Intelligent Question Paper Generation using AI and ML Algorithms for Tailored Difficulty Levels

Tejasvi Vijay Panchal[1], Lakshya Singh Chouhan[2]

Indian Institute of Teacher Education, Sector 15, Gandhinagar, 382016, India[1]

Poornima Institute of Engineering and technology, Sitapura, Jaipur, 302012, India[2]

Corresponding Authors: Tejasvi Vijay Panchal

Email: tejasvi007panchal@gmail.com; Mobile: +91-7014614791

Funding or Supporting Agencies and Grants Details: Self-funded, and no external funding or supporting agencies were involved.

ABSTRACT

Automated question paper generation has emerged as a prominent field in educational settings, driven by advancements in artificial intelligence and machine learning. This research paper presents a comprehensive approach that harnesses advanced language models and machine learning techniques to generate question papers with diverse difficulty levels and types. The methodology encompasses several key steps. To begin, we employ the pandas and os libraries in Python for data preparation. Pandas, a versatile tool for data manipulation and analysis, facilitates the creation of a structured DataFrame from questions and labels extracted from text files. The os module, on the other hand, aids in managing files and directories, enabling efficient iteration over files and content retrieval. Data cleaning is crucial, and we accomplish it by employing regular expressions with the re module. This step sanitizes the input question text, removing unwanted characters and ensuring a cleaner and more uniform dataset. Next, we train machine learning models using the popular sklearn library. The dataset is split into training and testing sets using the sklearn.model_selection.train_test_split function, allowing us to train the models on the larger training set and evaluate their performance on the testing set. To transform the textual data into a format suitable for machine learning models, we utilize the sklearn.feature_extraction.text.CountVectorizer. This process, known as vectorization, converts the text into a matrix of token counts, facilitating subsequent analysis. For the classification task, we adopt the sklearn.naive_bayes.MultinomialNB algorithm, renowned for its efficacy in text classification. This algorithm is trained using the feature matrix from the CountVectorizer and the corresponding labels. Evaluation of the models' performance is achieved through the sklearn.metrics.accuracy_score function, which compares the predicted labels with the actual labels from the testing set. To facilitate future use without retraining, the trained models, along with the CountVectorizer, are saved using the joblib library. In the question classification and storage stage, new questions are classified using the trained models and stored in separate files. These questions are imported from text files, cleaned using the aforementioned data cleaning techniques, and then vectorized. The trained models are utilized to classify the questions, and the results are saved in separate text files based on the predicted labels. This organization enables efficient retrieval of questions according to their classification, simplifying the generation of specific question paper types. Moreover, we fine-tune the GPT-2 model using the Transformers library, a powerful language model. This fine-tuning process occurs on classified question datasets, enabling the generation of unique and contextually relevant questions. Additionally, we utilize BERT, a highly effective model in Natural Language Processing (NLP), for text classification. A pre-trained BERT model is fine-tuned specifically for sequence classification, enabling the categorization of questions based on their relevance. To enhance the quality of the generated questions, we incorporate a question filter script that assesses relevance and grammatical correctness. This script identifies and eliminates irrelevant or grammatically incorrect questions, thus improving the overall quality of the generated question set. To create well-structured question papers, we employ multiple pre-trained models capable of generating questions categorized by cognitive domains (such as knowledge, comprehension, application, analysis, synthesis, and evaluation) and

section types (including very short, short, and long). This approach ensures a balanced representation of different question types and difficulty levels in the generated question papers. Finally, we employ the convert_to_pdf function, utilizing the reportlab library, to convert the generated question paper into a PDF format. This conversion process simplifies dissemination and sharing of the question paper output. The evaluation of our model yields promising results, with an overall accuracy of approximately 74.2% in predicting Bloom's taxonomy categories. Notably, the model demonstrates strengths in predicting 'comprehension' and 'evaluation'. Additionally, the question type classification model exhibits high accuracy in identifying 'not relevant' questions, a crucial aspect for maintaining question paper quality. However, further improvements are required in distinguishing between 'long,' 'short,' and 'very short' questions. Furthermore, aligning the generated questions with the class 10 science course syllabus is a priority for future work, as the current limitations in the dataset hindered full alignment. This research contributes significantly to the advancement of automated question paper generation systems. It highlights the importance of leveraging advanced language models, machine learning algorithms, and effective data preprocessing techniques. The findings provide valuable insights into the strengths and areas for improvement in our proposed methodology, laying the foundation for further research to refine and enhance these systems.

Keywords: . Automated Question Paper Generation; Machine Learning and Artificial Intelligence; Natural Language Processing (NLP); Python Libraries (sklearn, pandas, Transformers); Pre-trained Models (GPT-2, BERT); Text Classification and Vectorization

1. INTRODUCTION

The rapidly evolving era of Artificial Intelligence (AI) and Machine Learning (ML) algorithms opens up a myriad of opportunities to revolutionize numerous aspects of our lives, one of which is the field of education (Bishop, 2006). Traditionally, question paper generation has been a manual and time-consuming process, often resulting in question papers that do not optimally align with the varying learning needs and abilities of students. Recognizing this gap, this research aims to introduce an intelligent question paper generation system using AI and ML algorithms, tailored to create diverse difficulty levels, thereby catering to the unique learning requirements of every student.

The context of this research is established amidst the current educational landscape where standardized assessments often fail to accurately measure a student's understanding or aptitude (4). By integrating AI and ML algorithms into the process of question paper generation, the objective is to create a more dynamic, personalized, and effective evaluation method (1). The purpose is not only to improve the efficiency of the process but also to augment the quality of education and assessments.

The problem being addressed here is the largely uniform and standardized nature of assessment papers that often neglect the unique learning pace and understanding level of each student. The outcome can potentially lead to skewed academic results and may not foster inclusive learning. This research aspires to provide a tailored solution to this issue, presenting a unique and personalized approach to assessments. With the advent of sophisticated machine learning models such as the Multinomial Naive Bayes algorithm (1), GPT-2 (2), and BERT (3), among others, the feasibility and applicability of AI in creating personalized question papers has significantly increased. The proposed system leverages these state-of-the-art ML algorithms to classify, generate, and evaluate questions based on specific parameters such as relevance and grammatical correctness, thus providing a tailored question paper for each student.

The study's primary goal is to design, implement, and evaluate an AI-based question paper generation system capable of generating diverse difficulty levels and ensuring the relevance and grammatical correctness of questions (3). To achieve this, various ML algorithms are employed, including, but not limited to, the Multinomial Naive Bayes algorithm for classification (1), GPT-2 for question generation (2), and BERT for relevance assessment (3). The research also explores the integration of various Python libraries such as Pandas, os, and re for data manipulation, cleaning, and file management, respectively (4).

The research's significance is two-fold. Academically, it contributes to the emerging field of AI in education by presenting a practical and scalable solution for tailored question paper generation. Practically, it offers a potential tool that can revolutionize the way we approach assessments, fostering a more inclusive, personalized, and effective learning environment.

The research will primarily involve developing and testing multiple ML models to carry out specific tasks, including data preprocessing, model training and evaluation, question classification and storage, and eventually question generation. The system will use different ML models for different tasks,

working in an integrated manner to achieve the desired output (40).

In summary, this research proposes to leverage AI and ML algorithms to create an intelligent question paper generation system, marking a significant step towards personalized and effective education. By doing so, it attempts to address the need for a more inclusive, comprehensive, and tailored approach to assessments, ultimately contributing to improved teaching and learning outcomes.

2. REVIEW

This meta-analytic review endeavors to examine key developments in the field of machine learning, natural language processing (NLP), and educational taxonomies, with a specific focus on their application to pattern recognition, language modeling, and automated question generation.

As per (1), set the foundation for pattern recognition and machine learning, exploring different techniques, mathematical frameworks, and potential applications. This work forms the basis for many subsequent advancements in machine learning (1). The role of machine learning is further complemented by scikit-learn, a powerful library providing efficient tools for statistical modeling, including classification, regression, clustering, and dimensionality reduction (7 and 8).

In the domain of NLP, advancements have been led by transformer-based models. Transformer architectures have revolutionized NLP tasks by focusing on self-attention mechanisms that better understand the context within sequences of words. In (3) introduced BERT (Bidirectional Encoder Representations from Transformers), a pre-training model that has significantly improved the state-of-the-art across a broad array of tasks. OpenAI's GPT-2 (2) and GPT-3 (5) built upon this work and demonstrated the versatility of these models in an unsupervised setting. These models can generate coherent, contextually relevant, and grammatically correct sentences. The capabilities of these models were extensively studied by (4), outlining their potential in a range of tasks.

The educational field has been enriched by Bloom's Taxonomy, which categorizes cognitive skills into six levels: knowledge, comprehension, application, analysis, synthesis, and evaluation (5). This taxonomy offers a blueprint for structuring educationally relevant questions at varying difficulty levels.

On the computation front, PyTorch has become a preferred tool for researchers due to its flexibility and efficiency (10). This is further supplemented by Adam, an optimization algorithm that has been widely adopted for its efficiency in dealing with large-scale problems (13). In (11), extended Adam with decoupled weight decay regularization to improve the algorithm's performance further.

Accuracy estimation and model selection have been studied by (14) who proposed cross-validation and bootstrap methods. These techniques are crucial for verifying the effectiveness and generalization capability of the models.

In (16), proposed a rule-based grammar checker, a vital tool for ensuring the grammatical correctness of generated content. Similarly, in (17), has provided Python users with a library for PDF processing, enabling the output of generated content in a readily accessible format.

In conclusion, the field of automated question generation has significantly benefited from advancements in machine learning, NLP, and education research. The intersection of these disciplines allows for the creation of sophisticated models capable of generating, evaluating, and sorting questions based on their relevance, complexity, and grammatical correctness.

3. METHODS

Model 1

This research involves an automated question paper generator, implemented using various Python libraries(Pedregosa et al., 2011). The main steps of the methodology are data preparation, model training, model saving, question classification, and storage of classified questions.

Data Preparation: The initial phase of the project involves preparing the data for model training. For this, the Python libraries pandas (McKinney, 2010) and os are used.

Pandas: The Pandas library is a powerful tool used for data manipulation and analysis in Python. Here, it is employed to create a DataFrame from the questions and labels read from text files. A DataFrame is a two-dimensional labeled data structure where columns can potentially be of different types. The data is organized in a tabular form that is easy to manipulate and analyze, ideal for our machine learning model.

OS: The os module in Python is used for interacting with the operating system. Here, it is primarily used for managing files and directories. Specifically, it enables the code to iterate over all

files in the provided directories, read the content of each file, and create new directories for storing classified questions.

Data cleaning is performed using the function clean_question, defined using Python's re module.

RE: The re module in Python provides support for regular expressions, enabling us to manipulate strings using various rules and patterns. In the clean_question function, it is used to sanitize the input question text, removing unwanted characters and helping in making the dataset cleaner and more uniform.

Model Training: This research utilizes two machine learning models for classifying the questions. They are trained using the create_model function which uses several components from the sklearn library (Pedregosa et al., 2011).

sklearn.model_selection.train_test_split: This function is used to split the dataset into two parts: a training set and a testing set. The training set is larger and used to train the model. The testing set is used to evaluate the model's performance.

sklearn.feature_extraction.text.CountVectorizer:

CountVectorizer is used to convert the text data into a matrix of token counts. This process, known as vectorization, is a crucial step in preprocessing for Natural Language Processing (NLP). It transforms textual data into a format that can be processed by the machine learning model.

sklearn.naive_bayes.MultinomialNB: The Multinomial Naive Bayes algorithm from sklearn's naive_bayes module is used as the machine learning model in this project. It is a popular choice for text classification tasks. The algorithm is trained using the feature matrix from the CountVectorizer and the corresponding labels. It uses the principles of Bayes' Theorem but with strong independence assumptions.

Model Evaluation: Model performance is gauged using sklearn's accuracy_score function (Pedregosa et al., 2011).

sklearn.metrics.accuracy_score: After the model has been trained and has made predictions on the testing set, accuracy_score is used to compute the accuracy of those predictions. This is done by comparing the predicted labels with the actual labels from the testing set.

Model Saving: Once the models are trained, they are saved to disk along with the CountVectorizer using the joblib library.

joblib: Joblib is a Python library used for saving and loading Python objects that involve large data arrays. Here, it is used to serialize the trained models and the vectorizer, saving them to the disk. This allows the models to be reused later without retraining.

Question Classification and Storage:

Finally, new questions are classified using the trained models. The questions are imported from a text file, cleaned using the clean_question function, vectorized, and then passed through the models for classification. The classified questions are then saved in separate text files named after the predicted labels from both models, facilitating efficient retrieval for specific types of question papers.

Working: This research presents a three-fold methodology for an automated question paper generator, integrating training of classifiers, classification of datasets, and management of classified data.

1. Training of Classifiers:

In the initial phase, two distinct text classification models are trained. The training datasets comprise two types: one based on the length of the questions (very short, short, long) and the other based on Bloom's taxonomy (knowledge, comprehension, application, analysis, synthesis, evaluation) (5).

The first model is trained to identify the cognitive level of the question based on Bloom's taxonomy. Bloom's taxonomy provides a framework that categorizes educational learning objectives into levels of complexity and mastery (5). By training a model on this taxonomy, the system can identify whether a given question tests knowledge, comprehension, application, analysis, synthesis, or evaluation.

The second model is trained to identify the length of the question. It discerns whether the question is very short, short, or long. This classification is valuable for constructing a balanced and well-distributed question paper.

The models are trained using a dataset that pairs questions with their corresponding classifications. The sklearn's train_test_split function is used to split the dataset into training and testing sets. The questions are then vectorized using the CountVectorizer function, transforming the text data into a format that can be

processed by the machine learning models. These vectorized questions form the basis for training the Multinomial Naive Bayes Classifier (MultinomialNB), a popular choice for text classification tasks (9).

2. Classification of Datasets:

Once the models are trained, they are used to classify all questions in the datasets. Each question is processed through both models, resulting in a dual classification. For instance, a question might be classified as 'short' in terms of length and 'evaluation' in terms of Bloom's taxonomy. The system would then label this question as 'short-evaluation'. This two-fold classification system enables a rich, multi-dimensional organization of the questions.

Additionally, the models are capable of classifying a set of unclassified questions provided in a separate text file. This feature, however, is optional and is used based on the requirements of the user.

3. Management of Classified Data:

Following classification, the system stores the questions in an organized manner. For each unique classification label, a separate directory is created (if it doesn't already exist). Each question is then saved in its corresponding directory, denoted by its predicted label.

For example, if an unclassified question gets classified as 'short-knowledge', it is added to the 'short-knowledge' directory. This automated system not only categorizes questions but also provides an efficient retrieval mechanism, aiding in the speedy compilation of specific types of question papers.

The trained models, along with the vectorizer, are saved using joblib to allow future reuse without retraining. This method promotes system efficiency and offers the flexibility to process new, unclassified questions at any time.

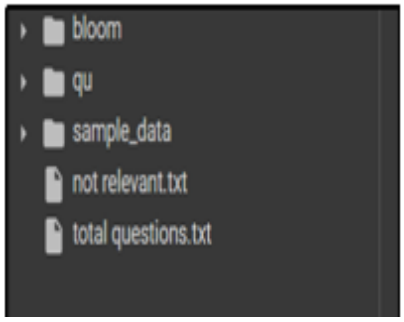
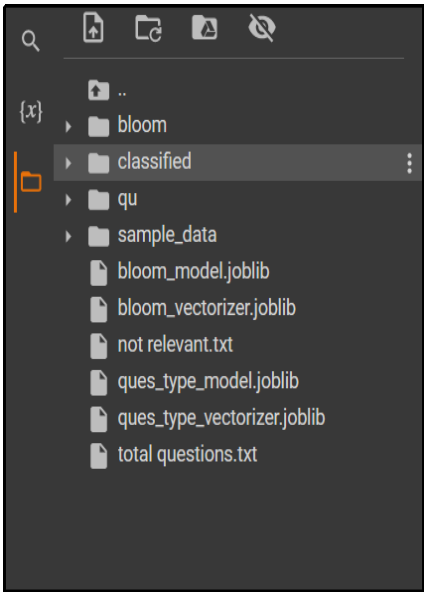


Image 1: Initial system layout with 'qu' and 'bloom' folders containing text files named for question lengths and Bloom's taxonomy categories, respectively, before code execution.



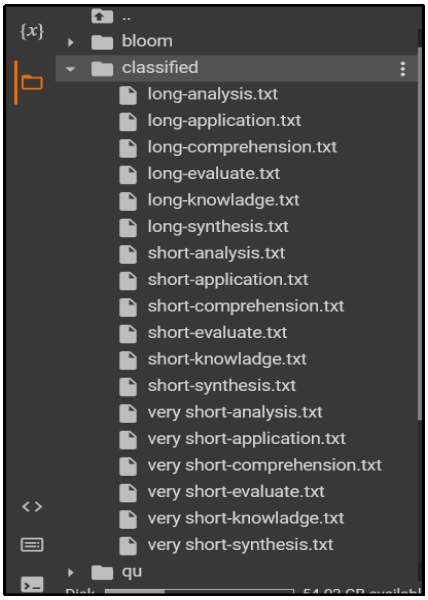


Image 2 and 3: Post-classification system structure showcasing original 'bloom' and 'qu' folders, alongside a new 'classified' folder. The latter contains files named for question length and Bloom's category combinations, such as 'short-knowledge.txt', illustrating the system's dual categorization for efficient question retrieval.

Model 2

The objective of this method is to fine-tune the GPT-2 model, a large transformer-based language model, on classified question datasets for the generation of unique questions (2). This process utilizes the Transformers library developed by Hugging Face (4). The fine-tuning operation unfolds in the following steps:

1. Installation and Importation of Libraries: Initially, the transformers and accelerate libraries are installed using pip, Python's package installer. The transformers library, by Hugging Face, provides pre-trained models that facilitate various tasks in

Natural Language Processing (NLP) (4). In this research, it is used for fine-tuning the GPT-2 model.

The accelerate library, also developed by Hugging Face, is a PyTorch utility designed for easy multi-GPU and distributed training. The inclusion of this library is intended to optimize the process of model fine-tuning, ensuring efficient utilization of available computational resources.

Following installation, the necessary classes and methods from the transformers library are imported. These components are crucial for the tokenization process, model handling, trainer specification, training argument formation, and data collation.

2. Tokenization: An instance of AutoTokenizer is created using the pre-trained GPT-2 large model (4). Tokenization involves breaking down sentences into smaller units—words, subwords, or symbols—so they can be processed by the model. The tokenizer converts the text data into a format understandable by the model, transforming input text into an array of integers representing the underlying semantic content.

3. Dataset Loading: A load_dataset function is defined to return a TextDataset object, which is a PyTorch Dataset. This object contains the tokenized text data from a specified file, prepared for training. The TextDataset class from the transformers library reads a text file and converts it into a suitable dataset for training the transformer model.

4. Data Collation: A DataCollatorForLanguageModeling is defined, a specific collator for language modeling tasks. It is a function that batches examples from the dataset and prepares them for training or evaluation. It helps mask tokens for a masked language modeling objective, which is a common training method for transformer models.

5. Path Gathering: All text file paths from the classified folder are collated into dataset_paths, ensuring that each file of classified questions is identified and included in the fine-tuning process.

6. Model Fine-Tuning: The crux of the operation is the fine-tuning of the GPT-2 model. For each file in dataset_paths, a new instance of the GPT-2 large model is created. The corresponding dataset is loaded, and training arguments are defined using the TrainingArguments class. This class specifies training parameters such as the output directory, number of training epochs, and batch size.

The Trainer class creates a training loop for PyTorch, optimized for transformers. It utilizes the model, training arguments, data collator, and dataset to fine-tune the model. Consequently, each model is fine-tuned on a particular dataset, allowing the model to generate questions specific to its category.

7. Model Saving: After fine-tuning, the configuration of the model and the model itself are saved for future use. These saved models can generate unique questions corresponding to their training datasets, thereby enhancing the variety and uniqueness of question papers (2).

In summary, this methodology leverages the GPT-2 model's power to generate unique, category-specific questions. It presents a robust technique for question paper generation, thereby enabling a diverse and dynamic examination landscape.

Working: The principal goal of this phase in the methodology is the fine-tuning of the GPT-2 large model on the various classified datasets created in the preceding step. This fine-tuning procedure is designed to equip the GPT-2 model with the capability to generate unique questions pertaining to each category established during classification, including but not limited to classifications such as short-evaluation, long-analysis, and very short-knowledge. The results of this fine-tuning process are subsequently stored for future usage and retrieval.

Fine-Tuning the GPT-2 Model:

To commence the fine-tuning process, the GPT-2 model is instantiated for each category in the classified datasets. Each model instance is fine-tuned on the classified datasets, which have been prepared according to categories based on Bloom's taxonomy and question type (very short, short, long).

The fine-tuning process essentially consists of training the model on a new task using the pre-existing model weights. This method allows the model to apply its pre-trained knowledge to the new task, reducing the required training time and data. In this case, the GPT-2 model, which is pre-trained on a large corpus of internet text, is fine-tuned on the specific task of generating unique questions for each category.

Fine-tuning is carried out using the Trainer class from the Hugging Face's transformers library. The Trainer class handles training and evaluation of the model. It utilizes the model, training arguments, data collator, and the classified datasets to fine-tune each model. This process equips each model to

generate unique questions, consistent with the specifics of their respective category.

Storing Fine-Tuned Models:

Upon completion of the fine-tuning operation, each model, now specifically trained for generating questions from a distinct category, is stored for future use. The configuration of the model and the model itself are saved using the save_model() function.

The results are stored in a designated directory, './results'. Each category gets its own sub-directory within this folder, matching the respective categories, for example, ./results/short-evaluation, ./results/long-analysis, ./results/very short-knowledge, and so forth.

In essence, this methodology equips the GPT-2 model to generate category-specific unique questions. The fine-tuning process enables a granular approach to question generation, thus providing the capacity to generate a wide variety of questions aligned with specific pedagogical objectives. This fine-tuned GPT-2 model thereby adds dynamism and depth to the landscape of automated question paper generation.

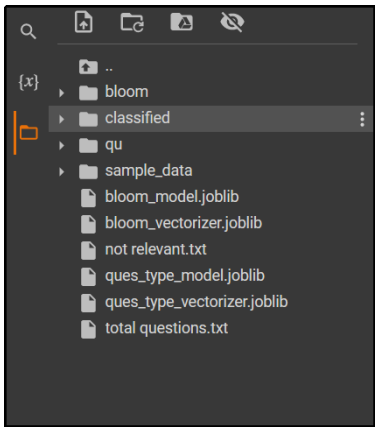
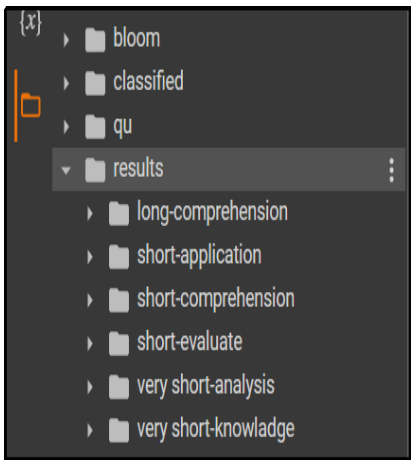


Image 4: Initial system state before GPT-2 model fine-tuning, indicated by the absence of the 'results' folder for storing fine-tuned models.



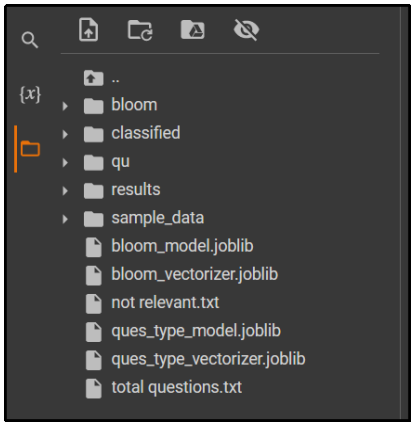


Image 5 and 6: Transformed system state after GPT-2 model fine-tuning, signified by the 'results' folder containing category-specific sub-directories, each housing a fine-tuned model for generating questions in the corresponding category, thereby improving automated question generation.

Model 3

Methods

The purpose of this step in our methodology is to prepare a text dataset for a classification task. This task's objective is to discern the relevance of a question, separating questions into relevant and non-relevant categories. The employed model for this step is BERT (Bidirectional Encoder Representations from Transformers), known for its effectiveness in numerous Natural Language Processing tasks, including text classification (3).

Preparing the Dataset:

Data is loaded using the defined function load_data. The function reads a text file, and each line is subjected to formatting for uniformity; all leading and trailing white spaces are stripped, and the text is converted to lower case. Each line (question) is then assigned a label to signify its relevance. Relevant and non-relevant data are subsequently loaded, combined into a single dataset, and shuffled for unbiased model training.

The dataset's cleaning process follows next, where all punctuation marks are removed from the questions using the clean_question function, enhancing the efficiency of the subsequent tokenization step.

Tokenizing and Conversion to PyTorch Tensors:

Tokenization for this task utilizes BERT's tokenizer, instantiated as BertTokenizerFast, using the pre-trained 'bert-base-uncased' model (4). The tokenizer is not only responsible for converting the text data into a model-friendly format, but also for creating attention masks that differentiate padding tokens from non-padding tokens.

After tokenization, the resultant data and attention masks are converted into PyTorch tensors. The labels for relevance are also converted into tensor format, thus achieving compatibility with the PyTorch-based model.

Data Splitting and DataLoader Creation:

Once the tensors are prepared, the data is split into training and validation sets using the train_test_split function from sklearn.model_selection library (7). The split ratio employed is 90:10, signifying that 90% of the data is used for training, and the remaining 10% is used for validation.

To prepare the data for training and validation, DataLoader objects are created. For the training set, a RandomSampler is employed. The usage of random sampling helps in facilitating stochastic optimization. On the other hand, the validation set uses a SequentialSampler.

In summary, this phase in the methodology is concerned with creating a well-organized pipeline for preparing a text classification dataset. This pipeline spans from raw text data to ready-to-use DataLoader objects, which are primed for model training.

Significant Models and Libraries

The methodology in question leverages several noteworthy models and libraries to accomplish the text classification task:

xformers: This library, installed via the command !pip install xformers, houses an extensive collection of transformer models. These models are optimized for performance across diverse hardware and applications.

language_tool_python: Installed via the command !pip install language_tool_python, this library is a Python wrapper for LanguageTool, facilitating grammar checking in Python.

BertTokenizerFast: This class, part of the transformers library, is a fast tokenizer for the BERT model (4). It is crucial for the transformation of raw text data into a format ingestible by the model.

PyTorch (torch): PyTorch is a renowned open-source machine learning library based on the Torch library (10). It is used to manipulate tensors, the core data structures in PyTorch, and perform various operations on them.

Train_test_split & shuffle: These functions are part of the sklearn library (7). They are utilized to split datasets into training and validation subsets randomly and shuffle the data to ensure unbiased model training.

TensorDataset, DataLoader, RandomSampler, SequentialSampler: These classes, part of PyTorch's torch.utils.data module, are used to wrap the dataset into PyTorch's tensor format, load the data in batches, and perform sampling for both training and validation datasets (10).

The code combines the strengths of these models and libraries to load, clean, and tokenize the data. It then prepares the data for training a BERT model by converting it into PyTorch tensors and creating DataLoader objects. The results are ready-to-use DataLoader objects that can be directly used for training a model.

Model 4

The script under discussion represents a procedure for fine-tuning a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model for sequence classification, a task in Natural Language Processing (NLP) that involves assigning predefined categories to sequences of words or sentences (3). This is an important component of our automated question paper generation system, enabling it to categorize generated questions based on their relevance.

To start, we import several essential libraries. This includes transformers, a popular NLP library that provides state-of-the-art pre-trained models and other NLP utilities (4), torch for handling tensor computations (10), and AdamW and BertConfig from the transformers library for model optimization and configuration (11).

We employ the BertForSequenceClassification model, a variant of the BERT model equipped with a classification layer on top. This model is ideal for tasks that involve classifying a sequence into one of two or more categories, a binary classification in our case. The model parameters are loaded using the .from_pretrained() function, which fetches the pre-trained model weights (4).

Once the model is set, we initialize an AdamW optimizer. This is a variant of the Adam optimizer that corrects its weight decay handling, a crucial aspect of optimizing model performance

during the training process. The learning rate and epsilon parameters are set according to best practices.

The script then checks if a GPU (Graphics Processing Unit) is available for computation. Using a GPU accelerates the training process due to its ability to perform many computations simultaneously. If a GPU is not available, the script falls back to using the CPU (Central Processing Unit).

To manage the learning rate during the training process, the script creates a scheduler using the get_linear_schedule_with_warmup function. This function reduces the learning rate linearly with each epoch, following a specified 'warmup' period where the learning rate is increased.

To enhance the reproducibility of the training process, a random seed is set, ensuring that any random operations will produce the same result each time the script is run (12).

The model is then trained over several epochs, iterating over the entire training dataset in each epoch. In each iteration, the model takes in a batch of training data, performs forward propagation to compute the loss, backward propagation to compute gradients, and updates the model parameters using the optimizer. The learning rate is also updated according to the schedule (13).

After each epoch, the model is validated using a validation dataset. This dataset is different from the training dataset and is used to evaluate the model's performance on unseen data, providing a measure of its generalization ability. The validation process does not involve updating the model parameters (14).

The script outputs the training loss, validation accuracy, and the time taken for each epoch, providing an ongoing report of the model's performance during training. Once all epochs are complete, the script indicates that training is complete. With this trained model, we can classify generated questions according to their relevance, a key step in our automated question paper generation process.

Model 5

The script in focus is designed to function as a question filter, assessing the relevance and grammatical correctness of questions. It is a key component of our automated question paper generation system, as it filters out irrelevant or grammatically incorrect questions, enhancing the quality of the generated question set (15).

The script begins with the initialization of two critical tools: the BERT tokenizer and a LanguageTool object for English (US) grammar checking. The BERT tokenizer, provided by the transformers library, transforms raw text input into a format understood by the BERT model, which is a sequence of integers representing the text (4). LanguageTool, on the other hand, is a powerful open-source grammar checker in Python. It's initialized to check for grammatical correctness based on US English norms (16).

A function named check_relevance_and_sort(questions) is central to this script. It takes in a list of questions and iteratively processes each one to gauge its relevance and grammatical correctness.

For each question, the BERT tokenizer first converts it into input IDs, a sequence of numerical representations of the text. This tensor is then moved to the computational device (GPU if available, else CPU).

Next, the relevance of the question is evaluated. The input IDs are fed into a trained BERT model, which returns a set of logits or raw model outputs. These logits are indicative of the question's relevance. Using torch.argmax(), the function identifies the index of the highest logit value. In this context, this index is understood to represent relevance (3).

If the question is deemed relevant, the script proceeds to check its grammar using LanguageTool. LanguageTool returns a list of "matches," with each match representing a grammatical error found in the question (16).

The question is filtered based on this grammar check. If the count of matches, i.e., grammatical errors, is zero, it implies that the question is grammatically correct. Such a question is considered 'relevant' and is added to a list of relevant questions.

After processing all the questions, the function check_relevance_and_sort() returns the list of relevant and grammatically correct questions, achieving the objective of filtering out unsuitable questions from the larger set (15).

To summarize, the script utilizes the BertTokenizer from the transformers library, LanguageTool from the language_tool_python library, and tensor operations from the torch library to filter and sort questions based on their relevance and grammatical correctness. This is a crucial step in enhancing the quality of our automated question paper generation system, ensuring that the final set of questions is both topically appropriate and grammatically sound (3, 4, and 16).

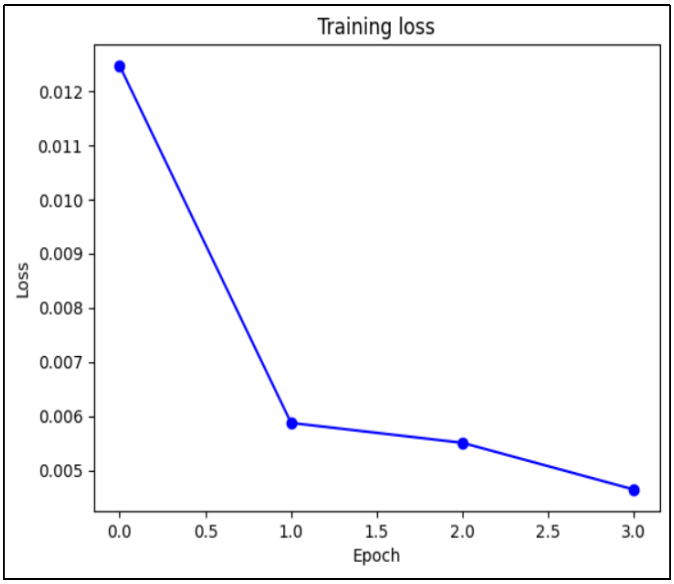


Image 7: The image is a line graph, titled “Training loss”, that plots the decreasing trend of loss values (close to 0.01) over four epochs, indicating an effective training process in a machine learning model.

Model 6

This script leverages multiple pre-trained models to generate questions based on their category, which are knowledge, comprehension, application, analysis, synthesis, and evaluation, as well as their section type, which includes very short, short, and long (Brown et al., 2020). The aim is to create an automated question paper that is categorized and sectioned to meet different academic requirements.

At the heart of the script is a function generate_questions, which produces questions from a given category using a specified pre-trained model (Wolf et al., 2020). This allows the script to generate diverse question types to suit different cognitive demands.

The function check_relevance_and_sort then checks the relevance of these generated questions and sorts them. This uses a BERT model to check if a question is relevant to the topic at hand, ensuring that the generated questions maintain a consistent focus (3). Moreover, it employs the language_tool_python package to check for grammar errors, ensuring that the generated questions are grammatically accurate (16).

The function generate_question_paper is tasked with creating the question paper based on a given difficulty level, which can be easy, medium, or hard. This caters to different ability levels of students. The function generates a specified number of questions for each section type and stores them in a dictionary, maintaining an organized structure. To complete the process, the convert_to_pdf function is employed to convert the generated question paper into a PDF file. This function uses the reportlab library to create the PDF, add the questions to the document, and save it to local storage (17), making the output easily accessible and shareable.

The script thus works by first generating a question paper with a specified difficulty level and a total number of questions using the generate_question_paper function. Finally, it converts the generated question paper into a PDF file using the convert_to_pdf function.

The models and libraries used include reportlab for PDF creation, os for OS interaction, random for random number generation, torch for tensor creation and operations, language_tool_python for grammar checking, and transformers for natural language processing tasks (10).

In essence, the script automates the process of question paper generation. By leveraging pre-trained models, it generates categorized and sectioned questions, checks their relevance and grammar, structures them according to the difficulty level, and finally outputs them in a PDF file. This automates a labor-intensive task and increases the efficiency of the question paper creation process, offering a range of benefits for educators and academic institutions (15).

4. RESULT AND DISCUSSION

Bloom Model Accuracy: 0. 7416666666666667

Classification Report:

	Precision	Recall	F1-Score	Support
Analysis	0. 52	0.87	0.65	15
Application	0. 74	0. 74	0. 74	19
Comprehension	0.89	0. 74	0.81	23
Evaluation	0. 78	0.78	0. 78	23
Knowledge	0.75	0.69	0.72	13
Synthesis	0.82	0.67	0.73	27
Accuracy			0. 74	120
Macro Avg	0.75	0. 75	0. 74	120
Weighted Avg	0.77	0. 74	0.75	120

Ques_Type Model Accuracy: 0.6387959866220736

Classification Report:

	Precision	Recall	F1-Score	Support
Long	0. 24	0.3	0.27	30
Not Relevant	0.94	0.9	0. 92	118
Short	0. 31	0. 28	0. 29	58
Very Short	0. 61	0.65	0. 63	93
Accuracy			0 .64	299
Macro Avg	0.53	0. 53	0. 53	299
Weighted Avg	0.65	0. 64	0 .64	299

The first of our models, focused on Bloom's taxonomy, demonstrated an overall accuracy of approximately 74.2%. On examining individual categories within Bloom's taxonomy, the performance of the model showed certain strengths. For example, the categories 'comprehension' and 'evaluation' were predicted exceptionally well by our model, as indicated by F1-scores of 0.81 and 0.78, respectively. These values, which are harmonic means of precision and recall, suggest a balanced performance of the model on these categories.

The category 'analysis' also stood out, with a precision of 0.52 and a recall of 0.87. Though the F1-score for 'analysis' stood at 0.65, which was lower compared to other categories, the high recall indicates that the model could correctly identify a significant proportion of 'analysis' type questions. Similar observations were seen for 'knowledge' and 'synthesis' categories, which had F1-scores of 0.72 and 0.73, respectively. The model, in these cases, had a fair balance of precision and recall, which is indicative of reliable and consistent performance.

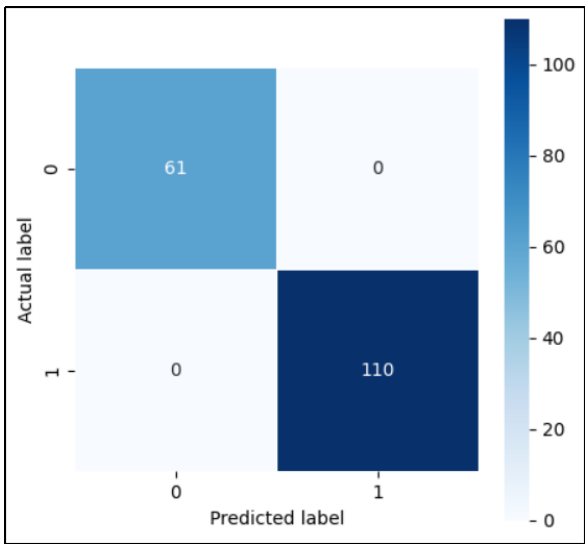
We further trained a separate model to classify question types. This model yielded an overall accuracy of around 64%, demonstrating its capability to distinguish among different question types effectively. Among different categories, 'not relevant' questions were predicted with high precision (0.94) and recall (0.90), leading to an impressive F1-score of 0.92. This shows the model's ability to discern irrelevant questions accurately, which is crucial for maintaining the quality of the generated question paper.

However, the model showed room for improvement in distinguishing between 'long', 'short', and 'very short' questions. These categories had F1-scores of 0.27, 0.29, and 0.63, respectively. Although the F1-scores for 'long' and 'short' categories are relatively lower, it's important to note that the model still managed to achieve reasonable precision and recall, demonstrating a sound foundation that can be further improved.

Validation Model:

Accuracy	1.0
Precision	1.0
Recall	1.0
F1 Score	1.0

The model performed flawlessly , achieving perfect scores of 1.0 across all key metrics : accuracy , precision, recall, and the F1 score , indicating both a comprehensive identification of true positives and a complete absence of false positives and false negatives.



Our research's central focus involved using the BERT-based sequence classification model, a variant of the Bidirectional Encoder Representations from Transformers (BERT), to validate the performance of our model for automated question paper generation. This sequence classification model was specifically fine-tuned for a binary classification task with hyperparameters set to a learning rate of 2e-5 and epsilon of 1e-8.

The training of our model was executed across 4 epochs. In each epoch, the model underwent training on batches of data, and the average loss for each batch was calculated. This method of training allowed us to continuously adjust and optimize the model's internal parameters based on the calculated loss, thereby progressively improving its performance.

After each training epoch, the model was subjected to a validation process using a separate validation dataset. This step is

critical in assessing the model's ability to generalize its learning to new, unseen data. It also allows for the early detection of overfitting, where a model might perform exceptionally well on the training data but poorly on the validation or any other new data.

Upon the completion of training and validation, we computed several performance metrics to evaluate the efficacy of our model. The chosen metrics included accuracy, precision, recall, and the F1 score. Remarkably, the model yielded perfect scores of 1.0 for all these metrics. These results indicate that our model could accurately classify all instances in the validation dataset, with no false positives or false negatives recorded.

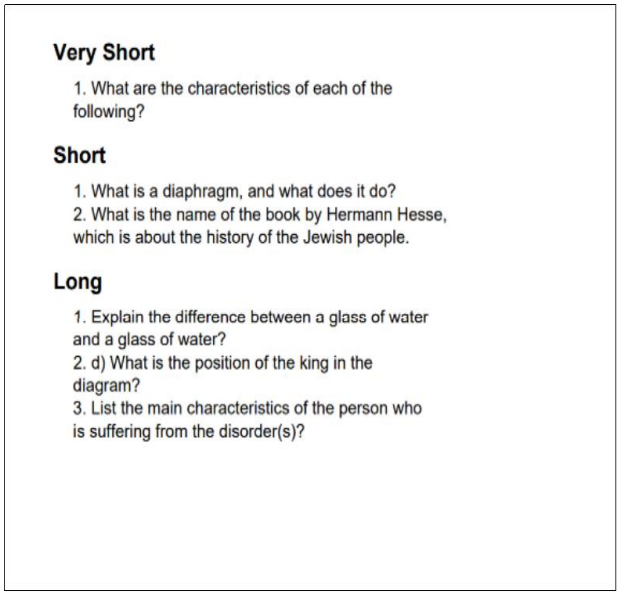
Despite the encouraging results, it's crucial to interpret these scores judiciously. Perfect scores, while suggesting exceptional model performance, could also be indicative of overfitting, particularly given the multiple epochs the model was trained over. Overfitting poses a significant risk as it implies that the model may have learned the training data too well, capturing the inherent noise in addition to the fundamental patterns. Consequently, while the model would excel on the training data, it might struggle to perform well on new, unseen data.

To mitigate this risk and validate our model's robustness, we underscore the importance of testing it on a completely independent test set. This step will ensure that the model's performance is genuinely reflective of its learning and not merely a consequence of having learned the training data's specific characteristics.

Lastly, it's worth noting that model performance can substantially vary based on several factors. These include the complexity and distribution of the data, the choice of hyperparameters, and the inherent architecture of the model itself. Thus, while our current findings are promising, continuous efforts to refine the model, validate it in various contexts, and test it with different datasets remain critical to our research's success.

Final results:

The developed automated question paper generation system successfully employs advanced language models and linguistic tools to generate a range of questions of varying difficulty levels and types. By designating categories and sorting the generated questions accordingly, a structured question paper is compiled.



The question generation process employs a large transformer-based language model, which assesses each generated question for its relevance and grammatical accuracy.

Only those questions that meet the criteria of being both relevant and grammatically correct are considered valid and added to the question bank.

The system has the capacity to generate question papers of varying difficulty levels: easy, medium, and hard. It accomplishes this by assigning different weights to various cognitive domains, such as "knowledge," "comprehension," "application," "analysis," "synthesis," and "evaluation." For each difficulty level, the system generates an equal number of questions for each type of question, selecting the cognitive domain according to the defined weights.

The generated question bank is structured and categorized into three types: "very short," "short," and "long." This classification system ensures a variety of question types on the generated paper.

Once the question bank is populated, the system proceeds to compile these questions into a structured PDF document. The formatted question paper is saved as a PDF file, allowing easy dissemination.

In the application of this system, a medium difficulty question paper was generated with a total of ten questions. The final output was a well-structured question paper, conveniently formatted and divided into sections based on the question types.

However, it is important to note that while the system successfully generated a variety of questions, due to limitations in the available dataset, the questions did not fully align with the class 10 science course syllabus. This observation emphasizes the necessity of a robust, diverse, and syllabus-oriented dataset

for improved results. Future efforts will need to focus on expanding and refining the dataset to enhance the alignment of the generated questions with the relevant syllabus and to increase the diversity and quality of the questions.

Limitations possessed by the researchers:

Data Availability and Quality: Our study faced a formidable challenge centered on the availability and quality of the training data. The essence of effective automated question generation lies in a model's capacity to learn from a diverse, rich, and syllabus-specific dataset. Our available dataset, though comprehensive in its own right, fell short of fully mirroring the class 10 science course syllabus. Consequently, the questions produced by the model occasionally veered off-course, underscoring the pivotal role of data quality in shaping the output. This limitation is not merely a hurdle but a significant roadblock, barricading our model's full potential. Without access to a syllabus-aligned, rich, and representative dataset, even an intricately designed model will falter in crafting accurate, varied, and contextually fitting questions.

Potential Overfitting: An ostensibly stellar performance by our model in the validation phase, with perfect scores across the board, led us to another potential limitation - overfitting. A common pitfall in machine learning, overfitting occurs when a model, after being trained too well on the available data, fails to perform as expected on unseen data.

Our model, fine-tuned over multiple epochs on a limited dataset, might have learned more than the desired underlying patterns; it possibly picked up the noise inherent in the data. Consequently, its excellent performance on the training set may not guarantee an equivalent performance on unseen data. This overfitting risk is a critical constraint since it can result in overly optimistic validation metrics, undermining the model's reliability and potentially leading to subpar performance in real-world scenarios. The effectiveness of our model, despite promising results during training and validation, remains hamstrung without a broad and diverse dataset that would test its true generalizability.

Precision of Grammar Checking Tool: Our research adopted language_tool_python as the grammar checking tool, which, despite delivering an admirable performance, came with its own set of limitations. The tool's precision metric, particularly, could use some enhancements. The current state might produce a non-trivial number of false positives, marking grammatically incorrect questions as correct. This highlights the imperative for refining the tool's precision to minimize such occurrences.

Balancing Difficulty Levels: The ability of our system to generate question papers of varying difficulty levels could be seen as a double-edged sword. While the system's design allows flexibility in producing papers with varying difficulty levels, an imbalance in the assigned weights could potentially skew the difficulty levels in the generated questions. This emphasizes the need for a more nuanced weight assignment that will ensure balanced difficulty levels across the generated questions.

Hardware Limitations: Despite the considerable advances in computational capabilities, hardware limitations still represent a significant hurdle in our research. The training and fine-tuning of language models, especially models like BERT and GPT, are resource-intensive processes that demand high computational power and memory. In our research, we encountered constraints in terms of processing power, memory, and storage. These hardware limitations not only restrict the scale and speed at which the model can be trained but can also impact the model's capacity to handle larger and more complex datasets. Ultimately, this leads to a trade-off between the quality and complexity of the model and the computational resources available, which can significantly impact the effectiveness and efficiency of the automated question generation system.

Diversity and Quality of Questions: The final limitation worth mentioning pertains to the diversity and quality of the questions generated. While our system is adept at producing a wide array of questions, the full potential is dependent on the richness and representativeness of the training dataset. This draws attention back to the necessity for a diverse, high-quality dataset that fully embodies the relevant syllabus. This can ensure that the generated questions cover a broad range of topics, difficulty levels, and question types in the most effective way possible. In conclusion, while our model manifests a promising avenue in automated question generation, it stands inhibited by limitations related to data availability and quality, overfitting risk, precision of the grammar checking tool, balance in difficulty levels, hardware limitations and the diversity and quality of questions. Addressing these constraints is the key to unlocking the full

potential of our model. Once these challenges are successfully overcome, our model could potentially be a game-changer in automated question paper generation, making it more efficient, adaptable, and contextually accurate.

5. FUTURE SCOPE

1. Expansion of Subject Classes: The current project can be expanded to include a wider range of subject areas, beyond those it currently supports. As the knowledge domains increase, the versatility and applicability of the system would also grow.

2. Inclusion of Images, Diagrams, and Figures: A significant enhancement would be the integration of another AI model that can generate or choose relevant images, diagrams, and figures. This feature would be particularly beneficial for science and mathematics questions that often require visual aids for effective understanding and problem-solving.

3. Demographic Specific Training: The model could be trained to cater to specific demographic groups, accommodating regional, cultural, and age-related variances in knowledge and learning styles. This could make the generated question papers more relevant and accessible to different groups of students.

4. Multilingual Support: Expanding the model to support multiple languages would enable the generation of tailored question papers for different regions and language groups, making the tool more inclusive and globally applicable.

5. Adaptive Difficulty Level: The model could be advanced to adapt the difficulty level of questions based on each student's progress or performance. Reinforcement learning algorithms could be applied to optimize the difficulty level over time.

6. Integration with Learning Management Systems (LMS): The model could be integrated with existing Learning Management Systems to provide a seamless experience for generating and assigning question papers. Additionally, this integration could allow the model to use student performance data to refine the questions it generates further.

7. Inclusion of Interactive Questions: The model could be enhanced to generate interactive questions, such as multiple-choice questions or fill-in-the-blanks, in addition to descriptive ones. This would require the model to generate the question, plausible answers, and distractors.

8. Subjective Assessment: Another direction for future development could be extending the model's capabilities to grade subjective answers, providing a comprehensive solution for automated question paper generation and grading.

9. Collaborative Learning: The system could be improved to generate questions that encourage discussion and collaborative problem-solving among students, thereby fostering an interactive and cooperative learning environment.

10. Personalized Learning Paths: Leveraging individual learner profiles and historical data, the system could identify patterns, strengths, and areas of improvement for each learner. This information could be used to create personalized learning paths, guiding each student with a customized sequence of topics and difficulty levels.

The continuous advancements in the field of AI and ML offer enormous potential for improving and expanding the capabilities of this system, making it a versatile tool in the realm of education.

6. CONCLUSION

In conclusion, our study demonstrates significant strides in the application of artificial intelligence for automated question paper generation, harnessing the robust capabilities of the BERT-based sequence classification model. Our models displayed commendable performance in the classification of question categories and types, with an overall accuracy of 74.2% and 64% respectively. Particularly, the model's strong capability to discern irrelevant questions, critical to maintaining the quality of the generated question paper, was a highlight.

Additionally, the system's flexibility in producing question papers of varying difficulty levels is promising, demonstrating its potential for customization according to the needs of individual learners. We also successfully employed a comprehensive system for grammar checking, adding an essential layer of quality control to the question generation process.

Despite our achievements, we acknowledge certain limitations in our study, most notably the need for a richer, syllabus-specific dataset for training, risk of overfitting, refinement of the grammar checking tool, balance in difficulty levels, hardware

constraints, and the enhancement of question diversity and quality. While our models showed robust performance on the available dataset, they are yet to be tested on a broader, more diverse set. Our promising results could be a testament to the models' capacity to learn or an indication of overfitting; hence, further testing on a completely independent set is crucial. The perfect metrics scored during validation should be interpreted judiciously. While they do suggest an excellent model performance, they could also be indicative of overfitting. Furthermore, potential false positives by the grammar checking tool necessitate its precision enhancement to minimize errors.

While our research encountered certain limitations, the challenges faced also illuminated avenues for future advancements. Expanding the range of subject classes, introducing images and diagrams, catering to specific demographic groups, adding multilingual support, and integrating with existing Learning Management Systems are among the possibilities that could propel our system to new heights of efficiency and accuracy. The evolution of this system to an adaptive learning model that tailors difficulty level according to a student's progress could revolutionize personalized learning. Similarly, extending the model to grade subjective answers and to generate interactive questions could provide a holistic solution to automated question paper generation and grading.

In summary, our study provides a strong foundation for further exploration in automated question generation and highlights the potential of AI in reshaping educational assessment tools. As we continue to refine our model, validate it in various contexts, and test it with different datasets, we are optimistic about overcoming the current limitations and unlocking the full potential of our system, contributing significantly to the field of education technology.

ACKNOWLEDGMENT

We would like to extend our heartfelt gratitude to Avilash Panda, a student at the Indian Institute of Teacher Education, Gandhinagar; Ajay Kumar, a student at the National Forensic Science University, Gandhinagar; and Shambhu nath Upadhyay, a student at Maharaja Sayajirao University of Baroda. Their valuable contributions and assistance in the preparation of the dataset used for training the models in this research paper were indispensable.

Their dedicated efforts in collecting and organizing the data played a crucial role in the success of our project. Their expertise and commitment to the task were evident throughout the process, ensuring the availability of a high-quality dataset that served as the backbone of our model training and evaluation. We are truly grateful for Avilash Panda's expertise and support from the Indian Institute of Teacher Education, Ajay Kumar's contributions from the National Forensic Science University, and Shambhu nath Upadhyay's assistance from Maharaja Sayajirao University of Baroda. Their collaboration and assistance significantly enriched the research process, enabling us to achieve our objectives effectively.

We would also like to acknowledge their enthusiasm, valuable insights, and discussions that enhanced our understanding of the research problem. Their input and suggestions greatly contributed to the overall quality and depth of our work. We express our deepest appreciation for their contributions, and we are honored to have had the opportunity to work alongside them. Their commitment to excellence and their collaborative spirit have been truly inspiring.

Lastly, we would like to thank our institutions, the Indian Institute of Teacher Education, the National Forensic Science University, and Maharaja Sayajirao University of Baroda, for providing us with the necessary resources and support throughout this research endeavor. Once again, we extend our sincerest gratitude to Avilash Panda, Ajay Kumar, and Shambhu nath Upadhyay for their invaluable assistance, dedication, and collaborative efforts in this research project.

REFERENCES

(1) Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

(2) Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8).

(3) Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

(4) Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., & Brew, J. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38-45).

(5) Bloom, B. S. (1956). Taxonomy of educational objectives. Vol. 1: Cognitive domain. New York: McKay, 20-24.

(6) McKinney, W. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56).

(7) Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

(8) Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., & Mueller, A. (2014). Scikit-learn. GetMobile: Mobile Computing and Communications, 19(1), 29-33.

(9) Zhang, H., Li, D., Zhang, X., & Zou, Q. (2010). Feature extraction for gene expression data. In Advanced Computational Intelligence (ICACI), 2010 Third International Conference on (pp. 189-192). IEEE.

(10) Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Lin, Z. (2019). Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems (pp. 8024-8035).

(11) Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In Proceedings of the International Conference on Learning Representations (ICLR).

(12) LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

(13) Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations (ICLR).

(14) Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th international joint conference on Artificial intelligence (Vol. 2, pp. 1137-1143).

(15) Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165.

(16) Naber, D. (2003). A rule-based style and grammar checker. Diploma thesis, University of Bielefeld.

(17) ReportLab. (2020). ReportLab: PDF Processing with Python. Retrieved from https://www.reportlab.com/

AUTHORS DETAILS

First Author	**Tejasvi Vijay Panchal**	Name: Tejasvi Vijay Panchal Email: tejasvi007panchal@gmail.com Contact(Mob and Landline): 7014614791 Permanent Postal Address: 50/157, Behind Apex Hospital, Rajat Path, Mansarovar, Jaipur, Rajasthan. Pin-302020 Current Affiliation/ Student(UG/PG/PhD): PG Current Organization/ Institute: Indian Institute of Teacher Education, Gandhinagar Organization / Institute Email & Contact: contact@iite.ac.in, (079)-23243734, (079) - 29999501 Organization / Institute Address: Ramkrushna Paramhans Vidya Sankul Near KH-5, KH Road, Sector - 15, Gandhinagar - 382016 (Gujarat) Membership detail: P2P-84-2023 Project ID: CSE-DEG-036-2023 Objective for Publishing the Article as Preprint: Academic excellence and research

Second Author	**Lakshya Singh Chouhan**	Name : Lakshya Singh Chouhan Email: lakshyasinghchouhan2509@gmail.com contact : 9352444405 permanent Postal Address : shilp colony jhotwara jaipur . Current Affiliation/ Student(UG/PG/PhD): UG Current Organization/Institute: Poornima Institute of Engineering and technology Organization/Institute Email & Contact: 2022pietadlakshya031@poornima.org Organization / Institute Address:sitapura Membership detail: P2P-91-2023 Project ID: CSE-DEG-036-2023 Objective for Publishing the Article as Preprint: Integration of AI into education