A Strategy for Generating Synthetic Educational Datasets using Large Language Models

Aarathi Vijayachandran, Sumat Singh Bedi, Chukwuka Victor Obionwu
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg
Magdeburg, Germany
{aarathi.vijayachandran,sumat.bedi,obionwu}@ovgu.de

ABSTRACT

Recent research endeavors suggest that academic failure can be readily remedied through the deployment of automated learning interventions, or artificial intelligent agents, that can assess and recommend learning improvements that can structure a student's learning engagement behavior at an early stage. These interventions can provide personalized feedback and adaptive learning suggestions, catering to the unique needs of each student. Furthermore, by analyzing the data collected from these interventions, researchers can gain insights into the effectiveness of different teaching strategies and make informed decisions to enhance educational outcomes for diverse individuals or communities. However, the availability of large datasets is crucial for the development of these personalized interventions, as for accurate prediction and the provision of an optimal recommendation, these intelligent agents require large training datasets. Further, there is currently no benchmark for a tabular synthetic data generation process for students structured query language study engagement data. Thus, in this study, we propose and devise an expert in the loop pipeline for generating structured query datasets for the training and evaluation of these intelligent agents that will be tasked with mediating students structured query language (SQL)-based study engagements and related activities. To evaluate the effectiveness of our strategy, we have categorized the generated datasets into three different language areas: the data definition language, the data manipulation language, and the data query language. With respect to these language areas, we measured the accuracy of the generated datasets in terms of their similarity with the original datasets and their ability to accurately create an equivalent schema as the original query string. The performance of our prototype reaches a cosine similarity value of 0.767, which justifies our pipeline as a potential strategy for generating complex synthetic datasets.

1 INTRODUCTION

Traditionally, educational strategies and research have been based on the development of interventions that are focused on the student, the instructor, and the knowledge artifact. As a result, a large number of studies and theories now exist for virtually any form of educational practice. However, with the advancement of technology and the increasing accessibility of online learning platforms, there is a growing need to explore new approaches that incorporate digital tools and resources into educational strategies. This shift in focus allows for a more personalized and interactive learning experience, catering to the individual needs and preferences of students while also fostering collaboration and engagement among peers. One such tool that has lately taken the educational sector by storm is the generative pretrained transformer. This tool has proven to be highly effective in augmenting learning as a result of the recent and ongoing progress in the field of generative artificial intelligence, which has led to the creation of models such as GPT 4, LLaMA-2, Gemini-Pro, and PaLM-2. These models have consistently outperformed the state of the art in a variety of natural language processing (NLP) tasks, which has created new opportunities for carrying out previously performed tasks by humans. The the development of interventions that are focused on the student, the instructor, and the knowledge artifact (AIA), is now capable of generating high-quality text, understanding and responding to natural language queries, and even carrying out complex language-based tasks such as summarization, translation, and sentiment analysis. With the generative pretrained transformer's ability to learn from vast amounts of data, it has become an invaluable tool in the educational sector, offering personalized learning experiences and enhancing student engagement. Additionally, its potential for automating administrative tasks and providing real-time feedback has made it a game-changer in revolutionizing education. This makes a case for a shift from the development of interventions that are focused on the student, the instructor, and the knowledge artifact to the development of interventions that are focused on the student, an artificial intelligent agent, and the knowledge artifact with the instructor's oversight. However, the unavailability of large educational datasets that depict student learning engagement poses a challenge to fully harnessing the potential of artificial intelligent agents in education. Without access to comprehensive and diverse data, it becomes difficult to train these agents effectively and ensure their accuracy in understanding and responding to student needs. Also, state-of-the-art studies show that GPT-3.5, when used directly for tasks such as named entity recognition and relation extraction tasks, yields poor performance compared to state-of-the-art models trained on the dataset, as indicated in [4, 11]. This further reinforces that while GPT-3.5 displays impressive inference and reasoning abilities in various classic natural language understanding tasks, without domain- or task-specific training, it will perform poorly. Furthermore, we are developing a learning intervention, and there are several government policies regarding the use and sharing of student data. To address these challenges, we have developed an expert in the loop pipeline for generating structured query datasets for the training and evaluation of these intelligent agents. In this work, we will attempt to address the following research questions:

- What strategies exist to assure that synthetic data, while functioning as a digital twin of the original data, maintains any direct connection to the original data?
- What methodologies are available for producing synthetic time-series datasets of superior quality while preserving the privacy of the training dataset?

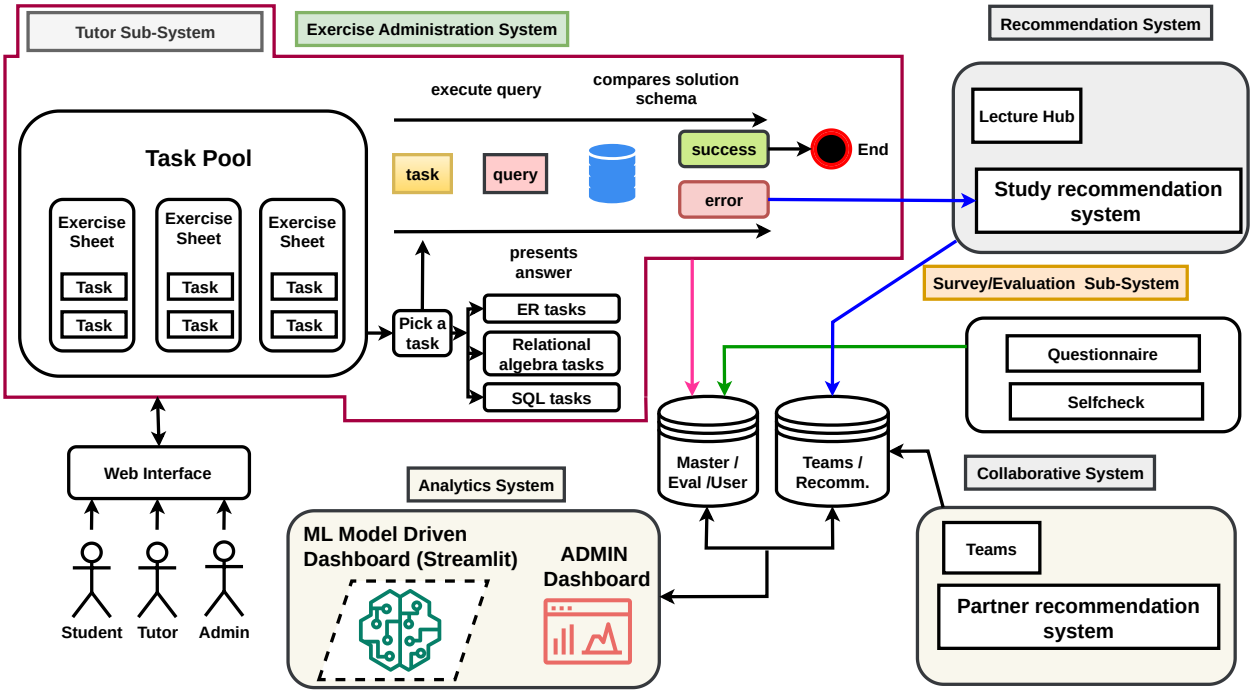

Figure 1: System Schematic

The rest of the paper is structured as follows: The next section introduces our learning management system and a description of the activities and respective datasets we seek to create. In Section III, we describe our pipeline schematic and an overview of the functionalities, and in Section V, we give in-depth details of the pipeline, its description, and the important algorithms we employ. In Section V, we evaluate and discuss the performance of our system. In Section VI, we discuss related work and show limitations. We summarize our contributions in Section VIII and indicate directions for future efforts.

2 THE SQLVALIDATOR

Several learning management systems have been developed for various educational purposes. These systems aim to provide a centralized platform for educators and students to interact, collaborate, and manage academic content. Some popular learning management systems include Moodle, Canvas, and Blackboard, each offering unique features and functionalities tailored to the needs of different institutions and users. These platforms typically offer tools for online course creation, assignment submission, grading, discussion forums, and multimedia content integration. With the increasing demand for remote learning and e-learning solutions, learning management systems have become essential tools in modern education.

In the SQLValidator architecture shown in fig 1 users access, and interact with the platform via a web interface. The tasks and submissions components contain all information necessary for its processing. A PHP server mediates between user requests, the databases and the relational database management system[9].

Current use cases for SQLValidator fall into the following categories:

- **Course Exercises:** The database lecture in Magdeburg includes multiple exercise assignments. SQLValidator is utilized to manage these activities. Students can continue revising their queries until they get the desired outcome. Throughout the testing and submission process, they receive the anticipated schema and helpful feedback on their queries. This hands-on approach allows students to actively engage with the course material and improve their SQL skills in a practical setting. Additionally, the use of SQLValidator streamlines the grading process for instructors, providing efficient and effective feedback to students.
- **Self Checks:** These tests are crafted to help students assess their level of understanding of the already covered topics in the database concept course. Students can repeatedly take the self-check test until they are satisfied. The questions used in these self-checks are mainly standardized and multiple choice. Additionally, we incorporate tests for SQL query skills relevant for the learned course skills.
- **Questionnaires:** At specific times during the semester, students are prompted to assess either their learning experience in the course or their user experience with certain tools used in conducting course activities, such as the SQLValidator system. We utilize our SQLValidator directly to do these assessments , which allows students to provide feedback on the effectiveness of the tool in helping them learn and apply SQL concepts. This feedback is then used to make improvements and adjustments to enhance the

overall learning experience for students. Additionally, instructors may also use the feedback to tailor their teaching methods and materials to better meet the needs of the students. This continuous feedback loop helps to create a more dynamic and engaging learning environment.

```
CREATE TABLE MADE_OF (
amount decimal,
wname varchar(20),
gname varchar(20),
foreign key(wname)references WINE(name),
foreign key(gname)references GRAPE(name),
primary key(wname))
```

Listing 1: Sample Student Query with Error

The SQLValidator learning management system facilitates a hybrid learning environment. Table 1 shows the different types of error that students may encounter while engaging with the exercise tasks. Also, it requires an instructor's oversight during students exercise engagement, which is not always feasible. Also, efforts have been made towards devising an automatic feedback strategy in the form of lecture slide recommendations. While this is a step in the right direction, it is not humanlike, as the recommendations are page and section numbers. An instructor explains and clarifies doubts before pointing the student to reference material. However, an agent can replace the instructor, and the instructor will supervise and fine-tune the agent to behave more like a human instructor. This way, the system can provide personalized and interactive support to students, addressing their doubts and guiding them through the learning process. Additionally, the use of natural language processing techniques can enhance the agent's ability to understand and respond to students' queries in a more human like manner. More importantly, the agent will always be available, even in remote locations without internet facilities. This is especially beneficial for students in underserved areas. To realize this system, training datasets consisting of various examples of student queries and corresponding responses are necessary. The dataset should be diverse and representative of different learning scenarios to ensure the agent's effectiveness across a wide range of topics and situations. Also, a pipeline should be established to regularly update and maintain the dataset. This will ensure that the agent remains up-to-date and can continue to provide accurate and relevant responses. The objective of this endeavor is the development of a syntetic data generation process for student structured query learning engagement. This will be described in the next section.

3 IMPLEMENTATION

The schematic diagram for the implementation of this expert in the loop pipeline model is given in Fig. 2, 3 and 4. There are 4 main blocks in the implementation process, namely Data Collection, Data Preprocessing, Data Generation and Evaluation.

Table 1: Error Codes Table

Error Class	Error Code	Error Name	Description
Syntax Error	0	Syntax Error	Missing semi-column at the end of the statment
Table	2	Column Counts	Missing to intitialize one or more columns
	3	Column Order	Starting a table with "Name" column and then the "ID" column
	4	Column Name	Name a column as "Name" instead of "First Name"
	6	Row Count	Giving a higher number of rows than the number in the table
	20	Table Content	There is a "plz" column in the student's courses table
	21	Table RowOrder	The Rows show up not in the order of "ID" column
Constraint	7	Constraint Count	Missing one or more constraints
	8	Primary Key	Missing to intitialize PK
	9	Unique	Missing to intitialize unique column
	10	Foreign Key	Missing to intitialize FK
	11	General Keys	Missing to intitialize unique key
Foreign Keys	17	FK Name	Writing the FK-Name wrongly
	18	FKRef Table	Writing the FK-Name of reference table wrongly
	19	FKRef Column	Writing the FK-Name of reference column wrongly
Schema	5	Column Count CT	Entering 4 values to a table that contains 3 columns
	12	Data Type	Intialize "name" column with integer value
	13	IsNull	Missing to intialize that a value can be null
	14	IS Default	Missing to intialize that a value is deafult value for a column
	15	Column Name	Intialize "name" column instead of "firstname"
	16	Table Name	Name a student table instead of Employee

3.1 Data Collection

In the initial phase of the implementation, we started the model training by collecting the fundamental dataset from the SQLValidator tool. This is a dataset of the students' responses to a number of SQL exercises given in varying semesters, that was exported to Comma-Separated Values (CSV) format to enable further processing and analysis. The whole programming process was done using the Python programming language with Google Colab as the main platform.

The unprocessed CSV was then converted into a dataset that was more appropriate for analysis, by organizing it into a structured pandas dataframe with vital columns such as Name, Course, Student ID, User ID, Task ID, Error Class, Student Questions, etc.

Once the initial organization of data was complete, a series of data cleaning operations were performed to get rid of the repeating

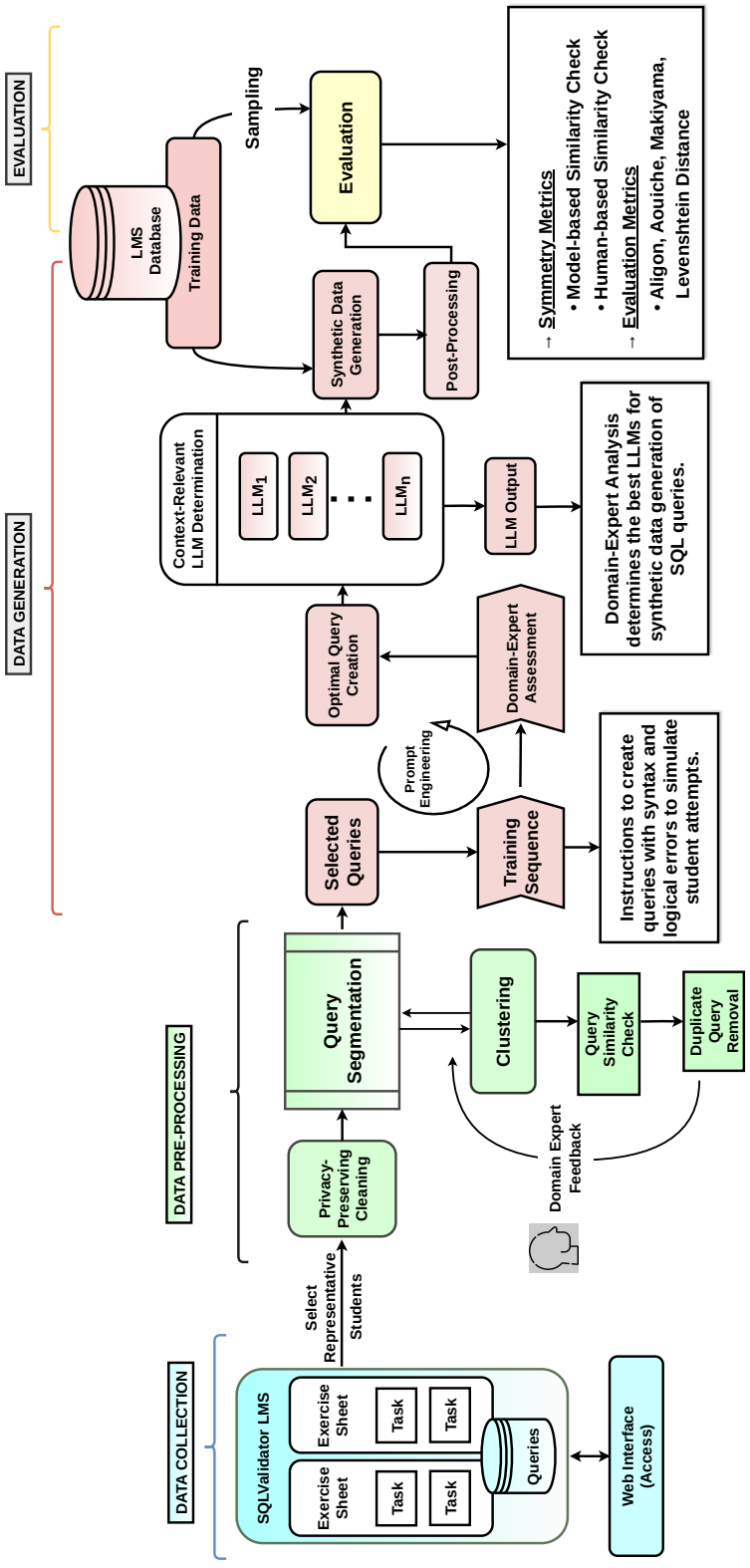

Figure 2: Synthetic Data Generation Pipeline

rows and remove insignificant columns. This pre-processing made the dataset better stratified, thus improving its quality for further analysis processes.

To ensure the representativeness of our dataset, we implemented a selection criterion that focused on students who demonstrated a considerable engagement with the SQL tasks, as evidenced by their completion of at least 60% of the total exercise tasks available. This selection criterion was instrumental in identifying a subset of representative students, whose data provided meaningful insights into the learning patterns and challenges encountered in SQL exercises.

The refined dataset comprising these representative students was then integrated into the Data Preprocessing block of model.

3.2 Data Preprocessing

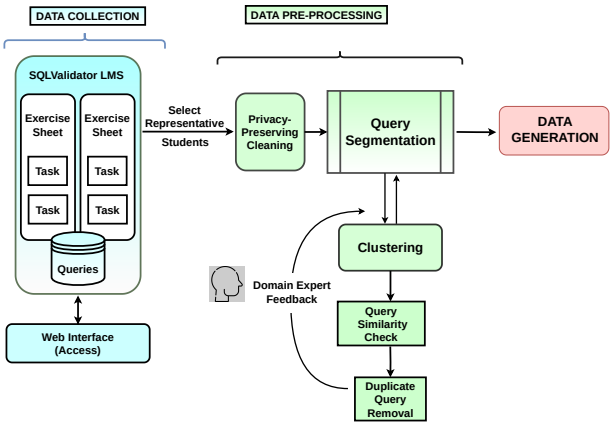

Figure 3: Data Collection and Preprocessing

The data preprocessing block begins with the implementation of a privacy-preserving data cleaning process. This process is the systematic and automated removal of students' personally identifiable information such as the names, student IDs, and course details. It is critical that the performance of the model is as valid as possible while ensuring participant anonymity. Thus, we retain only the unique User ID to represent each of the participating students.

Next process is Query Segmentation, that intends to eliminate student attempt queries that are too similar, as they can be considered as duplicates. This was achieved through a methodical clustering of the dataset based on distinct parameters: User ID, Task ID, and Error Class. From each cluster, the cosine similarity metric was used to determine which attempt is highly similar to the others. Only one query from a cluster of similar queries was retained, and the remaining were discarded. Furthermore, a rigorous manual evaluation was carried out to make sure that the retained queries are proper and relevant.

This preprocessed dataset is then passed onto the Data Generation block.

3.3 LLM Determination, Data Generation and Post-Processing

In the Data Generation block, the first task is to implement prompt engineering to create optimal prompts for each student's attempts for every exercise task.

```
def preprocessing(Q):
    # 1. Select students who attempted at least 60% of the questions Q
    filtered_students = []
    threshold = 0.6
    for student, attempts in Q.items():
        if sum(attempts.values()) / len(Q) >= threshold:
            filtered_students.append(student)

    # 2. Select the last 10 attempts of each student
    filtered_attempts = []
    for student in filtered_students:
        filtered_attempts.extend(student[-10:])

    # 3. Cluster the attempts of each student for each question
    cluster_dict = {}
    for attempt in filtered_attempts:
        student, question, attempt_text = attempt
        if student not in cluster_dict:
            cluster_dict[student] = {}
        if question not in cluster_dict[student]:
            cluster_dict[student][question] = []
        cluster_dict[student][question].append(attempt_text.split())

    #Remove duplicate attempts from the students based on a criteria
    for student in cluster_dict:
        for question in cluster_dict[student]:
            cluster_dict[student][question] = statremove_duplicate_attempts(cluster_dict[student][question])

    return filtered_attempts, cluster_dict

# Generate prompts using student attempts, correct query and human instructions
def prompt_generation(filtered_attempts, cluster_dict):
    prompts = []
    for student in filtered_attempts:
        for question in filtered_attempts[student]:
            student_attempts, correct_answer, instruction = generate_prompt(student, question, cluster_dict[student])
            prompts.append(student_attempts)

    return prompts
```

The average number of exercise task attempts for each student was calculated and stored, let's call this number as X for every student. To create an optimal prompt for each student's attempts

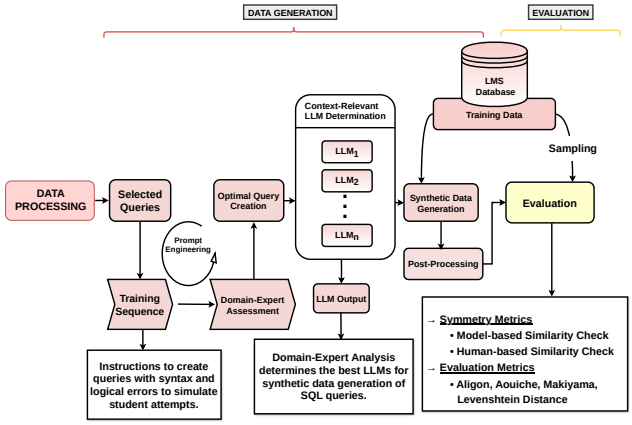

Figure 4: LLM Determination, Data Generation and Evaluation

for each exercise task, we took the SQL attempts of each student for each task and correct answers of each task and added instruction commands to generate X number of similar attempts with logical errors and X number of similar attempts with syntax errors. Manual evaluation was conducted on a sample of optimal prompts to check for correctness. Finally, all the optimal prompts for all student attempts for all exercise tasks were stored in a list. In the Context Relevant LLM Determination section of the Data Generation block, we took a sample from the list of optimal prompts and fed it to different LLMs like GPT-4, GPT-3.5-Turbo, GPT-2, Gemini-Pro, PaLM-2, LLaMA-2-70B, LLaMA-2-13B, LLaMA-2-7B, BERT-Base, T5-Base, etc. Then we manually evaluated the output from each LLM to identify the 'best LLM' for our expert in the loop pipeline model.

To this 'best LLM', we fed the entire list of optimal prompts and stored the corresponding output in a CSV, which is our required raw synthetic data of SQL student attempts.

To this raw synthetic data, we applied post processing to remove unwanted characters and explanations added by the LLM. Finally the dataset was converted into a structure suitable for passing onto the Evaluation block.

3.4 Evaluation

The evaluation of synthetic dataset was conducted through a detailed comparison with the original training dataset. We employed two types of evaluation metrics: Symmetry Metrics and SQL Query Metrics.

As part of the Symmetry Metrics, we performed a model-based similarity check with cosine similarity to quantify the degree of resemblance between synthetic and training data. Additionally, a thorough human-based similarity check was performed on samples of synthetic data, to identify trends, abnormalities, or discrepancies that automated metrics may not capture.

The SQL Query Metrics that we used were Aligon's similarity [1], Aouiche's similarity [2] and Makiyama's similarity [7]. These methods are essential for assessing the datasets' content- and structure-based similarity.

Through the various evaluation metrics, we assessed the consistency and dependability of the synthetic data in imitating the structural subtleties and content specificity of the original datasets.

4 RELATED ALGORITHM

The pseudocode for implementation process described above is given in two parts as Listing 1 and 2.

4.1 Pseudocode Part 1: Algorithm for Data Preprocessing, Prompt Generation

This algorithm operates in two primary stages: preprocessing and prompt generation.

```
Determine the best LLM for our specific usecase using various evaluation metrics
def context_relevant_LLM_determination(LLMs, prompts):
    best_LLM = None
    max_score = 0

    for LLM in LLMs:
        prompt_scores = []
        for prompt in prompts:
            generated_attempts = LLM.generate_attempts(prompt)
            evaluation_metrics = evaluate_attempts(generated_attempts, prompt['correct_answer'])
            score = sum(evaluation_metrics)
            prompt_scores.append(score)

        if max_score < max(prompt_scores):
            max_score = max(prompt_scores)
            best_LLM = LLM

    return best_LLM

#Generate sythetic data using the prompts generated for the best context-relevant LLM
def synthetic_data_generation(best_LLM, prompts):
    synthetic_data = []

    for prompt in prompts:
        generated_attempts = best_LLM.generate_attempts(prompt)
        synthetic_data.extend(generated_attempts)

    return synthetic_data

#Evaluate the generated synthetic data using domain-relevant evaluation metrics
def synthetic_data_evaluation(synthetic_data, original_data):
    evaluation_metrics = evaluate_data(synthetic_data, original_data)
    return evaluation_metrics
\end{minted}
```

Listing 1: Algorithm for LLM Determination, Data Generation and Evaluation

4.1.1 Data Preprocessing.

- Student Filtering: Initially, we filtered out students based on their engagement level, specifically selecting those who have attempted at least 60% of the total exercise questions.
- Attempt Selection: For each selected student, the algorithm then narrows down the data to their last 10 attempts. This approach aims to capture the most recent understanding and performance of students, potentially reflecting their current knowledge state more accurately.
- Clustering Attempts: Subsequently, it clusters the attempts of each student for each question.
- Duplicate Removal: For each of the clusters, the algorithm removes duplicate attempts based on a specific similarity criterion.

4.1.2 Prompt Generation. Utilizing the preprocessed data, the algorithm generates prompts for each question based on the students' attempts, the correct answers, and provided instructions to generate syntax and logical error attempts. The output of this stage is a set of optimal prompts.

4.2 Pseudocode Part 2: Algorithm for LLM Determination, Data Generation and Evaluation

4.2.1 Context-Relevant LLM Determination. This algorithm begins by evaluating a collection of LLMs to identify the one best suited for our expert in the loop pipeline model. This is achieved by providing a sample optimal prompt to each LLM and assessing the generated responses against predefined manual evaluation metrics. The LLM that yields the highest score is selected as the most appropriate for the model.

4.2.2 Synthetic Data Generation. Upon determining the most suitable LLM, the algorithm proceeds to generate synthetic data. It uses the selected LLM to respond to a set of optimal prompts, creating a dataset of generated synthetic student attempts.

4.2.3 Synthetic Data Evaluation. The final stage involves a thorough evaluation of the synthetic data against the original data using domain-relevant metrics. This step assesses the quality, relevance, and utility of the generated data in comparison to original data.

5 EVALUATION AND DISCUSSION

For evaluation, we used various similarity metrics commencing from cosine similarity. Once the data generation process is completed, the data is then pre-processed as described above. The initial step in post-processing after the synthetic data generation is cleaning of the generated data. In this imperative phase, we clean the data of any inconsistencies and remove any erroneously generated data by the LLM. For our synthetic data generation process, the data generated should eventually be transformed in the format of the original dataset. This is essential for the evaluation phase as this ensures data quality, relevance and comparability. After the cleaning phase is completed, to align the data format - we parse each SQL query generated by the LLM and assign various classifications and categories to the query. The query gets assigned an Error Class based on the error category generated by the LLM (0 for Syntax Errors and 2 for Logical Errors). Once the Error Class is assigned to the query, we forward each query to an SQL Command Classification function which assigns the relevant commands to the parsed query (also depicted in table2).

Command Category Classification		
DDL	**DML**	**DQL**
CREATE	INSERT	SELECT
DROP	UPDATE	WITH
ALTER	DELETE	
TRUNCATE		

Table 2: Command Category Classification

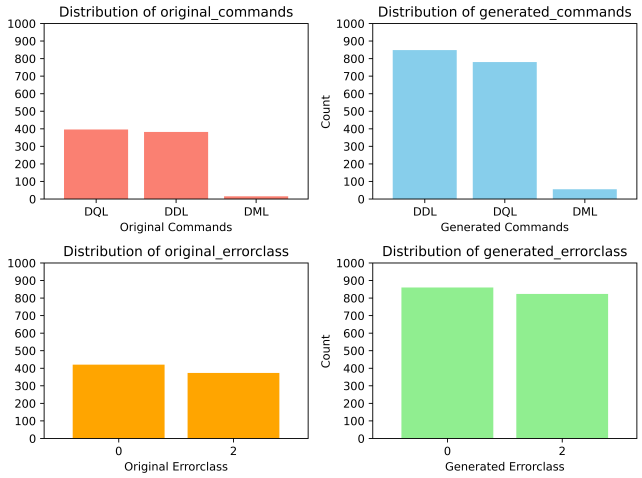

Figure 5: Original & Segregation Commands with Error Class

The function assigns the command category 'DDL' (Data Definition Language) to queries that start with 'CREATE', 'DROP', 'ALTER', 'TRUNCATE', the command category 'DQL' (Data Query Language) to queries that start with 'SELECT' or with, and finally the command category 'DML' (Data Manipulation Language) to queries that manipulate the data and start with 'INSERT', 'UPDATE' and 'DELETE'. The 'DCL' (Data Control Language) command category is unneeded as the SQLValidator does not have any specific tasks that test the users on DCL commands.

5.1 Similarity Evaluation

To perform the evaluation, the original SQL query attempts and synthetically generated SQL query attempts are grouped by UserID and TaskID, allowing for the comparison of corresponding queries. Following this, we iterate over groups of UserIDs and TaskIDs extracting the original and synthetically generated SQL query attempts for each group.

In each of these iterations between (UserID, TaskID) groups, we calculate for each original query and generated query their Cosine Similarity. Cosine Similarity for our SQL queries with be calculated by treating each SQL query as a vector in a high-dimensional space, where each dimension corresponds to a unique term or token present in the queries. Then, the cosine similarity can be computed

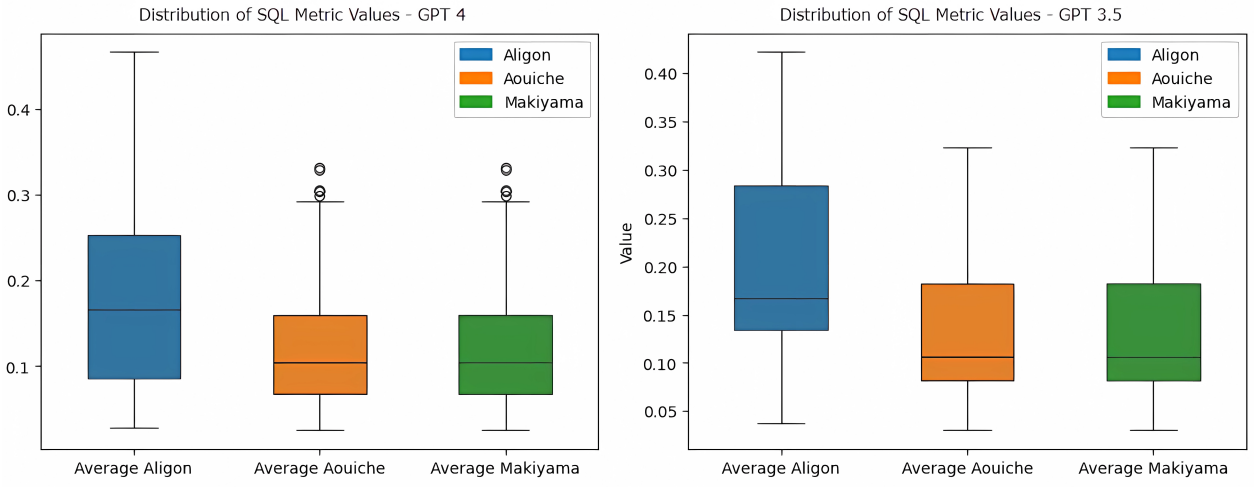

Figure 6: Distribution of similarity metrics for GPT-4 and GPT-3.5 Turbo Model

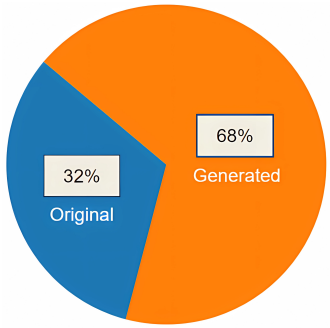

Figure 7: Proportion of Original and Synthetic Data

as the cosine of the angle between their respective vector representations. In further detail, we calculate the Cosine Similarity by first tokenizing the SQL queries into individual words, keywords, operators and identifiers by breaking the query down to its basic constituent elements. Next, we represent each query as a vector in a high-dimensional space and apply TF-IDF (Term Frequency-Inverse Document Frequency) Weighting to the vector representation of the SQL queries. Finally, we calculate the Cosine Similarity using the formula below and apply a threshold value to determine whether the queries are sufficiently similar.

Apart from the Cosine Similarity, we also evaluate specialized metrics used for the calculation of similarity in SQL queries. These metrics, specified by Kul et al.[6] are Aligon's similarity, Aouiche's similarity and Makiyama's similarity. Aligon et al.[1] considers syntactical query information to calculate query similarity focusing on various components of SQL queries, such as projection, group-by, and selection-join items. Additionally, the Aligon metric allows for the adjustment of weights assigned to these feature sets based on the specific requirements or characteristics of the SQL queries and the underlying database schema. Aouiche et al.[2] try to evaluate similarity by understanding the underlying data dependencies. By applying the Hamming Distance measure, they evaluate how similar two queries are without considering how often or where items appear in the query. Lastly, Makiyama et al.[7] evaluate query similarity by first extracting the terms in selection, joins, projection, from, group-by and order-by items separately and recording their appearance frequency. Then, a feature vector is created using the frequency of these terms which they use to calculate the pairwise similarity of queries with cosine similarity.

The magnitude of data expansion depends on various factors including the complexity of the original queries, diversity of error patterns as well as the LLM model's generative capabilities. It is critically important to strike the perfect balance between the quantity and quality of synthetically generated data. Merely increasing the synthetically generated dataset size will result in poor model performance and evaluation, therefore after extensive domain expert feedback analyses and discussion, the data expansion scale was kept at roughly twice the original data.

The figure 5 shows the overall structure of the dataset and aids in examining the original and synthetic data at a glance. Understanding the composition and characteristics of the data is imperative to gauge the efficacy of the evaluation. The figure shows bar charts for each data distribution, distribution of the original query commands is represented along with the distribution of the error classes evaluated for the original queries. The data has been clustered in the 3 Command Categories depending on the various queries that the users submit to the SQLValidator. Alongside this, the proportion of the original queries and synthetically generated queries in the dataset has also been shown.

The pie chart in figure 7 shows the proportion of original data to the synthetic data. As discussed before, the data expansion scale for our usecase and considering various factors has been kept at roughly 2x.

The scatter plot shown in figure 8 demonstrates one particular category of errors (in this case, Logical Errors) and the varying cosine similarity with each UserID and TaskID pair. On the Y-axis, various UserID_TaskID pairs are displayed and on the X-axis their cosine similarity in the range of 0 to 1 is displayed. The data points

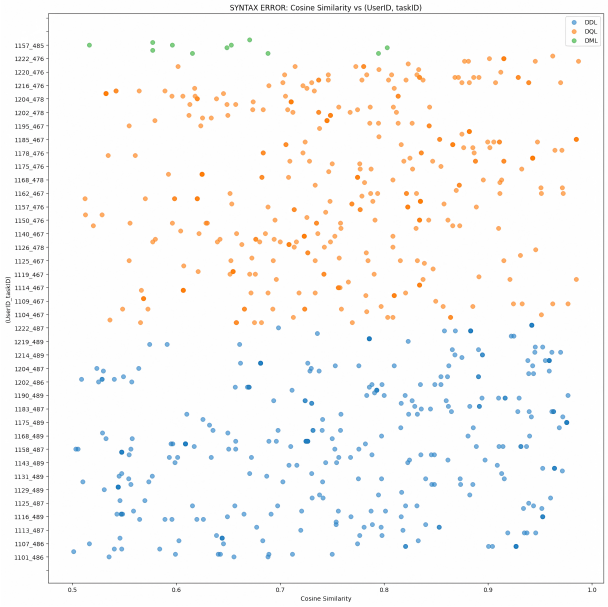

Figure 8: SQL Segregation Scatter Plot with Cosine Similarity (Syntax Error)

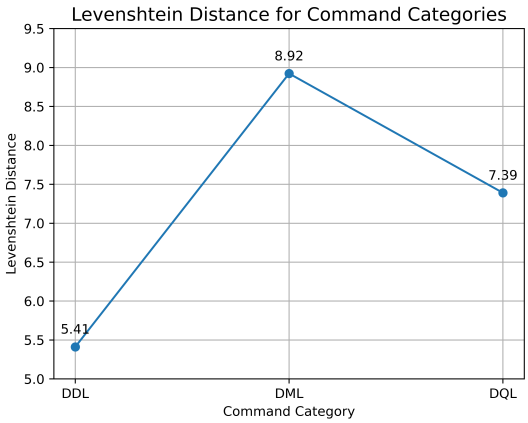

Figure 9: Levenshtein Distance values for query command categories

are displayed as a scatter plot depicting variations in similarities based on DDL, DQL or DML queries. DDL and DQL queries are prioritized as they represent a larger group in our representative tasks used for the testing, while very few DML queries were selected based on the availability in the original data.

The performance of the two models GPT-4 and GPT-3.5 can be compared through a comprehensive box plot analysis. The box plot shown in figure 6 compares the average value of the specialized SQL Query similarity metrics namely Aligon, Aouiche and Makiyama similarity. A close examination of the box plot depicts a similar median value for the average values of the three similarity metrics compared between the two models. On further investigation, it can be seen that the GPT-3.5 Turbo model shows that the Interquartile range (IQR) for the Aligon metric displays a considerably higher value than that of GPT-4. While the median is similar, the 3rd quartile is significantly longer for the GPT-3.5 model. This can be explained by GPT-4 being a more capable model in understanding SQL query data and therefore displaying a higher structural homogeneity and semantic consistency in the generated data. Subsequently, it can also be seen that while the Aouiche and Makiyama metric for GPT-4 displays some outliers near the 0.3 range, there are no outliers displayed by GPT-3.5 rather, the values are included in the upper whisker itself. An inference for this behaviour could be that GPT-4 consistently shows its efficacy in handling complex query structures and index utilization, however GPT-3.5 Turbo model generates more erroneous data and therefore no outliers explicitly.

5.2 Levenshtein Distance

Levenshtein distance, introduced primarily in 1965 by Vladimir Levenshtein has high acclaim and is widely used in bio-informatics, information theory, and other NLP tasks. Levenshtein distance (LD) in essence, is a metric used to compare the similarity between two strings. This metric works by calculating the minimum number of changes to the characters (or word) required to match the ground truth with the characters (or words) generated by the prediction model. This is typically accomplished by allocating a predetermined cost for each transformation of each character and the minimum cost thus analysed is used by the metric. [5]

Levenshtein Distance (also called string edit distance in various contexts) itself can represent the similarity of two strings. Intuitively, the higher the distance, the higher the dissimilarity between the two strings. Given two strings, S and T, with lengths m and n respectively, the edit distance between them is represented using LD, and their similarity is denoted by Sim(S, T), [8] then:

$$Sim(\mathrm{S}, \mathrm{T}) = 1 - \frac{ld}{\max(\mathrm{m}, \mathrm{n})}$$

Consequently, a Levenshtein distance of zero denotes identical strings (i.e., zero operations to match the strings), whereas as discussed a Levenshtein distance greater than 0 indicates an inherent dissimilarity of the same magnitude. To generate synthetic data in the form of SQL queries, its reasonable to expect that the threshold for Levenshtein Distance would be relatively higher compared to its typical application in NLP and information theory tasks, where the focus is on string similarities with meticulous accuracy constraints.

Although the Levenshtein algorithm is innately classic, it was further improved by Zhang et al., 2017 [13] by combining with Longest Common Subsequence (LCS) and Longest Common Substring (LCCS) and thereby improving the string similarity result based on Levenshtein Distance generating a more distinct and accurate result. This further improved the universal applicability whilst reducing the space complexity of the algorithm from the order of magnitude. After considering this enhanced string similarity context, we came to the conclusion that using this for SQL query data would not be inherently useful as it does not delve into how improved string similarity results would benefit synthetic data generated for SQL data processing tasks such as querying databases,

generating synthetic data, or any other specific SQL-related applications. A further approach called the Damerau–Levenshtein distance differs from the classical Levenshtein distance by including transpositions among its allowable operations in addition to the three classical single-character edit operations (insertions, deletions and substitutions)[3]. After various simulations of the Damerau–Levenshtein distance on the synthetic generated data the results generated were still inconspicuous.

Figure 9 shows the values obtained for the Levenshtein Distance metric for various query command categories DDL, DML and DQL. The presented Levenshtein distances (LD) for SQL query command categories offer insights into the average character-level differences between these categories. Our analyses show that on average, the categories differed by a distance of approximately 7.24.

5.2.1 Data Definition Language Commands. The average LD for DDL queries was calculated at 5.412. This distance indicates that the lowest disparity for the synthetic data generation was for the DDL command category. The DDL category of queries include queries relate to defining or modifying database structure (creating or altering tables, indexing, renaming, constraints), therefore a comparatively lower value indicates the queries synthetically generated exhibited relatively minor differences on average.

5.2.2 Data Manipulation Language Commands. The average LD for DML queries was found to be 8.923. This distance highlights the maximum degree of diversity or dissimilarity for synthetic data generation between the ground truth and the prediction model used. DML queries as opposed to DDL queries involve operations such as inserting, updating, deleting data within tables, and therefore the metric suggests a higher differences on average between the queries.

5.2.3 Data Query Language Commands. The average LD for DQL queries was found to be 7.39. DQL queries primarily SELECT statements are used to retrieve data from the database. As compared to DDL and DML queries, a distance of 7.39 lies very close to the cumulative mean of all three query categories, indicating a moderate level of variability among DQL queries. This further indicates that the synthetically generated data was able to generate DQL query commands closed to the ground truth with moderate competency.

To summarize the evaluation results, although the comparison is subtle our evaluation provided a cosine similarity value of the generated query data for the GPT-4 model as 0.783 and that of GPT-3.5 Turbo model as 0.76. This sheds light into the handling of SQL query data by both models and suggests that GPT-4 generates better diversity of syntax and logical errors. Moreover, as further verified by domain-expert analyses GPT-4 also generates errors in varying formats and type which in turn helps in capturing the nuances of the query data more accurately. However, within the evaluation the execution time for both the models should also be mentioned as this can vary in importance depending on the scalability of the data generation. For the GPT-4 model we had a running time of over 65 minutes, while the running time for the 3.5-Turbo model was significantly shorter at 23 minutes. The shorter running time of the GPT-3.5 Turbo model suggests a higher computational efficiency compared to GPT-4. This efficiency can translate directly into cost savings, especially when processing large datasets or operating in cloud-based environments where computational resources are billed per usage. For organizations or individuals with limited budgets, the choice of model could significantly impact the overall cost of synthetic data generation projects. Therefore, the choice between GPT-4 and GPT-3.5 Turbo model might involve considering the trade-off between the quality of the synthetic data generated and the efficiency of the process. If capturing the various nuances of the original data along with an inherent understanding of the queries is important for a specific use-case then GPT-4 proves to be better. If however, the goal is efficiency of the model - perhaps for scalability concerns or iterative development and testing, then GPT-3.5 Turbo model might be the winner.

6 RELATED STUDIES

A state-of-the art effort by Pinceti et. al. [10]created a comprehensive generative framework to produce artificial bus-level time-series load data for transmission networks. The model is trained using a genuine dataset consisting of more than 70 terabytes of synchrophasor observations collected over several years. The system that was made uses principal component analysis and conditional generative adversarial network models to create data at different sampling rates (up to 30 samples per second) and for different lengths of time (from seconds to years). Generative models undergo thorough testing to ensure they accurately represent the many properties of actual loads. An open-source application named LoadGAN has been created to provide researchers with access to fully trained generative models using a graphical interface. Another related work was carried out by [12] in their work, The authors suggested a new way to make synthetic datasets that is based on data. They would use deep generative adversarial networks (GANs) to learn the conditional probability distribution of important features in the real dataset and then use that information to make samples. the strategy we adopted is similar with these efforts, out system is based on students interaction data which is based on SQL. This presents a unique situation.

7 CONCLUSION

Research indicates that automated learning interventions can help remediate academic failure or educational incompetence by assessing and recommending improvements at the early stage. Analysing data from these interventions can help researchers in improving the effectiveness of different teaching strategies and make affecting and significant decisions, however there is often a scarcity of useful educational data for this purpose. To resolve this ever-lasting concern, we have developed an expert in the loop pipeline for generating structured query datasets for the training and evaluation of these intelligent agents.

The query datasets have been categorised into different language areas pertaining to the Structured Query Language namely Data Definition, Data Query, and Data Manipulation. These various categories act as a framework guideline throughout the pipeline enabling organization and generation of synthetic data using LLMs like ChatGPT-4. By maintaining a delicate balance between generating synthetic query data that is both meaningful, and sufficiently

distinct from the original dataset, a cosine similarity score of 0.76% was attained.

By employing the proposed pipeline for generating complex datasets, educators and researchers now have access to a novel approach for SQL synthetic data generation, addressing a previously unmet need in the field. The findings of this research therefore shed light on the potential of leveraging the synthetic data generation pipeline, using expert-in-the-loop methodologies to overcome the challenges of limited educational data availability, paving the way for more effective and personalized learning interventions in this field.

LIMITATIONS

Although, the presented pipeline for generating synthetic datasets shows promising results, it is important to acknowledge several limitations of our approach. One major limitation faced while using LLMs like ChatGPT for our use-case was that although these large language models excel at generating text based on patterns in the provided context and training data, they often struggle in comprehending the intricacies of SQL queries. This often produced SQL data deprived of any meaning or understanding of the task. Hence, LLM-generated datasets may lack the variability and authenticity of real educational data, limiting their effectiveness in training intelligent systems. Another limitation arises from the dearth of original educational data available to serve as context for training the models to generate meaningful data. This scarcity not only affects the model's ability to produce accurate outputs, but also constrains its scalability. Lastly, ethical implications present with using LLMs to generate synthetic datasets that mimic real student interactions might be of concern and underscore the necessity for further exploration in AI ethics pertaining to Large Language Models.

ACKNOWLEDGMENTS

We thank all reviewers for their constructive feedback.

REFERENCES

[1] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. 2014. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.* 39, 2 (may 2014), 463–489. https://doi.org/10.1007/s10115-013-0614-1

[2] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. 2006. Clustering-Based Materialized View Selection in Data Warehouses. In *Advances in Databases and Information Systems*, Yannis Manolopoulos, Jaroslav Pokorný, and Timos K. Sellis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 81–95.

[3] Gregory V. Bard. 2007. Spelling-error tolerant, order-independent pass-phrases via the Damerau–Levenshtein string-edit distance metric. In *Proceedings of the Fifth Australasian Symposium on ACSW Frontiers : 2007*, Vol. 68. Conferences in Research and Practice in Information Technology, Australian Computer Society, Inc., Ballarat, Australia, 117–124.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[5] Fang-Yi Chao, Federica Battisti, Pierre Lebreton, and Alexander Raake. 2023. Chapter 5 - Omnidirectional video saliency. In *Immersive Video Technologies*, Giuseppe Valenzise, Martin Alain, Emin Zerman, and Cagri Ozcinar (Eds.). Academic Press, 123–158. https://doi.org/10.1016/B978-0-32-391755-1.00011-0

[6] Gokhan Kul, Duc Thanh Anh Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2018. Similarity Metrics for SQL Query Clustering. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2408–2420. https://doi.org/10.1109/TKDE.2018.2831214

[7] Vitor Hirota Makiyama, Jordan Raddick, and Rafael D. C. Santos. 2015. Text Mining Applied to SQL Queries: A Case Study for the SDSS SkyServer. In *Symposium on Information Management and Big Data*. https://api.semanticscholar.org/CorpusID:17589832

[8] Artur Niewiarowski and Marek Stanuszek. 2013. Mechanism of analysis of similarity short texts, based on the Levenshtein distance. *Studia Informatica* 34, 1 (2013), 107–114.

[9] Victor Obionwu, David Broneske, Anja Hawlitschek, Veit Köppen, and Gunter Saake. 2021. Sqlvalidator–an online student playground to learn sql. *Datenbank-Spektrum* 21 (2021), 73–81.

[10] Andrea Pinceti, Lalitha Sankar, and Oliver Kosut. 2023. Generation of synthetic multi-resolution time series load data. *IET Smart Grid* 6, 5 (2023), 492–502.

[11] Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. Is ChatGPT a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476* (2023).

[12] Chi Zhang, Sanmukh R Kuppannagari, Rajgopal Kannan, and Viktor K Prasanna. 2018. Generative adversarial network for synthetic time series data generation in smart grids. In *2018 IEEE international conference on communications, control, and computing technologies for smart grids (SmartGridComm)*. IEEE, 1–6.

[13] Shengnan Zhang, Yan Hu, and Guangrong Bian. 2017. Research on string similarity algorithm based on Levenshtein Distance. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2247–2251. https://doi.org/10.1109/IAEAC.2017.8054419