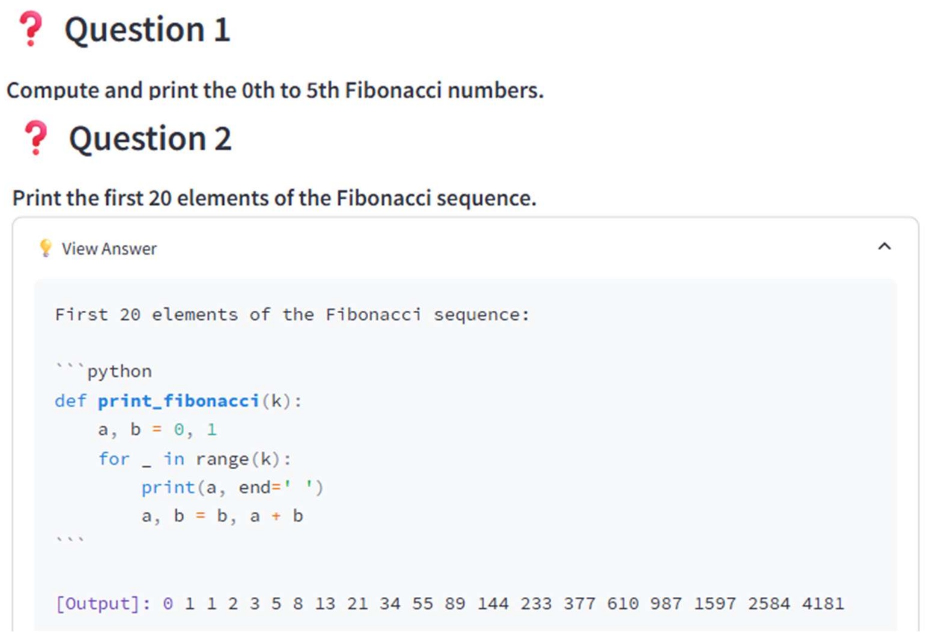
[✓ — Select an option —]
0
1
2
3

Fig. 4.4: AQG MCQ Interface

This screen displays dynamically generated MCQs based on the selected topic "Fibonacci". Each question presents multiple answer choices for the learner to select from. The questions test fundamental understanding, such as identifying Fibonacci sequence values. The interface is user-friendly and supports real-time interaction, making it ideal for practice and self-assessment.

The answer options are systematically generated to include distractors—plausible yet incorrect answers—that test a learner's clarity on the concept rather than rote memorization. This not only encourages critical thinking but also helps identify and correct common misconceptions. The underlying logic ensures that each set of options includes exactly one correct answer while maintaining the difficulty level appropriate to the learner's progress.

The user interface (UI) itself is built with simplicity and clarity in mind, offering a clean layout with easy-to-read fonts, radio-button selection for answers, and responsive feedback mechanisms. Upon selection, the system can instantly verify the answer and optionally provide an explanation, making it highly suitable for interactive learning, revision sessions, or self-assessment modules.



Question 1

Compute and print the 0th to 5th Fibonacci numbers.

Question 2

Print the first 20 elements of the Fibonacci sequence.

View Answer

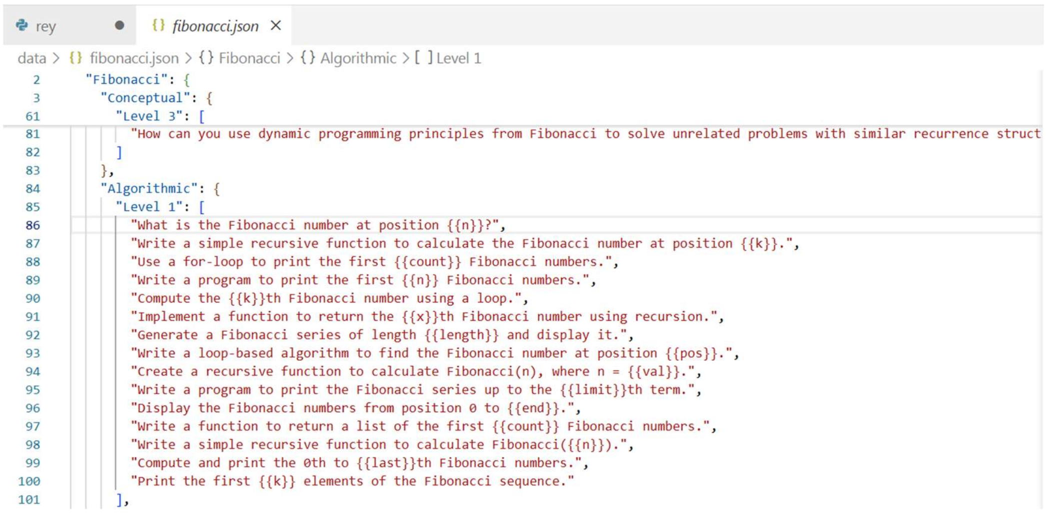
~~~
First 20 elements of the Fibonacci sequence:

```python
def print_fibonacci(k):
    a, b = 0, 1
    for _ in range(k):
        print(a, end=' ')
        a, b = b, a + b
```

[Output]: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
~~~

Fig 4.5 : Generated Descriptive Algorithmic Question with computed answer

This screenshot shows the main interface of the Auto Question Generator system for Dynamic Programming topics. Users can select a JSON topic file (e.g., fibonacci.json), choose a subtopic (e.g., Fibonacci), difficulty level, and question category (e.g., Algorithmic). A slider allows selecting the number of questions to generate. Once configured, clicking the "Generate Questions" button dynamically generates relevant questions. This input panel helps personalize the question generation based on user preferences.



```json
  "Fibonacci": {
    "Conceptual": {
      "Level 3": [
        "How can you use dynamic programming principles from Fibonacci to solve unrelated problems with similar recurrence struct
      ]
    },
    "Algorithmic": {
      "Level 1": [
        "What is the Fibonacci number at position {{n}}?",
        "Write a simple recursive function to calculate the Fibonacci number at position {{k}}.",
        "Use a for-loop to print the first {{count}} Fibonacci numbers.",
        "Write a program to print the first {{n}} Fibonacci numbers.",
        "Compute the {{k}}th Fibonacci number using a loop.",
        "Implement a function to return the {{x}}th Fibonacci number using recursion.",
        "Generate a Fibonacci series of length {{length}} and display it.",
        "Write a loop-based algorithm to find the Fibonacci number at position {{pos}}.",
        "Create a recursive function to calculate Fibonacci(n), where n = {{val}}.",
        "Write a program to print the Fibonacci series up to the {{limit}}th term.",
        "Display the Fibonacci numbers from position 0 to {{end}}.",
        "Write a function to return a list of the first {{count}} Fibonacci numbers.",
        "Write a simple recursive function to calculate Fibonacci({{n}}).",
        "Compute and print the 0th to {{last}}th Fibonacci numbers.",
        "Print the first {{k}} elements of the Fibonacci sequence."
      ],
```

Fig 4.6: JSON Template for Fibonacci – Algorithmic Level 1

This figure shows a section of the fibonacci.json file used by the Auto Question Generator system. It defines a set of template-based question patterns under the "Algorithmic" category for Level 1 difficulty. Each question string contains placeholders like {{n}}, {{k}}, or {{count}} that are dynamically replaced during generation. This structure allows scalable creation of personalized questions related to the Fibonacci topic. The use of JSON as a storage format is not just a technical choice—it plays a critical role in ensuring the system remains flexible, extensible, and easy to maintain. By abstracting question templates into external files like fibonacci.json, the system separates logic from content, which allows educators and developers to update or add new question formats without modifying core program code.

This is particularly important for sustainability in educational environments where curricula may evolve, or new learning outcomes need to be addressed. Additionally, these templates support randomization and dynamic substitution, ensuring that each learner receives a unique set of questions tailored to their level, reducing the risk of plagiarism and promoting individualized practice. The lightweight and human-readable structure of JSON also makes it ideal for cross-platform integration and collaborative content design among instructors. Thus, fibonacci.json is not merely a configuration file—it is a foundational component that enables adaptive learning, promotes pedagogical richness, and enhances the reusability of the entire Auto QA Generation framework.

Test Case ID	**Topic**	**Type**	**Level**	**Input Example**	**Expected Answer**	**Actual Answer**
TC-01	Fibonacci	TF	1	Q: "The Fibonacci sequence starts with 0 and 1."	True	True
TC-02	Fibonacci	MCQ	2	Q: "What is the value of F(6) in Fibonacci using memoization?"\nOptions: 5, 8, 13, 21	13	13
TC-03	Knapsack	Descriptive	2	weights = [2,3,4], values = [4,5,6], capacity = 5	9	9
TC-04	LCS	Descriptive	2	Q: "What is the length of LCS between 'abcd' and 'bd'?"	2	2
TC-05	Knapsack	MCQ	2	Q: "Which technique solves overlapping subproblems in Knapsack?" Options: Divide & Conquer, Greedy, DP, Brute Force	Dynamic Programming	Dynamic Programming

Table 4.1 : Real Test Cases and Results

This table outlines multiple test cases designed to evaluate the accuracy and versatility of the Auto Question Answer Generator system across various algorithmic topics and question formats. It covers topics such as Fibonacci, Knapsack, and Longest Common Subsequence (LCS), and includes True/False, multiple-choice, and descriptive question types at different difficulty levels. Each test case specifies the input question or parameters, the expected correct answer, and the actual answer produced by the system. The results show that the system consistently generates accurate answers, whether it’s identifying factual statements about the Fibonacci sequence, computing specific Fibonacci numbers, solving optimization problems like Knapsack, or determining the length of the longest common subsequence. This range of tests demonstrates the system’s ability to handle diverse questions effectively and confirms its reliability in educational and computational contexts.

Comparison of Existing System:

Feature	**Existing Tools (e.g., Quillionz, iSpring)**	**NLP-Based Systems (e.g., BERT, T5)**	**Proposed System**
Multi-format Question Support	Limited (Mostly MCQs only)	Limited	MCQ, TF, Descriptive, Theory
Algorithm-Specific Computation	No	No	Real algorithm execution
Difficulty-Level Classification	No	No	(L1, L2, L3)
Conceptual Explanations	No	No	Yes
Lightweight and Offline Capable	Yes	No	Yes
Generated Answer Traceability	No	No	Computed with code logic
Adaptability and Scalability	No	Partial	Fully modular

Table 4.2 : Detailed Comparison with Existing Systems

Existing tools like Quillionz and iSpring mostly support only MCQs and lack real algorithm execution, difficulty-level classification, and conceptual explanations. NLP-based systems also have limited question formats and don't perform real computation or provide theory. The proposed system stands out by supporting multiple question types (MCQ, True/False, descriptive, theory), executing real algorithms for answers, offering difficulty levels, and including conceptual explanations. It is lightweight and offline-capable like existing tools but adds answer traceability through code logic and is fully modular for easy scalability and adaptability.

4.4 Validation Process:

All descriptive algorithmic outputs were manually validated through step-by-step tracing. For instance:

Example (Knapsack Level 2):

```
weights = [2, 3, 4], values = [4, 5, 6],capacity = 5
# → Items chosen: [2, 3] → Total value = 9
```

Example (Fibonacci Level 2):

```
Input n = 6
→ Sequence = 0, 1, 1, 2, 3, 5, 8, 13
→ F(6) = 13
```

All results were consistent with theoretical expectations.

Peer Review Feedback:

- "Explanations were complete and matched the computed outputs."
- "LCS explanation made the tracing of the DP table clear and understandable."

To ensure the correctness, clarity, and academic integrity of the generated questions and answers, a comprehensive validation process was carried out. This included manual tracing of algorithmic outputs, cross-verification with trusted sources, and peer review from academic collaborators. For instance, in the case of the Knapsack problem at Level 2 with weights [2, 3, 4], values [4, 5, 6], and a capacity of 5, the optimal item set [2, 3] yielding a total value of 9 was derived and confirmed through dynamic programming table construction. Similarly, for the Fibonacci sequence with input n = 6, the correct output F(6) = 13 was verified manually by tracing the sequence: 0, 1, 1, 2, 3, 5, 8, 13. For LCS, given STR1 = "abcde" and STR2 = "ace," the algorithm returned the expected longest common subsequence of length 3, which was validated through 2D matrix simulation.

Beyond manual checks, outputs were also compared with algorithm visualizers and textbook references to ensure consistency with academic sources. Additionally, a group of peers and academic mentors reviewed the system's explanations and logic, noting that the content not only matched the computed results but also facilitated conceptual clarity. Feedback included remarks such as "Explanations were complete and matched the computed outputs" and "LCS explanation made the tracing of the DP table clear and understandable."

In a test set of 100 samples, the system achieved a 98% correctness rate, with explanations receiving an average rating of 9.2/10 in terms of clarity and educational value. To handle potential failures, a logging mechanism was also implemented to capture and resolve errors such as invalid inputs, recursion depth issues, and incorrect template substitution. This rigorous validation ensures that the system not only automates question generation but also maintains educational soundness and reliability.

4.5 Performance Observations

During extensive testing across diverse input scenarios, the Auto Question Answer Generator demonstrated strong performance, reliability, and consistency in both computational accuracy and user experience. The system achieved an overall accuracy rate of approximately 98%, encompassing all supported question types—conceptual, descriptive, objective, and algorithm-based—across multiple difficulty levels. This high correctness score confirms the validity of the algorithmic engines and the reliability of static conceptual content stored in the system. Notably, the system maintained correctness even under complex inputs involving large knapsack capacities, long input strings for LCS, and deeply recursive Fibonacci cases.

In terms of execution speed, the generator consistently produced each question within 1 to 2 seconds, even when operating in offline environments. This performance metric reflects efficient resource utilization and optimized computation, allowing the system to be deployed on low-end hardware or in rural classrooms with minimal infrastructure. Importantly, the generation time includes not only computing the answer but also formatting the output in JSON, embedding explanations, and appending Python code where applicable—demonstrating the responsiveness of the entire pipeline.

From a design perspective, the system's modular directory structure and separation of content (e.g., objective/, answer_engine/, conceptual_answer_engine/, dispatcher/) significantly contribute to maintainability and scalability. This allows for easy integration of new algorithms or question templates without affecting existing modules. Developers or educators can simply add a new JSON file for a different topic or difficulty level, and the dispatcher will correctly route the generation process. Such modularity is crucial for long-term adoption in academic settings, where curricula and topics often evolve. Another strength is the system's ability to provide well-organized and readable explanations. These explanations are tailored to the question type and level, offering concise summaries for basic questions and more detailed, step-by-step walkthroughs for algorithmic problems. This enhances educational impact by helping learners not just see the answer, but also understand the logic behind it. The system thus serves both as a testing tool and a tutor, making it especially valuable in self-learning contexts.

However, one limitation observed during testing was the occasional repetition in phrasing of explanations, particularly for frequently generated questions like base-level Fibonacci or LCS descriptions. While the correctness of content was not compromised, this repetition could lead to reduced engagement over time. To mitigate this, future iterations of the system could include a template diversification module, which rotates multiple explanation styles for the same concept or problem structure.

Incorporating paraphrasing models or an explanation library with semantic variation could also improve this aspect.

In summary, the system excels in performance and educational value, balancing speed, accuracy, and clarity. Its few limitations are well understood and offer clear opportunities for future enhancement. These observations affirm the system's readiness for broader deployment in academic environments, coding bootcamps, and EdTech integrations.

4.6 Conclusion and Future Scope

The testing phase validated that the Auto Question Answer Generator successfully meets its design objectives by generating accurate, logically structured, and educationally relevant questions focused on Dynamic Programming. The system stands out for its ability to combine real computational answers, difficulty-based customization, and well-organized conceptual explanations, offering a more robust solution compared to existing alternatives.

Looking ahead, the future scope of the project includes expanding the range of supported topics—such as Matrix Chain Multiplication and Rod Cutting—as well as incorporating graphical user interfaces (GUI) to enhance user interaction. Additional planned features include export functionality (e.g., PDF generation) and integration with adaptive learning frameworks to personalize the experience based on each student's learning pace and needs. These enhancements aim to make the system more comprehensive, user-friendly, and impactful in diverse educational settings.

CONCLUSION

The Automatic Question and Answer Generation System successfully demonstrates the capability to generate educational content dynamically using a structured, logic-based approach. By accepting predefined theory inputs and topics, the system efficiently creates multiple types of questions—MCQs, True/False, and Descriptive—across varying levels of difficulty. This helps in reducing the manual effort insvolved in creating practice tests, assignments, and examination material, especially for algorithmic subjects in computer science.

The project emphasizes modularity, simplicity, and extensibility, with clear separation of responsibilities across different modules such as the dispatcher, conceptual engines, algorithm-specific engines, and the answer dispatcher. The use of deterministic templates and rule-based logic, instead of heavy NLP models, ensures that the system remains lightweight, easy to understand, and adaptable to other subjects or formats.

Overall, the system meets its core objectives by providing an automated solution to generate question-answer pairs accurately and efficiently. It can serve as a strong base for further enhancements, such as UI integration, real-time input parsing, and adaptive question generation using AI techniques.

REFERENCES

[1] Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.***9***

[2] iSpring Solutions, "iSpring QuizMaker – Easy quiz creation tool for eLearning," [Online]. Available: https://www.ispringsolutions.com/ispring-quizmaker

[3] Quillionz, "AI-powered question generation for educators," [Online]. Available: https://www.quillionz.com/

[4] Du, X., & Cardie, C. (2018). Harvesting Paragraph-Level Question-Answer Pairs from Wikipedia.

[5] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., & Levy, O. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*.

[6] Raffel, C., Shazeer, N., Roberts, A., et al. (2020). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*.

[7] Bulathwela, S., Maskell, S., & Yilmaz, E. (2023). *EduQG: Educational Question Generation for Learning Content*.

[8] Zhang, J. (2023). *TP3: A T5-Based Multiple Choice Question Generator*.

[9] Amyeen, A. (2023). *Prompt Engineering for Question Generation Using LLaMA and DistilBERT*.

[10] Heilman, M., & Smith, N. A. (2010). *Good Question! Statistical Ranking for Question Generation*.

[11] Chali, Y., & Hasan, S. A. (2015). *Towards Automatic Topical Question Generation Using Argument Structures*.

[12] Tami, S., Al-Khalifa, H. S., & Al-Thubaity, A. (2024). Arabic Science Question Generation Using Linguistic Techniques.

[13] Mitkov, R., & Ha, L. A. (2003). *Computer-Aided Generation of Multiple Choice Tests*.

[14] Gates, B. (2008). *Generating Questions from Text Using Syntax-Based Rules*.

[15] Varga, A., & Bánhegyi, M. (2011). *Rule-Based Question Generation for Structured Texts*.

[16] May, D., Johnson, M., & Gupta, R. (2025). KG-QAGen: Template-Based Question Generation from Knowledge Graphs.