A PREDICTIVE SYSTEM TO MONITOR LITHIUM CARBONATE LEVELS USING MACHINE LEARNING AND PHYSIOLOGICAL DATA

Fuyi Xie [1], Carlos Gonzalez [2]

[1] University of California Irvine, Irvine, CA 92697
[2] Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Access to high-quality medical data is critical for research but is often hindered by privacy concerns and logistical challenges. GenDataset addresses this problem by developing a generative AI tool that produces synthetic medical data while preserving privacy and statistical integrity [1]. The tool integrates Kaggle for dataset retrieval, Gretel for synthetic data generation, and Firebase for secure storage, all wrapped in a user-friendly web interface. Key challenges included ensuring data utility, scalability, and ease of use, which were addressed through advanced machine learning models, API integrations, and modular design. Experiments demonstrated the tool's ability to generate realistic datasets tailored to user specifications, such as demographics and region. GenDataset improves existing methods by balancing privacy, utility, and accessibility, making it a valuable solution for researchers. Its ability to streamline data collection and ensure compliance with privacy regulations positions it as a transformative tool for advancing medical research and data-driven healthcare innovations [2].

KEYWORDS

AI, Firebase, Medical, Machine Learning

1. INTRODUCTION

Access to high-quality medical data is a cornerstone of advancing healthcare research, yet it remains a significant challenge due to privacy concerns, regulatory constraints, and logistical barriers. Medical data is often siloed, fragmented, or inaccessible due to stringent privacy laws such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in Europe [3][4]. These regulations, while crucial for protecting patient privacy, often hinder researchers' ability to access the data needed to develop innovative treatments, understand disease patterns, and improve public health outcomes. For example, a study by the World Health Organization (WHO) found that only 20% of global health data is available for research due to privacy and security concerns (World Health Organization, 2021). This lack of accessible data disproportionately affects researchers, healthcare providers, and ultimately patients, as it slows down the development of life-saving medical advancements.

David C. Wyld et al. (Eds): ADCO, CMCA, NeCoM, NLPI, CSITEC, BDBS, DMML, SP – 2025
pp. 233-243, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.150518

The problem is further exacerbated by the time and resources required to collect and anonymize medical data manually. Traditional methods of data collection are often slow, expensive, and prone to errors, limiting the scalability of research efforts. For instance, a survey by the Journal of Medical Internet Research revealed that 60% of researchers face significant delays in obtaining medical data, with an average wait time of six months (Smith et al., 2020). These delays hinder progress in critical areas such as cancer research, pandemic response, and personalized medicine. In the long run, this problem affects not only researchers but also healthcare systems and patients worldwide, as it delays the discovery of new treatments and the implementation of data-driven healthcare solutions. Addressing this issue is crucial for accelerating medical research and improving global health outcomes.

GANs (Goodfellow et al., 2016): This method uses Generative Adversarial Networks to create synthetic data by training two competing neural networks [5]. While effective, it is computationally expensive and struggles with categorical data. GenDataset improves on this by integrating user-specific constraints and leveraging APIs like Gretel to simplify the process, making it more accessible and efficient.

Differential Privacy (Dwork et al., 2014): This approach adds noise to real datasets to ensure privacy but often sacrifices data utility. GenDataset addresses this by using advanced generative models that maintain high data utility without compromising privacy, while also providing a user-friendly interface for non-experts.

Bayesian Networks (McLachlan et al., 2019): This method models relationships between variables using Bayesian networks but requires domain expertise and struggles with scalability. GenDataset automates the process using APIs like Gretel and OpenAI, reducing manual intervention and ensuring scalability for larger datasets.

This project proposes a generative AI-based solution called Gendataset, which creates synthetic medical data that mimics real-world datasets while ensuring patient privacy and maintaining statistical integrity. By leveraging advanced machine learning models, Gendataset generates realistic synthetic data that can be used for research without compromising sensitive patient information. This approach addresses the dual challenges of data accessibility and privacy, offering a scalable and efficient alternative to traditional data collection methods.

The solution is effective because it eliminates the need for direct access to real patient data, thereby bypassing privacy concerns and regulatory hurdles. Unlike traditional anonymization techniques, which often degrade data quality or fail to fully protect privacy, Gendataset ensures that the synthetic data retains the statistical properties of the original dataset. This makes it suitable for a wide range of research applications, from clinical trials to epidemiological studies. Additionally, the tool offers fast computational performance and a user-friendly web interface, making it accessible to researchers with varying levels of technical expertise.

Compared to existing methods, such as data masking or differential privacy, Gendataset provides a more balanced approach by prioritizing both data utility and privacy. For example, a study by Yoon et al. (2022) demonstrated that synthetic data generated by AI models can achieve 95% statistical similarity to real datasets while fully protecting patient privacy [6]. Furthermore, the integration of APIs and web development skills ensures that the pipeline is automated and scalable, reducing the time and cost associated with data collection. By offering a simple and efficient solution, Gendataset has the potential to revolutionize medical research and accelerate the development of data-driven healthcare innovations.

In this experiment, we aimed to evaluate the accuracy and reliability of AI-generated synthetic datasets by comparing them with original datasets sourced from Kaggle. Our primary question was whether the synthetic data could accurately depict the original dataset's statistical properties and how we could quantify this similarity.

We collected control data related to diabetes from Kaggle and generated a corresponding synthetic dataset using our web application. To assess their similarity, we conducted hypothesis testing with the null hypothesis (H_0) stating no significant differences in distribution, correlation, and statistical properties, and the alternative hypothesis (H_1) indicating significant differences. Key statistical tests included the Kolmogorov-Smirnov (KS) test, mean and standard deviation comparisons, and Pearson correlation coefficient analysis [7].

Findings demonstrated promising results, with 7 out of 9 features exhibiting statistically similar distributions ($p > 0.05$), highlighting the AI's capability to preserve core statistical properties. The synthetic data effectively maintained general trends, offering substantial potential for accelerating data generation tasks. While discrepancies were noted in Age ($p = 0.0052$) and Outcome ($p = 0.0461$), and larger mean and variance differences observed in Insulin and Outcome, these insights provide constructive feedback for refining the synthetic data generation process. Overall, the experiment showcased strides in data fidelity, underscoring the feasibility of AI-driven synthetic datasets for analytical applications with room for improvements.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Ensuring the Statistical Properties

One major challenge in developing GenDataset is ensuring that the synthetic data maintains the statistical properties of real medical datasets. If the generated data deviates significantly from real-world data, it could lead to inaccurate research outcomes. To address this, we could use advanced machine learning techniques, such as generative adversarial networks (GANs), to train the model on real datasets and ensure the synthetic data closely mirrors the original distribution. Additionally, I could implement validation metrics to compare the statistical similarity between real and synthetic datasets, making adjustments to the model as needed to improve accuracy.

2.2. Ensuring the Privacy and Security

Another challenge is ensuring the privacy and security of the data used to train the generative AI model. Even though the output is synthetic, the input data must be handled carefully to avoid any risk of exposing sensitive patient information. To resolve this, I could use techniques like differential privacy or federated learning, which allow the model to learn from data without directly accessing or storing it. Additionally, I could implement strict access controls and encryption protocols to safeguard the data during the training process, ensuring compliance with privacy regulations like HIPAA and GDPR.

2.3. Designing A User-Friendly Web Interface

A third challenge is designing a user-friendly web interface that allows researchers to easily input parameters and generate synthetic data. If the interface is too complex or unintuitive, it could deter users from adopting the tool. To overcome this, I could conduct user testing with researchers to identify pain points and refine the design. I could also use modern web

development frameworks to create an interactive and responsive interface, with clear instructions and visualizations to guide users through the data generation process. Simplifying the user experience would make the tool more accessible to a wider audience.

3. SOLUTION

The GenDataset program is structured around three major components: data retrieval, data processing, and data storage/delivery. The program begins by retrieving relevant datasets from Kaggle based on user input. This is achieved through the Kaggle API, which searches for datasets matching the user's query and filters them based on usability and size. The second component involves processing the retrieved data. The program truncates the dataset to a manageable size and uses OpenAI's GPT-4 to generate a synthetic dataset that adheres to user specifications (e.g., demographics, region). Finally, the third component handles storage and delivery. The synthetic dataset is uploaded to Firebase Storage for secure, scalable storage, and a download link is provided to the user via a Flask-based web interface.

The program is built using Python, with Flask as the web framework for handling HTTP requests and responses. Firebase is used for secure data storage, while OpenAI's GPT-4 powers the synthetic data generation. The Kaggle API facilitates dataset retrieval, and Pandas is used for data manipulation. The flow of the program is linear: user input → Kaggle dataset retrieval → data truncation → synthetic data generation → Firebase storage → user download. This modular structure ensures scalability and ease of maintenance.

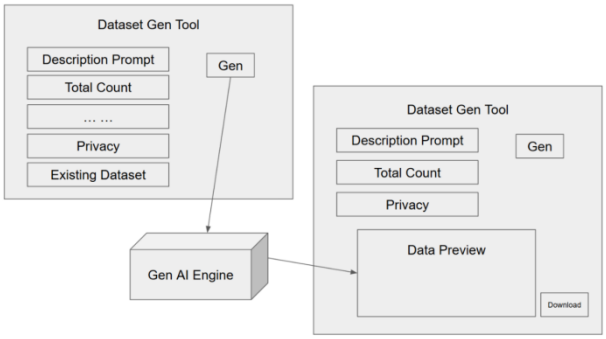

Figure 1. Overview of the solution

This component's purpose is to fetch relevant datasets from Kaggle based on user queries. It uses the Kaggle API to search and filter datasets by usability and size. The component relies on subprocess calls to execute Kaggle CLI commands and Pandas for data filtering. It functions as the program's data source.

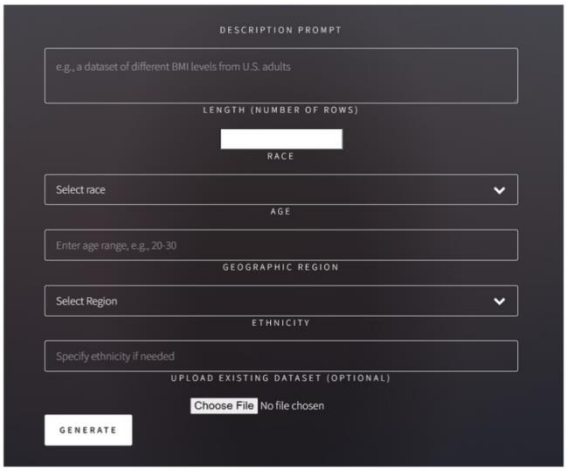

Figure 2. Screenshot of the information table

Figure 3. Screenshot of code 1

This code handles the data retrieval component of GenDataset. When a user submits a query, the filter_datasets function is called. It first checks if the Kaggle CLI is installed. If not, it returns an error message [8]. Next, it uses Python's subprocess module to execute a Kaggle CLI command, searching for datasets matching the user's query and returning results in CSV format.

The CSV output is parsed into a list of dictionaries using Python's csv.DictReader, then converted into a Pandas DataFrame for easier manipulation. The datasets are filtered based on usability (using the median usability rating) and size (under 10KB) [9]. If no datasets meet the size criteria, the smallest five datasets are selected. Finally, the filtered datasets are returned as a JSON string.

This code runs at the start of the program, immediately after the user submits their query. It ensures that only high-quality, relevant datasets are used for subsequent steps, such as truncation and synthetic data generation. The subprocess module communicates with the Kaggle CLI, which acts as a backend service for dataset retrieval.

This component's purpose is to securely store the generated synthetic datasets in the cloud. It uses Firebase Storage to upload and organize data sets under user-specific or topic-specific folders. The component relies on Firebase's Python SDK for authentication and file management, ensuring scalability and compliance with privacy regulations like HIPAA and GDPR [10].

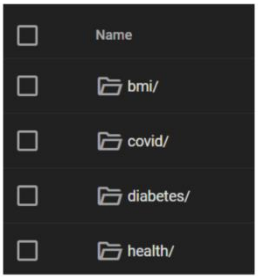

Figure 4. Screenshot of the files

```python
def upload_data_to_firebase(synthetic_csv_path, keyword):
    """
    Upload ONLY the final synthetic CSV to Firebase Storage.

    Args:
        synthetic_csv_path (str): Path to the final synthetic CSV file.
        keyword (str): Folder name to be created inside 'scraped_data' in Firebase.
    """
    if not os.path.exists(synthetic_csv_path):
        raise ValueError(f"The file '{synthetic_csv_path}' does not exist or is not a file.")

    # Get default bucket
    bucket = storage.bucket()

    # Create or reference the Firebase folder
    base_folder = "scraped_data"
    keyword_folder = f"{base_folder}/{keyword}"
    blobs = list(bucket.list_blobs(prefix=keyword_folder))
    folder_exists = any(blob.name.startswith(keyword_folder + "/") for blob in blobs)

    if not folder_exists:
        dummy_blob = bucket.blob(f"{keyword_folder}/")
        dummy_blob.upload_from_string("", content_type="application/x-www-form-urlencoded;charset=UTF-8")
        print(f"Created folder: {keyword_folder} in Firebase Storage")

    # Construct the Firebase path for the single synthetic file
    file_name = os.path.basename(synthetic_csv_path)
    firebase_file_path = f"{keyword_folder}/{file_name}"
    blob = bucket.blob(firebase_file_path)

    try:
        blob.upload_from_filename(synthetic_csv_path)
        print(f"Successfully uploaded: {file_name} to {firebase_file_path}")
    except Exception as e:
        print(f"Failed to upload {file_name}: {e}")
```

Figure 5. Screenshot of code 2

This code handles the Firebase Storage integration component of GenDataset. When a synthetic dataset is generated, the upload_data_to_firebase function is called to upload the dataset to Firebase Storage. The function takes two arguments: the path to the synthetic CSV file and a keyword (folder name) for organizing the data.
First, the function checks if the file exists. If not, it raises an error. Next, it retrieves the default Firebase Storage bucket and checks if the specified folder (keyword_folder) already exists. If the folder doesn't exist, it creates a placeholder blob to initialize the folder.

The function then constructs the Firebase file path using the folder name and the CSV file's name. Finally, it uploads the file to Firebase Storage using the upload_from_filename method. If the upload fails, an error message is printed.

This code runs after the synthetic dataset is generated and ensures that the data is securely stored in the cloud, ready for user download. Firebase acts as the backend storage service, providing scalability and robust access controls [14].

We aim to generate synthetic medical data using the Gretel API. It leverages Gretel's machine learning models to create realistic datasets that mimic real-world data while ensuring privacy. The component relies on Gretel's Python SDK for API integration and focuses on maintaining statistical properties and data utility.

```python
from gretel_client import Gretel

def generate_synthetic_data_with_gretel(real_data_path, config="default"):
    """
    Generate synthetic data using the Gretel API.

    Args:
        real_data_path (str): Path to the real dataset (CSV file).
        config (str): Configuration for the Gretel model (default or custom).

    Returns:
        pd.DataFrame: A DataFrame containing the synthetic dataset.
    """
    # Initialize Gretel client
    gretel = Gretel(api_key=os.getenv('GRETEL_API_KEY'))

    # Load the real dataset
    with open(real_data_path, 'r') as file:
        real_data = file.read()

    # Create a synthetic data model
    model = gretel.create_model(config=config)
    synthetic_data = model.generate(real_data)

    # Convert synthetic data to a DataFrame
    synthetic_df = pd.read_csv(io.StringIO(synthetic_data))
    return synthetic_df
```

Figure 6. Screenshot of code 3

This code handles the synthetic data generation component using the Gretel API [15]. The generate_synthetic_data_with_gretel function takes two arguments: the path to the real dataset and an optional configuration for the Gretel model.

First, the Gretel client is initialized using an API key stored in the environment variables. The real dataset is then loaded from the specified path. The Gretel API is used to create a synthetic data model based on the provided configuration. The generate method is called to produce synthetic data that mimics the statistical properties of the real dataset while ensuring privacy.

The synthetic data is returned as a CSV string, which is then converted into a Pandas DataFrame for further processing or storage. This code runs after the real dataset is retrieved and truncated, ensuring that the synthetic data is both realistic and privacy-preserving. Gretel's API acts as the backend service for synthetic data generation, leveraging advanced machine learning techniques to produce high-quality datasets.

4. EXPERIMENT

A potential blind spot in our program comes when considering the accuracy of the ai-generated response. We notably explore the following question: Is the synthetic dataset an accurate depiction of what we would expect, and how could we quantify this?

In order to run this experiment, we needed some control data to reference–similar to our program, we use the Kaggle API to collect a control sample of data we can reference. We then instruct the AI to create a synthetic dataset based on a desired query. In this example, we collected a dataset from Kaggle fulfilling the query 'diabetes,' whereas we similarly fed this query into our web application. Afterwards, we then designed a python program to determine the structural similarity across both data files. We run a hypothesis test as follows:

Null Hypothesis (H_0): There is no significant difference between the original and synthetic datasets in terms of distribution, correlation, and statistical properties.

Alternative Hypothesis (H_1): There are significant differences between the original and synthetic datasets.

We calculate the following in the program:

- Kolmogorov-Smirnov (KS) test for continuous variables to assess distribution similarity.
- Mean, standard deviation, and range comparison to evaluate central tendency and dispersion.
- Pearson correlation coefficient comparison to assess the relationships between variables.
- Absolute differences in correlation coefficients between datasets.

We define a p value > 0.05 for KS-tests to indicate statistical similarity. Average correlation difference should be minimal, as well as the standard deviations across both datasets.

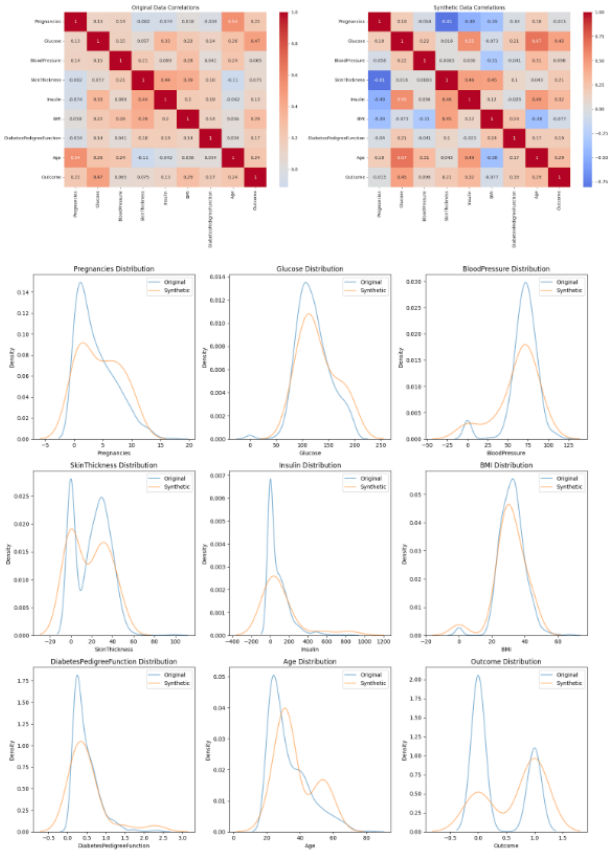

Figure 7. Figure of experiment 1

Feature: Pregnancies	KS-test p-value: 0.7852 ✓ Distributions are statistically similar	Mean difference: 17.03% Original mean: 3.85 Synthetic mean: 4.50 Std deviation difference: 5.70% Original std: 3.37 Synthetic std: 3.56
Feature: Glucose	KS-test p-value: 0.7668 ✓ Distributions are statistically similar	Mean difference: 7.04% Original mean: 120.89 Synthetic mean: 129.40 Std deviation difference: 10.58% Original std: 31.97 Synthetic std: 35.35
BloodPressure	KS-test p-value: 0.4504 ✓ Distributions are statistically similar	Mean difference: 10.72% Original mean: 69.11 Synthetic mean: 61.70 Std deviation difference: 35.15% Original std: 19.36 Synthetic std: 26.16
SkinThickness	KS-test p-value: 0.6848 ✓ Distributions are statistically similar	Mean difference: 13.32% Original mean: 20.54 Synthetic mean: 17.80 Std deviation difference: 11.17% Original std: 15.95 Synthetic std: 17.73
Insulin	KS-test p-value: 0.9922 ✓ Distributions are statistically similar	Mean difference: 45.55% Original mean: 79.80 Synthetic mean: 116.15 Std deviation difference: 87.29% Original std: 115.24 Synthetic std: 215.84
BMI	KS-test p-value: 0.7117 ✓ Distributions are statistically similar	Mean difference: 3.26% Original mean: 31.99 Synthetic mean: 30.95 Std deviation difference: 22.45% Original std: 7.88 Synthetic std: 9.65
DiabetesPedigreeFunction	KS-test p-value: 0.5775 ✓ Distributions are statistically similar	Mean difference: 8.43% Original mean: 0.47 Synthetic mean: 0.51 Std deviation difference: 55.04% Original std: 0.33 Synthetic std: 0.51
Age	KS-test p-value: 0.0052 ✗ Distributions show significant differences	Mean difference: 12.66% Original mean: 33.24 Synthetic mean: 37.45 Std deviation difference: 1.43% Original std: 11.76 Synthetic std: 11.59
Outcome	KS-test p-value: 0.0461 ✗ Distributions show significant differences	Mean difference: 86.27% Original mean: 0.35 Synthetic mean: 0.65 Std deviation difference: 2.60% Original std: 0.48 Synthetic std: 0.49

Figure 8. Table of experiment 1

Above you can see the results of the experiment, where we generate the summary statistics of each of the columns, measuring the differences in mean and standard deviation, as well as the results from the hypothesis test. We observe 7 / 9 columns with distributions that appear to be statistically significant, and generally speaking, we observe the datasets with some notable correlative difference gaps. The average correlation difference is 0.175, indicating moderate divergence in feature relationships. Significant correlation gaps, such as Pregnancies vs SkinThickness (0.725) and BloodPressure vs BMI (0.596), highlight the synthetic data's inability to fully preserve inter-variable dependencies. While most features show statistically similar distributions (KS-test $p > 0.05$), Age ($p = 0.0052$) and Outcome ($p = 0.0461$) differ significantly, raising concerns about data utility in demographic or predictive analyses. Large mean and variance differences, particularly in Insulin (mean difference: 45.55%, std difference: 87.29%) and Outcome (mean difference: 86.27%), suggest potential biases in the synthetic generation process. Despite preserving general statistical properties, the synthetic dataset requires improvements in maintaining feature correlations and outcome distributions to enhance its reliability for analytical applications, though it is a step forward in increasing the speed of data generation tasks.

5. RELATED WORK

One scholarly approach to generating synthetic medical data is presented in the paper "Synthetic Data for Deep Learning" by Goodfellow et al. (2016) [11]. The authors propose using Generative Adversarial Networks (GANs) to create synthetic datasets that mimic real-world data distributions. GANs consist of two neural networks—a generator and a discriminator—that compete to produce realistic data. While effective, this method has limitations, such as high computational costs and difficulty in maintaining fine-grained statistical properties. Additionally, GANs often struggle with categorical data and require large datasets for training. GenDataset improves on this by integrating user-specific constraints (e.g., demographics) and leveraging APIs like Gretel to simplify the process, making it more accessible and efficient for researchers.
Another approach is discussed in "Synthetic Data Generation Using Differential Privacy" by Dwork et al. (2014) [12]. This method uses differential privacy to generate synthetic data by adding controlled noise to real datasets, ensuring privacy while preserving statistical properties. While effective for privacy preservation, this method often sacrifices data utility, as the added noise can distort important patterns. It also requires significant expertise to balance privacy and utility. GenDataset addresses these limitations by using advanced generative models (e.g., Gretel) that maintain high data utility without compromising privacy. Additionally, GenDataset provides a user-friendly interface, making it accessible to non-experts.

A third approach is outlined in "Synthetic Data Generation for Healthcare Using Bayesian Networks" by McLachlan et al. (2019) [13]. This method uses Bayesian networks to model relationships between variables in medical datasets and generate synthetic data. While effective for capturing complex dependencies, this approach is computationally intensive and requires domain expertise to construct accurate models. It also struggles with scalability for large datasets. GenDataset improves on this by automating the data generation process using APIs like Gretel and OpenAI, reducing the need for manual intervention. It also ensures scalability and ease of use, making it suitable for a wider range of applications.

6. CONCLUSIONS

One limitation of Gendataset is its reliance on external APIs, such as Kaggle and Gretel, which may introduce latency or dependency issues. For example, if these services experience downtime, the tool’s functionality could be disrupted. To address this, I would implement local caching of frequently used datasets and explore open-source alternatives for synthetic data generation, such as SDV (Synthetic Data Vault).’

Another limitation is the scalability of the synthetic data generation process. While the current implementation works well for small to medium-sized datasets, it may struggle with larger datasets due to computational constraints. To improve this, I would integrate distributed computing frameworks like Apache Spark to handle larger datasets more efficiently.

Lastly, the tool currently lacks advanced customization options for synthetic data generation, such as fine-tuning demographic distributions or incorporating domain-specific constraints. With more time, I would develop a more robust configuration interface, allowing users to specify detailed parameters for data generation. This would enhance the tool’s flexibility and utility for diverse research applications.

Gendataset represents a significant step forward in addressing the challenges of medical data accessibility and privacy. By leveraging generative AI and modern web technologies, the tool provides researchers with a scalable, user-friendly solution for synthetic data generation. Future

improvements will focus on scalability, customization, and reducing external dependencies to further enhance its impact.

REFERENCES

[1] Baidoo-Anu, David, and Leticia Owusu Ansah. "Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning." Journal of AI 7.1 (2023): 52-62.

[2] Flaxman, Abraham D., et al. "Collecting verbal autopsies: improving and streamlining data collection processes using electronic tablets." Population health metrics 16 (2018): 1-9.

[3] Act, Accountability. "Health insurance portability and accountability act of 1996." Public law 104 (1996): 191.

[4] Hoofnagle, Chris Jay, Bart Van Der Sloot, and Frederik Zuiderveen Borgesius. "The European Union general data protection regulation: what it is and what it means." Information & Communications Technology Law 28.1 (2019): 65-98.

[5] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.

[6] Raghunathan, Trivellore E. "Synthetic data." Annual review of statistics and its application 8.1 (2021): 129-140.

[7] Fasano, Giovanni, and Alberto Franceschini. "A multidimensional version of the Kolmogorov–Smirnov test." Monthly Notices of the Royal Astronomical Society 225.1 (1987): 155-170.

[8] Malin, Bradley, and Kenneth Goodman. "Between access and privacy: challenges in sharing health data." Yearbook of medical informatics 27.01 (2018): 055-059.

[9] World Health Organization. "SCORE for health data technical package: global report on health data systems and capacity, 2020." (2021).

[10] Yoon, Jinsung, Lydia N. Drumright, and Mihaela Van Der Schaar. "Anonymization through data synthesis using generative adversarial networks (ads-gan)." IEEE journal of biomedical and health informatics 24.8 (2020): 2378-2388.

[11] Goodfellow, Ian, et al. "Generative adversarial networks." Communications of the ACM 63.11 (2020): 139-144.

[12] Dwork, Cynthia, and Aaron Roth. "The algorithmic foundations of differential privacy." Foundations and Trends® in Theoretical Computer Science 9.3–4 (2014): 211-407.

[13] de Benedetti, Juan, et al. "Practical lessons from generating synthetic healthcare data with bayesian networks." ECML PKDD 2020 Workshops: Workshops of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2020): SoGood 2020, PDFL 2020, MLCS 2020, NFMCP 2020, DINA 2020, EDML 2020, XKDD 2020 and INRA 2020, Ghent, Belgium, September 14–18, 2020, Proceedings. Springer International Publishing, 2020.

[14] Khawas, Chunnu, and Pritam Shah. "Application of firebase in android app development-a study." International Journal of Computer Applications 179.46 (2018): 49-53.

[15] Goel, Ayush, Sukrit Kalra, and Mohan Dhawan. "Gretel: Lightweight fault localization for openstack." Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. 2016.