Generating Bank Transaction Sequences with Tabular GAN models

by

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Science

Supervisor: Dr. Hamid Usefi

Co-Supervisor: Dr. Alexander Bihlo

Department of Computer Science

Memorial University of Newfoundland

April 2024

St. John's Newfoundland

Abstract

The digital age has equipped financial institutions with vast amounts of data. Privacy concerns have posed challenges to harnessing this data's full potential. Generation of synthetic data is one of the most promising solutions for allowing analysis of the patterns and trends contained in this data without compromising privacy. Although initial methods for generating synthetic data were basic, emerging generative models have expanded the possibilities. However, generating synthetic data for unique datasets, like bank transaction sequences, remains challenging. These sequences exhibit complex variability driven by the various customer transaction behaviors, distinguishing them from the more predictable patterns in other data types. We propose BankGAN, an innovative conditional tabular GAN architecture designed specifically for synthesizing bank transaction sequences that exhibit non-uniform date patterns. We show that BankGAN outperforms a recurrent neural network (RNN)-based model in achieving superior statistical resemblance to real data. Moreover, it excels at replicating features of periodic transactions, surpassing both the RNN and transformer-based models. BankGAN distinguishes itself by generating privacy-preserving synthetic data without compromising data quality—a stark contrast to the existing models where adding privacy-preserving guarantees typically degrades performance.

General Summary

In this Master's thesis, we explore the pressing issue of privacy in financial data analysis. With the advent of digital banking, financial institutions are flooded with data, yet the privacy of individuals remains a paramount concern. Our research introduces a novel solution: the development of "BankGAN," an artificial intelligence model designed to generate synthetic bank transaction sequences. These sequences mimic the complex patterns of real transactions without compromising individual privacy. Unlike traditional methods, BankGAN excels in replicating intricate transaction details while ensuring data remains detached from personal identities. This breakthrough offers a safer, more ethical approach to analyzing financial data, enabling banks to harness the power of their data for improvement and innovation while upholding the privacy of their customers. This work has the potential to reshape how financial data is utilized, balancing the need for analysis with the imperative of privacy.

Acknowledgements

My journey through this research endeavor has been significantly shaped and supported by a remarkable group of individuals and organizations, to whom I owe a debt of gratitude.

My sincere appreciation goes to Dr. Alexander Bihlo, my supervisor at Memorial University, and, Dr. John Hawkin and Dr. Farzaneh Shoeleh from Verafin for the unwavering guidance, mentorship, and valuable insights that have been instrumental in the preparation and completion of this thesis. Their expertise and encouragement have been pivotal to my academic growth.

A special note of thanks goes to Dr. Kyle Nickerson, a distinguished PhD graduate from Memorial University. His collaborative spirit, strategic brainstorming, and technical prowess have greatly contributed to the success of this project.

I am profoundly grateful to the School of Graduate Studies at Memorial University of Newfoundland and Labrador for awarding me the Master's fellowship. This financial support has been a cornerstone in enabling my research and scholarly endeavors. Additionally, I would like to acknowledge the generous support from the AARMS CRG on Scientific Machine Learning, which covered part of my funding at Memorial University. Furthermore, I wish to express my gratitude to Mitacs for the Accelerate funding, which has facilitated applied research vital to my thesis. The funding from Verafin has also

been essential in supporting my work, for which I am immensely thankful.

Statement of Contributions

The research presented in this thesis was accepted at the Canadian AI 2024 Conference, with collaborative authorship including Hamideh Mehri, Alexander Bihlo, Farzaneh Shoeleh, John Hawkin, and Kyle Nickerson. The development and design of the BankGAN model, execution of experiments, and initial manuscript preparation were spearheaded by Hamideh Mehri. Guidance and project supervision were provided by Alexander Bihlo, Farzaneh Shoeleh, and John Hawkin.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Financial services now have access to enormous datasets due to the development of digital technology, but due to privacy issues, they face challenges in exploiting the potential of their data. Concerns around privacy impede academic research and the development of cutting-edge AI-based applications [1]. Any AI-based service accessing, processing, or storing large amounts of sensitive data becomes a potential vulnerability point. As the financial sectors are among the top targets for cybercriminals [2], data breaches can occur, which leads to direct financial losses and significant reputational damage. In addition, as financial institutions push for cutting-edge services, they often collaborate with third-party tech providers. While offering advanced capabilities, these integrations introduce another layer of potential vulnerability and complexity in ensuring end-to-end privacy [1]. Generating high-quality synthetic

data is one of the countermeasures [3]. Synthetic data is artificially generated to mimic the properties and structure of real data without copying any of the original data [4]. By adopting this strategy, financial institutions can amass data that mirrors real-world datasets without compromising personal or corporate confidentiality.

While the earliest methods for generating synthetic data relied on computational simulations [5, 6, 7, 8], the rise of deep generative models (DGMs), including generative adversarial networks (GANs) [9], Variational Auto Encoders (VAEs) [10], diffusion models [11], and transformers [12] has revolutionized this field. These models excel in crafting homogenous data types, such as images or texts. However, synthesizing tabular data— the predominant data type in the financial domain—poses unique challenges [13], including maintaining intricate inter-feature correlations, ensuring a statistical similarity to the original data, and adhering to the specificities of structured data [14]. In addition to these challenges common to all tabular datasets, each type of data may also involve unique challenges. One of the crucial datatypes in the financial domain is bank transaction sequences, which reflect an individual's or organization's financial behavior. However, generating their synthetic counterparts is a complex endeavor. One primary challenge stems from the unpredictable intervals at which transactions occur [15]. So the dataset cannot be classified as regular time-series, even though the order of rows is essential. There have been recent

endeavors to generate non-sequential and sequential tabular data with regular time intervals. Of these, Banksformer [15], which is transformer-based, is the only model designed explicitly for generating synthetic financial transaction sequences.

The primary objective of this study is to generate high-quality synthetic bank transaction sequences that can serve as substitutes for original data. This allows for the sharing of data across and within organizations when protecting data privacy is a legal requirement. To achieve this, we undertake several key activities outlined in the subsequent chapters. In Chapter 2, we detail the development of BankGAN alongside a baseline model for comparative analysis. BankGAN is a tabular GAN-based model that stands out from existing frameworks by integrating a sequential conditional vector and leveraging a date generation mechanism inspired by the Banksformer model [15], which facilitates the simulation of transaction sequences. In contrast, our baseline model employs a Recurrent Neural Network (RNN) approach to synthesize bank transaction sequences. This model serves a dual purpose: it offers a point of comparison with Banksformer [15], a transformer-based model, and establishes an additional reference for evaluating BankGAN's performance.

Chapter 3 focuses on the evaluation and comparison of the synthetic data generated by the three models: BankGAN, the baseline RNN model, and Banksformer. This includes assessing the models' ability to accurately repli-

cate both periodic and sporadic transactions and examining the privacy integrity of the generated data. Throughout the remainder of this chapter we begin with a literature study reviewing techniques for synthesizing tabular data, providing context for our approach. This is followed by a detailed exploration of CTGAN, which is referenced to elucidate its role in generating synthetic tabular data. Additionally, the chapter includes an in-depth explanation of the Banksformer model to enhance understanding of its integration and function within BankGAN.

1.1 Related Work

Synthetic data, defined as data generated via algorithms or mathematical models rather than direct real-world measurement, plays a crucial role in modern data science [4]. Its significance is particularly pronounced in fields where real data is scarce, sensitive, or subject to privacy concerns, such as healthcare and finance. In these domains, synthetic data is not only supplementing [16, 17] but, in some instances, replacing real datasets [18, 19] to safeguard privacy while facilitating research and development. The enhancement of real datasets using synthetic data is particularly advantageous for improving small or imbalanced datasets in machine learning applications, offering a balanced and enriched data environment for more effective training and analysis [20, 21]. Similarly, in the development of autonomous vehicles, synthetic data proves

indispensable, especially for simulating rare events like accidents under extreme conditions, where real data is often limited [22, 23].

The generation of synthetic data can be broadly categorized into two approaches: process-driven methods and data-driven methods [24]. Process-driven methods are grounded in simulating physical processes, usually necessitating expert knowledge and human interaction for accurate modeling. These techniques are embodied in various forms such as numerical simulations, agent-based modeling, discrete-event simulations, and Monte Carlo simulations [24].

Agent-based modeling (ABM) is a powerful technique used for synthesizing payment data, particularly in the context of banking and mobile payment systems. This approach effectively captures the behaviors of real actors involved in financial transactions. By modeling different actors in payment systems and simulating their interactions, ABM can create realistic scenarios. Once the agents interact realistically, the system can simulate new scenarios, thereby producing synthetic datasets that are invaluable in studying and detecting fraudulent patterns in financial transactions [25]. Notable ABM implementations in this field include BankSim [26], PaySim [27], AMLSim [28], and Retsim [29], each contributing uniquely to the domain of synthetic data generation. However, these approaches have limitations: they are complex and resource-intensive, require substantial data that is challenging to obtain, and face issues with scalability and applicability to different contexts. Addition-

ally, there is a risk of oversimplification, which could lead to inaccurate results. In addition to the specific challenges of agent-based modeling, process-driven approaches in general involve the need for domain experts to design tailored simulators. The authenticity of the resulting data heavily depends on the simulator's design and may not always accurately reflect real-world data. However, a significant advantage of these methods is their ability to avoid the direct use of private data, thereby reducing the risk of exposing sensitive information [24].

More recently, data-driven methods have gained attention as big data and deep learning algorithms gained momentum in the research community. In this approach a machine learning model is trained to produce data that matches the desired distribution, eliminating the need for manually creating simulators. The process of selecting or developing an appropriate generative model, while simpler than simulator design, still requires some effort. These generative models are typically more adaptable across various domains, enabling the creation of a versatile framework for generating synthetic data that can replicate different types of datasets. However, a potential downside is that the model might inadvertently 'memorize' the training data, which poses a risk of revealing private information if not managed with caution.

In the realm of data-driven methodologies, there are four prominent types of deep generative models (DGMs) that have gained significant attention: Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs),

Transformers, and Diffusion Models. These models have shown remarkable success in generating text, audio, and images. However, their application in synthesizing tabular data, a common data structure across various fields, is still in its nascent stages. We divide the related studies using DGMs to generate tabular data into sequential and non-sequential data, as each category has its own unique characteristics.

Generating non-sequential tabular data. Among DGMs, GANs have been favored for tabular data generation due to their success in other domains, e.g. image [30], coupled with their flexibility and ability to be customized for specific requirements in generating tabular data. This has established GANs as the preferred choice for crafting synthetic tabular datasets, despite the valuable contributions of VAEs [31, 32, 33, 34] and diffusion models [35]. GAN models for tabular data have been developed for many specific use cases, including TableGAN [36] for privacy preservation, ehrGAN [37] for realistic health record generation, and CrGAN [38] for Passenger Name Records (PNR) creation. A shared limitation among these models is their inability to generate specific data categories on demand, leading to the development of conditional tabular GAN models [32, 39, 40, 41]. Conditional tabular GANs incorporate conditional vectors into the training and generation process, allowing for the controlled generation of data across specified attributes. CTGAN [32], as the precursor in this area, introduced significant innovations such as *mode-specific normal-*

ization for handling continuous features with multi-modal distributions, and a *training-by-sampling* strategy to guarantee comprehensive representation of categorical features during training. These innovations significantly improved the model's ability to accurately capture varied data distributions and effectively handle categorical variables of high cardinality.

Generating sequential tabular data. Sequential tabular data generation is divided into handling data with regular and irregular time intervals. For regular intervals, models such as TimeGAN [42] and DoppelGANger [43] leverage RNN and GAN integration to capture time dependencies, whereas TimeVAE [44] utilizes a VAE approach, and recent efforts explore diffusion models for this purpose as well [45, 46]. These approaches, however, predominantly cater to sequences with consistent timing between data points. Conversely, Banksformer [15], based on a decoder-only transformer architecture, stands as a unique solution for generating synthetic sequential tabular data with irregular time intervals, focusing on bank transaction sequences. Banksformer [15] employs a special *conditional training and generation* and *date mechanism* to simulate the irregular timing patterns characteristic of transactional data.

Integrating Banksformer's [15] *conditional training and generation* techniques and *date mechanism* into a StackedLSTM Autoencoder [47, 48], we set a new baseline for our investigation. The StackedLSTM Autoencoder enhances

the traditional LSTM by adding depth, enabling better handling of complex sequences, making it suitable for multivariate multi-step time series forecasting [47, 48]. In our study, we adopt a conditional Sequence to Sequence (Seq2Seq) learning approach akin to Banksformer's [15] methodology for both training and generating bank transaction sequences. This method incorporates *conditional generation* and the Banksformer *date mechanism* [15]. By mirroring Banksformer's [15] training and generation process, this baseline model facilitates a direct comparison between LSTM-based and attention-based models in generating synthetic bank transactions.

Building upon the baseline, our research introduces BankGAN, a novel generative adversarial network (GAN) model specifically tailored for the generation of synthetic bank transaction sequences. BankGAN represents a significant leap forward from traditional GAN frameworks by integrating several unique elements. Foremost among these is the adoption of the *sequence conditional vector*, a novel component that encodes sequential information, thereby enabling the model to maintain the intrinsic chronological order present in transactional data. This is crucial for replicating the irregular and sporadic nature of financial transactions, which are not adequately captured by standard GANs. Furthermore, BankGAN incorporates the date generation mechanism inspired by Banksformer, which allows it to simulate the varying intervals between transactions. By blending the foundational principles of CTGAN

[32] with the advanced capabilities of Banksformer, BankGAN demonstrates competitive performance, particularly in capturing transaction amount distributions, and cash flow distributions of recurring transactions. Furthermore, BankGAN exhibits significant potential in preserving the privacy of synthetic data, ensuring that the generation process does not affect the quality of the produced datasets.Through this model, we aim to bridge the gap between the need for data richness in financial analytics and the imperative for strict privacy adherence, thereby enabling safer and more robust AI-driven financial services.

1.2 CTGAN

CTGAN [32] (Conditional Generative Adversarial Network), depicted in Figure 1.1, is a type of GAN designed to generate synthetic tabular data. Initially, the model distinguishes between discrete and continuous features: discrete features undergo a one-hot encoding transformation, while continuous features are modeled using a Gaussian mixture model and mode-specific normalization to preserve their statistical properties. The core of CTGAN's [32] methodology lies in the use of a conditional vector, which guides the synthetic data generation, ensuring that the output adheres to specific conditions or distributions, thereby enhancing the utility and relevance of the generated data.

The architecture of CTGAN [32] is built upon the principles of Generative

Adversarial Networks (GANs), where a generator and a discriminator engage in an iterative adversarial process. The generator aims to produce synthetic data instances that are indistinguishable from real data, informed by the conditional vectors to meet specific data conditions. Meanwhile, the discriminator evaluates the authenticity of the generated data against actual data, improving its ability to identify real versus synthetic samples. This adversarial training continues until the generator produces data sufficiently realistic that the discriminator can no longer easily differentiate from real data.

To delve deeper into the workings of CTGAN, it is beneficial to segment our explanation into several key areas: the processing of discrete and continuous features as they are fed into the discriminator, the construction of the conditional vector, the architecture of the generator and discriminator, generator loss function and discriminator loss function. This comprehensive breakdown will provide a clearer understanding of CTGAN's intricate mechanics.

1.2.1 The Processing of Discrete and Continuous Features

CTGAN [32] is specially tailored to handle various challenges associated with non-image data, like mixed data types (e.g. continuous and discrete features). CTGAN manages discrete features by converting them into one-hot encoded vectors. Continuous data handling becomes more complex when the features

exhibit multi-modal distributions. CTGAN utilizes a *"mode specific normalization"* technique to address this. The tool deploys the Variational Gaussian Mixture Model (VGMM) to automatically identify the number of modes present in the distribution of a continuous feature. In essence, VGMM is a statistical model that aims to fit the data with multiple Gaussian distributions.

For any given continuous data point, CTGAN assesses the likelihood of that data point originating from each Gaussian distribution determined by VGMM. Based on these likelihoods, CTGAN decides which Gaussian distribution the data point most likely came from. This data point then undergoes scaling, and a one-hot encoded vector represents its corresponding Gaussian distribution. Together, these provide the input vector for the discriminator.

For instance, if a continuous feature has three different modes or Gaussian distributions identified by Variational Gaussian Mixture Model (VGMM), and we take a sample value of 5.2: VGMM might assign probabilities of 0.1, 0.7, and 0.2 to this sample belonging to the three respective Gaussian distributions. Given these probabilities, the sample is inferred to have most likely come from the second Gaussian distribution. If normalizing the sample value of 5.2 results in

$$\frac{5.2 - \eta_2}{4\phi_2},$$

where η_2 and ϕ_2 are mean and standard deviation of the second Gaussian

distribution, the input vector for the discriminator becomes

$$\left[\frac{5.2 - \eta_2}{4\phi_2}, 0, 1, 0\right],$$

with the latter three digits representing the one-hot encoded vector for the second Gaussian distribution.

1.2.2 Conditional Vector

A conditional generator is a technique used to generate data samples that match a particular condition. In the context of CTGAN, it is used to ensure an even distribution of discrete values in generated samples.

For this purpose a *conditional vector* is defined, and used as input to the generator network during training, in order to generate synthetic data that satisfies the given conditions.

The conditional vector is a binary vector that encodes the conditions that must be satisfied by the generated data. It is defined as the concatenation of binary mask vectors for all discrete columns in the dataset. Each binary mask vector corresponds to a single discrete column and has the same length as the number of distinct categories in the column. To create the conditional vector, we initialize all binary mask vectors with zeros. We then set the element of the binary mask vector corresponding to the desired category in the chosen discrete column to 1.

For example, suppose we have three discrete columns in our dataset, with

10, 15, and 20 categories, respectively. The dimension of the conditional vector would be 45, which is the sum of the number of categories in each column. Each binary mask vector would have a length corresponding to the number of categories in its corresponding column. To encode the condition that the first column should have a value of the sixth category, we would set the sixth element of the first binary mask vector to 1. If we also wanted to encode the condition that the second column should have a value of the eleventh category, we would set the eleventh element of the second binary mask vector to 1. The rest of the elements in the binary mask vectors would remain zero.

In the next step, to ensure that all categorical attributes are represented in the training process, a *training-by-sampling* technique is used. This involves selecting one of the categorical attributes at random and computing a probability mass function (PMF) based on the frequency of values in the training data. The generator then samples from this PMF to select a value for the attribute and reconstructs the conditional vector by concatenating all categorical attributes. The detailed steps of *training-by-sampling* technique are as follows.

Assume that we have three discrete columns in our dataset, and let n_1, n_2, and n_3 be the number of categories in the three discrete columns, respectively. We define the interval matrix, $category\_prob = (d_{i,j})$, as follows:

$$d_{i,j} = \begin{cases} \dfrac{\text{number of occurrences of category } j \text{ in discrete column } i}{\text{number of rows in the data frame}}, & \text{if } j < n_i \\ 0, & \text{otherwise} \end{cases}$$

Here, $i \in \{1, 2, 3\}$ and $j \in \{1, 2, \ldots, \max(n_1, n_2, n_3)\}$. Note that the *category_prob* matrix contains the probabilities of each category in the discrete columns, computed as the frequency of occurrence of each category in a discrete column divided by the total number of rows in the dataframe. If a category does not exist in a particular column, its corresponding probability is set to zero. The purpose of this matrix is to efficiently sample the conditional vector during training. The procedure of sampling conditional vector from the *category_prob* matrix (training_by_sampling technique) is as follows:

1. **Random Column Selection**: Utilize 'np.random.choice' to randomly select one of the discrete columns.

2. **Cumulative Probability Calculation**: For the chosen discrete column indexed as 'i', extract its corresponding row from the category_prob matrix and compute the cumulative probabilities, denoted as *cum_probs*.

3. **Random Number Generation**: Employ 'np.random.rand()' to generate a random number, uniformly distributed between 0 and 1, to sample a fresh value for the selected discrete column.

4. **Index Identification**: Identify the index of the first element in *cum_probs* that is equal to or exceeds the random number.

5. **Conditional Vector Adjustment**: Set the identified element of the conditional vector to 1 and all other elements to 0, ensuring the generated sample adheres to the condition for the chosen discrete column.

In the training process, at each epoch, a batch of conditional vectors is produced, which is concatenated with a noise vector to generate synthetic data (i.e., fake data) using the generator. The generated synthetic data is then concatenated with the conditional vector and fed into the discriminator, which aims to distinguish between the real and synthetic data. On the other hand, the conditional vector is also used to filter the real training data that satisfies the given conditioning factors. A permutation of the conditional vector is then concatenated with the corresponding real data and fed into the discriminator.

1.2.3 Generator and Discriminator Architecture

The CTGAN generator, designed to craft synthetic data, is structured with an aim to understand and reproduce the complex relationships between the columns of a dataset. The generator leverages fully-connected networks. These networks are adept at capturing all potential correlations between columns, ensuring that the relationships between different attributes of the data are maintained in the synthetic samples. In the architecture, both the generator and

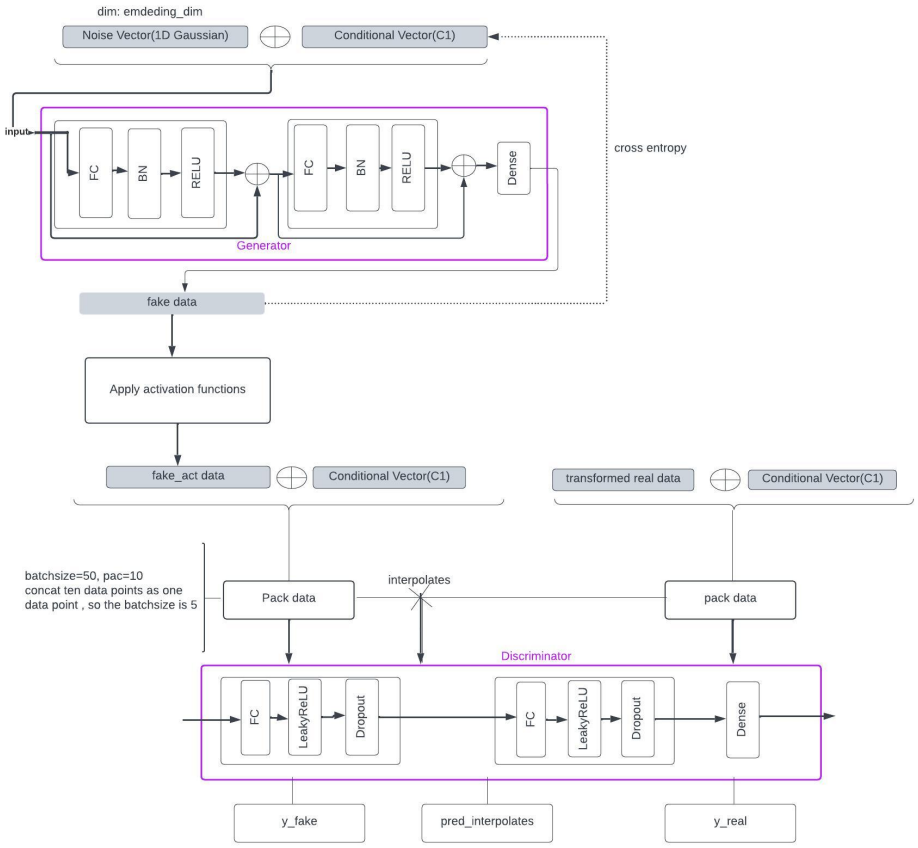

Figure 1.1: Architecture of CTGAN

the critic (or discriminator) of the CTGAN incorporate two fully-connected hidden layers.

To enhance the generator's efficacy, it integrates batch-normalization, which stabilizes and accelerates the learning process by maintaining consistent input means and variances for each layer. Post the inclusion of these layers, the generator introduces non-linearity using the ReLU (Rectified Linear Unit) activation function. Residual connections also used in generator to mitigate the vanishing gradient problem.

For generating scalar values, the generator uses the tanh activation function, which scales the output to lie between -1 and 1. For generating mode indicators and discrete values Gumbel Softmax function is used. This is a smooth approximation to the argmax function, helping in differentiating non-differentiable functions.

In the critic, also referred as the discriminator, there is an incorporation of the Leaky ReLU activation function, known to mitigate the vanishing gradient problem, especially during the earlier phases of training. Additionally, dropout is applied on each hidden layer in the critic. Dropout is a regularization technique, where randomly selected neurons are ignored during training, which helps in preventing overfitting.

1.2.4 Loss Function

1.2.4.1 Discriminator Loss Function

The discriminator loss function is identical to the discriminator loss function for WGAN-GP (Wasserstein GAN-Gradient Penalty):

$$\mathcal{L}_{critic} = -\mathbb{E}_{\mathbf{x}\sim P_r}[D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P_\mathbf{z}}[D(G(\mathbf{z}))] + \lambda \cdot \mathbb{E}_{\hat{\mathbf{x}}\sim P_{\hat{\mathbf{x}}}}[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2] \quad (1.1)$$

Let us break down each term:

1. **Wasserstein Loss:** This term consists of two parts:

   - $-\mathbb{E}_{\mathbf{x}\sim P_r}[D(\mathbf{x})]$: This is the expectation of the critic's scores on real

data. Here, $\mathbf{x}$ represents samples from the real data distribution P_r. The expectation $\mathbb{E}$ calculates the average score the critic assigns to real samples. The negative sign indicates that the critic aims to maximize these scores (since in optimization, we typically minimize loss functions, but here, we want high scores for real data, hence the maximization is turned into minimization by negating the scores).

- $+\mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}[D(G(\mathbf{z}))]$: This is the expectation of the critic's scores on generated (fake) data. Here, $\mathbf{z}$ represents samples from the noise distribution $P_{\mathbf{z}}$, and $G(\mathbf{z})$ represents the generated data samples from the generator function G. The critic tries to minimize these scores.

Combining these, the Wasserstein loss is represented as the difference between the critic's scores for real data and generated data, incentivizing the critic to distinguish effectively between real and fake samples.

2. **Gradient Penalty (GP):** This is the third term in the equation:

   - $\lambda \cdot \mathbb{E}_{\hat{\mathbf{x}}\sim P_{\hat{\mathbf{x}}}}\left[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2\right]$: This term imposes the 1-Lipschitz constraint via the gradient penalty. The term "1-Lipschitz constraint" refers to a mathematical condition that a function (in this case, the critic or discriminator function in a GAN) must satisfy to ensure that its output does not change too abruptly for small changes in its

input. Specifically, a function f satisfies the 1-Lipschitz condition if, for any two inputs x_1 and x_2, the change in the function's output is no greater than the change in the inputs, scaled by a constant (in this case, 1):

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2| \tag{1.2}$$

This condition is crucial for the stability of WGAN training because it prevents the critic's scores from varying too wildly, which can lead to erratic updates of the generator. The gradient penalty method is designed to enforce this 1-Lipschitz condition indirectly. The method involves creating interpolated samples $\hat{\mathbf{x}}$ that lie between real data points and generated data points. This is typically done by mixing real and generated samples using a random weight. These interpolated points help to smoothly bridge the gap between the real and synthetic data distributions. For each of these interpolated samples, we compute the gradient of the critic's score with respect to the sample, denoted as $\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})$. This gradient represents how much the critic's evaluation changes around that sample. We then calculate the norm (or length) of this gradient. In the ideal case, where the critic perfectly satisfies the 1-Lipschitz condition, the norm of this gradient should be exactly 1. This means that

the critic's score should change at most linearly with changes in the input. The gradient penalty term, $(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2$, penalizes any deviation from this ideal norm. If the norm is greater than 1, indicating too steep a change (too sensitive a critic), or if it is less than 1 (indicating too weak a critic), the penalty increases. By incorporating this penalty into the overall loss function, the training process nudges the critic to adjust its parameters to maintain the norm of the gradient close to 1 across its decision boundary. This enforcement ensures that the critic's function remains smooth and well-behaved, which in turn stabilizes the GAN's training process.

In essence, the gradient penalty serves as a regulatory mechanism, ensuring the critic's responses remain consistent and predictable across its input space, thereby enforcing the 1-Lipschitz condition without having to explicitly limit the parameters of the critic. This leads to more stable and reliable training of generative adversarial networks, especially in the context of WGAN-GP.

1.2.4.2 Generator Loss Function

For the generator loss function, in addition to the discriminator which provides feedback to the generator, the cross-entropy between the given conditional vector and the generated output classes is incorporated in the generator loss

function in CTGAN. The aim of this term is to minimize the difference between the generated data and the real data in terms of the distribution of discrete columns. Let us explain how to calculate this loss with an example. Assume that we are working with a dataset that has three discrete columns - d1, d2, and d3. These columns have 14, 39, and 21 categories respectively. To calculate the loss, first the generator is conditioned on a batch of conditional vectors, which provide information about the desired distribution of the discrete columns. Assume that a batch of 5 conditional vectors is sampled: (d3, cat3), (d2, cat19), (d2, cat15), (d2, cat26), and (d3, cat4). Here (di, catj) indicates that we are conditioning on the j-th category of the i-th discrete column. To generate fake data, the batch of conditional vectors is concatenated with a batch of noise vectors and is fed into the generator. Then the portion of the fake data that corresponds to the third discrete column (d3) is extracted, which results in a vector of size 21 containing logits. This vector is compared to the true distribution specified by the conditional vector, which is a one-hot encoded vector containing 21 elements where the third element is set to 1. For each element in the batch, the cross entropy loss between logits and true labels (distribution) is computed and the average is taken to arrive at the generator loss for this batch of data.

$$\mathcal{L}_{generator} = -\mathbb{E}_{\mathbf{z}\sim P_z}[D(G(\mathbf{z}))] + CrossEntropy \tag{1.3}$$

1.3 Banksformer

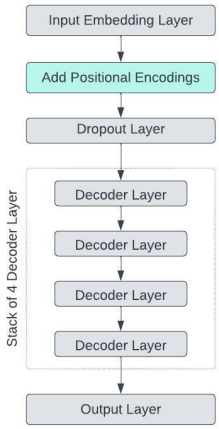

Figure 1.2: Overall Structure of Banksformer

Figure 1.2 illustrates the architecture of Banksformer, comprising three primary components. The input layer processes batches of multivariate transaction sequences, converting them into sequences with varied feature dimensions. Subsequently, the embedded sequences undergo processing by a stack of four identical decoder layers. Finally, the output layer employs a conditional generation mechanism to sequentially produce each feature of the sequence, conditioning each subsequent feature on all preceding ones. The subsequent sections provide detailed explanations of each layer within the Banksformer architecture.

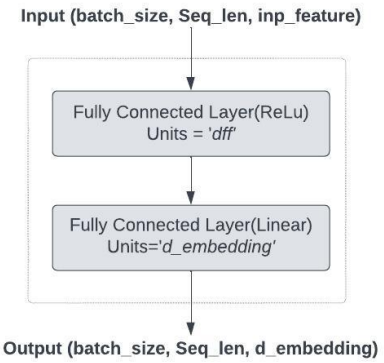

Figure 1.3: Architecture of the Input Embedding Layer

1.3.1 Input Embedding Layer

This layer, depicted in Figure 1.3, is designed to transform input data of dimension *'inp_feature'* into a representation of dimension *'d_embedding'*, which is then consistently used throughout the decoder stack. The Input Embedding Layer comprises two fully connected layers. The first layer outputs a representation with a dimensionality specified by *'dff'*. The incorporation of the ReLU activation function in this layer is pivotal for introducing non-linearity, enabling the layer to capture complex patterns in the data. Subsequent to this initial transformation, the second dense layer linearly projects the output of the first layer into a representation space of the dimension *'d_embedding'*.

1.3.2 Positional Encoding

Positional Encodings (PEs) are indeed a crucial component in the architecture of transformers, addressing one of their fundamental challenges. Since transformers process input sequences in parallel rather than sequentially, they initially lack any means of understanding the order of elements in a sequence. This is unlike traditional recurrent neural networks (RNNs) or Long Short-Term Memory networks (LSTMs), which inherently process data in a sequential manner and thus maintain an understanding of order.

Positional Encodings solve this problem by providing additional information to the model that helps it understand the position or order of each element in the sequence. These encodings are vectors that are added to each element of the sequence. These vectors follow a specific pattern which helps the model determine the position of each input vector in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they are projected to Q/K/V vectors and during dot-product attention.

There are different ways to generate these positional encodings. One common method, used in the original Transformer model [12], involves using sine and cosine functions of different frequencies:

For each position p and each dimension i of the encoding, the value of the

encoding at that position and dimension is given by:

$$\mathrm{PE}_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d_{\mathrm{model}}}}\right)$$

$$\mathrm{PE}_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d_{\mathrm{model}}}}\right)$$

In these formulas:

- PE_(p, i) represents the positional encoding at position p and dimension i.
- p is the position in the sequence (timestep).
- i is the dimension within the positional encoding vector.
- d_model is the dimensionality of the model's embeddings.

The positional encoding vector is a vector of shape (1, Seq_len, d_model = d_embedding) that is added to the output of the Input Embedding Layer. Then the resulting vector undergoes a dropout layer, where the rate of dropout is determined by the parameter 'rate'.

1.3.3 Decoder Layer

This essential layer, illustrated in Figure 1.4, encompasses multiple components. It begins with a Masked Multi-Head Self-Attention mechanism, as elaborated in subsection 1.3.3.1. This layer allows the decoder to focus on relevant parts of the input sequence while ensuring the autoregressive property

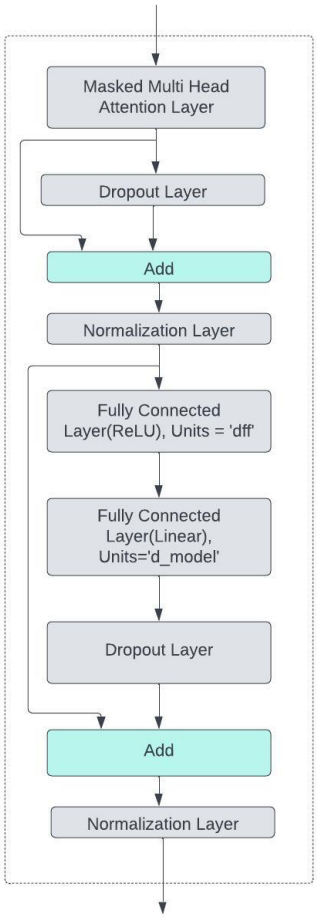

Figure 1.4: Architecture of the Decoder Layer

by excluding future elements from the prediction process. Following this attention phase, a residual connection combines the attention output with its initial input, which aids in maintaining a steady flow of gradients and reduces the risk of diminishing gradients. Layer normalization then follows, contributing to a more stable training process.

Afterwards, the processed data undergoes two sequential transformations. The initial transformation applies a ReLU activation function to introduce

non-linearity, while the second transformation is a linear transformation. This sequence is completed with another set of residual connection and normalization, further enhancing the network's learning capabilities.

To prevent overfitting, dropout [49] is strategically applied after the self-attention mechanism and again following the linear transformations. This technique randomly deactivates certain neurons during training, ensuring the model generalizes well to new, unseen data.

1.3.3.1 Masked Multi-Head Attention Layer

The journey of data through the Masked Multi-Head Attention mechanism, depicted in Figure 1.5, encompasses several critical steps, as detailed below:

1. **Input Projection:** First, the input sequence is converted into query (Q), key (K), and value (V) vectors through three separate linear layers, each with its distinct weights: $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$.

2. **Self-Attention Score Calculation:** This step involves taking the dot product of the query (Q) with all keys (K). It is a way to measure how much each element in the input sequence is related to the current element being focused on, resulting in a seq_len × seq_len matrix.

3. **Score Scaling and Masking:**

    - *Score Scaling:* To prevent excessively large values that could impair

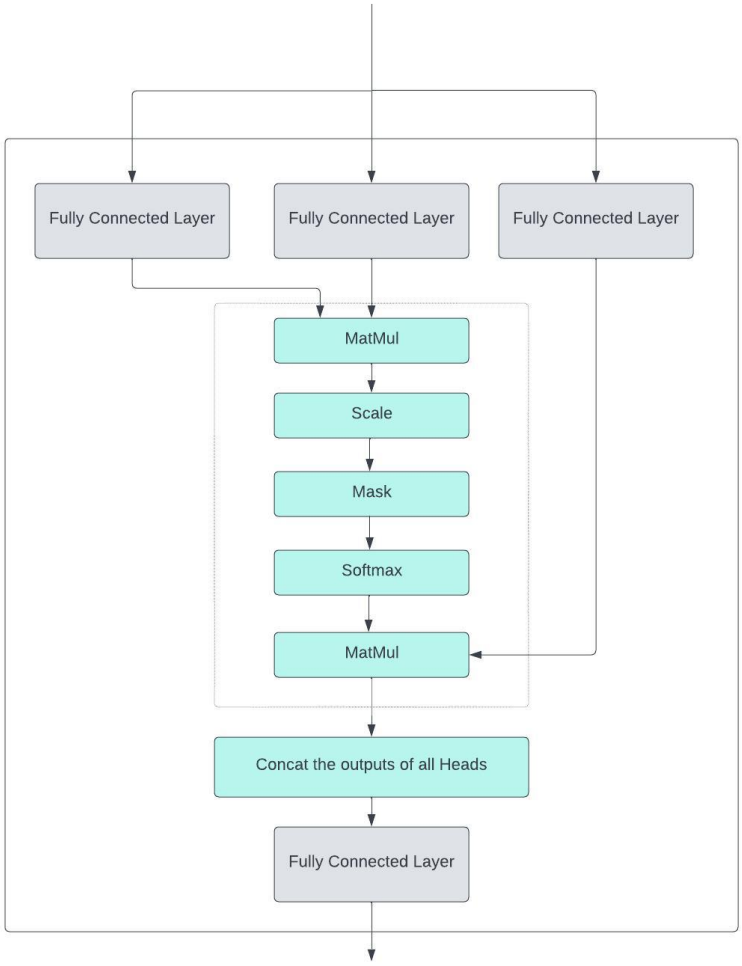

Figure 1.5: Architecture of the Masked Multi-Head Attention Layer

gradient stability, scores are scaled down by the square root of the dimension of the keys ($\sqrt{d_k}$).

- *Combined Mask Application:* At this stage, the combined mask, which merges the look-ahead mask and the padding mask, is applied. In this mask, positions requiring masking are assigned a 1, while unmasked positions are assigned a 0. The combined mask is

then multiplied by a large negative number (effectively applying a large penalty) and added to the scaled self-attention scores. This ensures that, after applying the softmax function, positions meant to be masked will have attention scores close to zero, effectively ignoring these positions during attention calculation.

- *Look-Ahead Mask:* The look-ahead mask ensures that the prediction for a particular position in a sequence does not depend on the future elements in the sequence. The look-ahead mask is a triangular matrix where the upper part (above the main diagonal) is filled with ones, and the lower part (including the diagonal) is filled with zeros. This structure allows the model to only attend to earlier positions in the sequence.
- *Padding Mask:* Padding masks are used to avoid the model paying attention to padding elements. Padding is often used in sequences to make them of uniform length. The padding mask is created based on the actual sequence. It marks the positions of padding elements (usually zeros in the sequence) and ensures that the model does not treat these positions as meaningful.

The combined mask ensures two things: first, the decoder does not peek at future positions in the sequence (thanks to the look-ahead mask), and second, it ignores the padding tokens (thanks to the

padding mask). This combined mask is applied to the self-attention layer in the decoder, ensuring the attention mechanism adheres to both constraints.

4. **Softmax Normalization:** Following the application of the combined mask, the modified attention scores are normalized via softmax, so they are all positive and add up to 1. The resulting matrix is the *attention weights* that can be seen as a percentage of total focus that is given to an element in the sequence, when encoding the current element.

5. **Output Generation:** In this step, each value vector (in matrix V) is multiplied by its softmax score (in attention weights matrix). This keeps the original value intact, but scales the overall vector in line with its relative importance of the current element in the sequence. Finally, all of the scaled values are summed together to produce the encoding of the current element in the sequence.

6. **Multi-Head Division:** The model diversifies the context capture by splitting Q, K, and V into h separate attention heads, each operating within a reduced dimensional space, $\frac{d_{\text{model}}}{h}$, enabling parallel processing of multiple representation subspaces.

All of the above steps can be summerized in this formula:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \text{Combined Mask} * \text{large negative value}\right)V$$

1.3.4 Output Layer

In the Banksformer architecture, the output module comprises several dense layers, each tasked with producing a specific feature. It employs a conditional generation technique, where each subsequent output is sequentially generated, influenced by both the preceding sequence elements and the actual values of previously generated features of the current element.

1.3.5 Training Procedure

In preparing the training data for the Banksformer model, sequences of data are first organized and then enhanced by appending an attribute row at the start of each sequence. This attribute row, specific to Banksformer, is created by duplicating the 'age' attribute of the customer across every feature in the sequence. Consequently, the training data's dimensions are reshaped to (batch_size, seq_len + 1, inp_features). The initial seq_len elements in each sequence are employed for training. The Banksformer model operates by taking the i-th element of a sequence as input and aiming to predict the subsequent (i+1)-th element. Therefore, the training goal is to minimize the discrepancy between the Banksformer's predictions, based on the i-th element as input,

and the actual (i+1)-th element in the sequence.

Loss Function

In the training loss function for Banksformer [15], each piece of transaction information is considered an individual component, with their individual losses summed and assigned unique weights to reflect their relative importance in the model's overall performance. For date-related features, except for the continuous time delta (td), they are handled as categorical variables. The Banksformer model generates outputs for these features in the form of probability distributions across various possible values, utilizing categorical cross-entropy for their evaluation. In contrast, for continuous features including time delta and transaction amount, the loss function is based on the natural logarithm of the probability density function (PDF) of a normal (Gaussian) distribution. This is a common loss component in machine learning, particularly in probabilistic models like Variational Autoencoders (VAEs). The model predicts the mean and standard deviation, which are then used to estimate the likelihood of the real data. The objective is to maximize this likelihood, hence, the negative value of this probability is incorporated into the loss function.

Let us explain the mathematical formula of this loss function. Given:

- x: The real data sample for which the probability is being evaluated.
- μ: The predicted mean of the distribution.

- σ^2: The predicted variance of the distribution.

The probability density function (PDF) for a normal distribution is given by:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

When working with loss functions in machine learning, we often deal with the logarithm of the probability (log-probability), as it provides numerical stability and simplicity in computation. The log-probability of a real sample x from a normal distribution with predicted mean μ and variance σ^2 is given by:

$$\log p(x|\mu, \sigma^2) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(x-\mu)^2}{2\sigma^2}$$

Breaking it down:

- The term $-\frac{1}{2}\log(2\pi\sigma^2)$ comes from the logarithm of the normalization factor of the Gaussian distribution.

- The term $-\frac{(x-\mu)^2}{2\sigma^2}$ results from the exponential part of the Gaussian distribution, reflecting how likely (or unlikely) the real sample x is under the distribution defined by predicted μ and σ^2.

Chapter 2

Methodology

2.1 Dataset

In this thesis we utilize a dataset comprising genuine banking transactions that took place in the Czech Republic during the 1990s. The dataset encompasses over 1 million transaction records from 4500 individual accounts, spanning a period of five years. Each transaction entry includes information such as the transaction's monetary value, multiple categorical codes describing the transaction type, and a timestamp indicating when the transaction occurred.

The raw dataset consists of the following columns:

- **account_id**: Unique identifier for accounts.
- **date**: Date of the transaction.
- **amount**: Transaction amount.

- **age**: Age of the account holder.
- **type**: refers to the basic nature of the transaction (e.g., Credit or Debit).
- **operation**: describes the specific kind of transaction (e.g., Cash withdrawal, Bank transfer).
- **k_symbol**: is a categorization code that provides additional context to the transaction (e.g., Interest credited, Household payment).

Preprocessing Steps

Drawing inspiration from the data preprocessing approach employed in Banksformer [15], we adopt a similar methodology. Below is the detail of how we generate new features:

- **tcode (Transaction Code)**
  - **Original Features:** *type, operation*, and *k_symbol*
  - **Description:** The *tcode* is a composite feature that encapsulates several categorical aspects of a transaction. It is created by concatenating three original columns from the dataset: *type, operation*, and *k_symbol*. For example, a transaction with *type* as "CREDIT", *operation* as "CASH WITHDRAWAL", and *k_symbol* as "HOUSEHOLD" would result in a tcode like "CREDIT__CASH WITHDRAWAL__HOUSEHOLD".

- **month, dow (Day of Week), day (day of month), dtme (days till month ends):**
  - Extracted from: date
  - Description: For the date "15th April 1995", the month feature is 4 (April), the day is 15, the day of the week (DOW) is 6 (Saturday), and there are 15 days remaining until the end of the month (DTME).
- **td (time delta between transactions):**
  - Description: This feature represents the number of days between consecutive transactions for each account. It is derived by calculating the difference in days between the current transaction date and the previous transaction date for the same account.

2.2 Stacked LSTM Autoencoder

We implemented a Stacked LSTM Autoencoder [47, 48] as a baseline to evaluate the efficiency of recurrent neural networks (RNNs) when integrated with the *conditional training and generation* approach, as well as the *Date Mechanism*, both of which are inspired by Banksformer [15].

As depicted in Figure 2.1, the model's structure includes an encoder and a decoder, each consisting of two LSTM layers. The encoder's role is to trans-

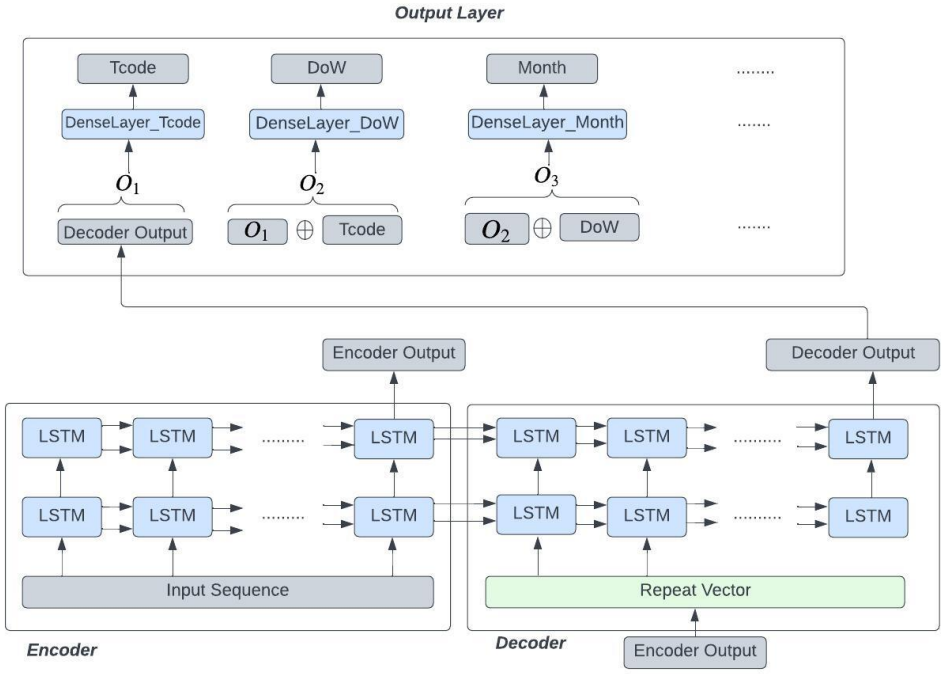

Figure 2.1: Stacked LSTM Autoencoder

form input sequences into a context vector, specifically capturing the hidden state from the final timestep of its second LSTM layer. This context vector is then replicated across timesteps via a repeat vector layer and fed into the decoder. The decoder's first LSTM layer receives this sequence, initializing its hidden and cell states with the final timestep states from the encoder's first LSTM layer. The second LSTM layer of the decoder processes the output of its first layer, with initial states set by the final timestep of the encoder's second layer. The decoder's output then proceeds through several dense layers, each responsible for predicting a distinct feature in a manner that depends on the previously predicted features. During the training phase, the values for fea-

ture conditioning are sourced from the actual dataset, while during generation, they are derived from the features predicted earlier in the sequence.

Feature Encoding. Date and categorical features are encoded as per the methodology outlined in section 2.3.2. In contrast, continuous features undergo normalization by scaling with their standard deviation.

Feature Decoding. For date and categorical features, we apply the softmax function in the dense layers, using categorical cross-entropy as the loss function. The dimensionality of the output vector for categorical variables aligns with the number of categories, and for date features, it matches the temporal period's units (e.g., 7 for day of the week). Continuous features are handled by a ReLU-activated dense layer predicting both mean and standard deviation. Then the loss function for these features is based on the likelihood of a normal distribution (explained in section 1.3.5), which measures how well the model's predictions fit the real data.

Date Sampling. The output vectors of dense layers for date features are directly used in Equation 2.6. Additionally, the predicted mean and standard deviation of td is used to construct the probability of time deltas in Equation 2.6.

2.3 BankGAN

BankGAN, developed for generating bank transaction sequences, maintains CTGAN's foundational architecture and training methodology [32]. Its distinct innovation lies in the strategic integration of a decoder-only transformer for pre-generating transaction code sequences. This approach was chosen due to the success of transformers in generating coherent text sequences, which closely resemble the sequential dependencies found in financial transactions. Just as the likelihood of the next word in a sentence depends heavily on its preceding context, each transaction code in a sequence depends on the historical pattern of transactions, necessitating a model that can effectively capture these long-term dependencies.

The use of a transformer allows BankGAN to address the limitations of traditional time-series GAN models like TimeGAN and DoppleGAN, which previous research [15] showed that they fail to generate synthetic financial transactions with realistic and coherent patterns. By leveraging the transformer's self-attention mechanisms, BankGAN effectively understands and generates transaction code sequences where each element is contextually tied to its predecessors.

Moreover, BankGAN features strategic sequential sampling of conditional vectors and an adapted Banksformer's *date mechanism* [15], with modifications for feature-to-probability conversion. In contrast to Banksformer, where the

output of date-related features is represented as probabilities derived from a softmax function, BankGAN employs a different approach. In BankGAN, the output for date-related features is a point in a two-dimensional space, generated through a tanh activation function. then the probabilities of date related features are evaluated based on its distance from the reference points within a clock-dimensional space, allowing for a more nuanced interpretation and utilization of temporal data.

BankGAN comprises two blocks: Generator and Discriminator. The architecture of the generator and discriminator in BankGAN is identical to that of CTGAN [32]. In the generator, two key layers are employed. Each of these layers is a fully connected network that incorporates batch normalization and ReLU activation. The input of each layer is concatenated with its output, which subsequently feeds into the next layer. The final step involves a fully connected layer that maps the output to the dimensionality of the encoded real data. The discriminator consists of two similar layers, each performing a linear transformation followed by leaky ReLU activation and dropout. The final layer is a dense layer that reduces the output to a singular value, indicative of the input data's authenticity. The model is trained using WGAN loss with gradient penalty.

2.3.1 Generator's Input

The generator requires a noise vector plus a conditional vector, which is used for conditioning on *transaction codes (Tcodes)*; the singular categorical attribute in our dataset. Similar to CTGAN [32], a *training-by-sampling* technique is used for sampling transaction codes during training. This technique is used to ensure that all Tcode categories are represented in the training process based on their frequencies in the training data. Mirroring CTGAN's strategy [32], an extra term is added to the generator's loss function. This element measures the disparity between the prescribed condition, in this case, the *Tcode*, and the *Tcode* generated by BankGAN. This adjustment is crucial for guiding the generator to produce the exact *Tcode* as dictated by the given condition. This capability facilitates the use of *sequential conditional vectors* in the generation process, as detailed in section 2.3.3.

2.3.2 Discriminator's Input

In BankGAN, mirroring the approach of CTGAN [32], each row of real data is processed variable by variable, with each feature undergoing independent encoding. These encoded features are then concatenated into a single vector, which serves as the input for the discriminator. The output from the generator is adjusted using specific activation functions, namely the tanh and softmax functions, to ensure compatibility with the encoded real data, effectively gen-

erating fake data samples for the discriminator's evaluation. The activation functions employed during training are the same as those used in the generation phase, as explained in subsection 2.3.3. The subsequent subsections will elaborate on the encoding techniques employed for date, continuous, and categorical variables; the three distinct types of features identified in BankGAN.

Date Feature Encoding. For date-related variables, three encoding techniques are explored: One-hot encoding, Clock encoding, and Radial Basis Function (RBF) encoding.

- *One-hot Encoding:* For a temporal period, such as days of the week, with n units (e.g., $n = 7$), each unit t (where $t \in \{0, 1, \ldots, n-1\}$) is encoded into an n-dimensional binary vector, with a 1 in the position of t and 0 for all others. This encoding approach is simple and clear but doesn't convey the continuity between units and struggles with large temporal ranges due to increasing dimensions.

- *Clock Encoding:* For each temporal unit t, the clock encoding $\mathbf{C}(t)$ maps t to a two-dimensional space [15]:

$$\mathbf{C}(t) = \left[\sin\left(\frac{2\pi t}{n}\right), \cos\left(\frac{2\pi t}{n}\right)\right] \tag{2.1}$$

  This encoding captures the inherent cyclical nature of time, preserving the relationship between the end and start of a cycle, which is particularly useful for patterns recurring over regular intervals.

- *RBF Encoding:* The RBF encoding for each temporal unit t into a two-dimensional space is given by[1]:

$$\mathbf{R}(t) = \left[e^{-\left(\min\left(\left|\frac{t}{n-1}\right|, 1-\left|\frac{t}{n-1}\right|\right)\right)^2}, e^{-\left(\min\left(\left|\frac{t}{n-1}-0.5\right|, 1-\left|\frac{t}{n-1}-0.5\right|\right)\right)^2} \right] \tag{2.2}$$

  This transformation allows the model to capture the cyclical relationship of the feature by mapping it onto two dimensions, where each dimension reflects the closeness to a point in the cycle, represented by the two basis functions.

Continuous and Categorical Feature Encoding. Continuous variables use *mode-specific normalization* [32] from a Gaussian mixture model, representing each value as a value-mode pair.

$$G(\tau) = \left[\frac{\tau - \mu_i}{4\sigma_i}, \text{one-hot}(i) \right] \quad \text{where } i = \arg\max_j \rho_j(\tau) \tag{2.3}$$

where $\rho_j(\tau)$ represents the probability density of τ for the j-th mode of a Gaussian mixture model, each with its own mean μ_j and standard deviation σ_j. This encoding ensures that each continuous value is normalized relative to the mode it most closely aligns with.

In the CTGAN model [32], the 1/4 scaling factor applied during the normalization of continuous variables serves several potential purposes that align with general practices in neural network training. This factor reduces each

[1] https://scikit-lego.netlify.app/_modules/sklego/preprocessing/repeatingbasis

data point's deviation to a quarter of its mode's standard deviation, thereby compressing the input data to within four standard deviations from the mean. This compression likely aims to limit the range of input values to a narrower, more manageable scale, which is crucial for stabilizing the training process of the generative adversarial network. Such scaling can be particularly beneficial in GANs, as it helps in mitigating the impact of outliers and ensuring that the generator and discriminator networks focus on the most statistically significant aspects of the data distribution. Although the specific rationale for choosing the 1/4 factor is not detailed in the original CTGAN paper [32], it can be inferred that this normalization technique aids in enhancing model convergence and robustness by providing a consistent scale across different input features, thus facilitating more effective learning dynamics within the GAN framework.

In addition to continuous feature encoding, categorical features undergo one-hot encoding.

2.3.3 Synthetic Data Generation

The synthetic data generation process, illustrated in Figure 2.2, begins with the generation of transaction code, Tcode, sequences using the transformer model described later in this section. Each Tcode sequence is uniquely associated with a specific bank customer's account. To generate synthetic data tailored to an individual customer, these Tcode sequences are fed into the

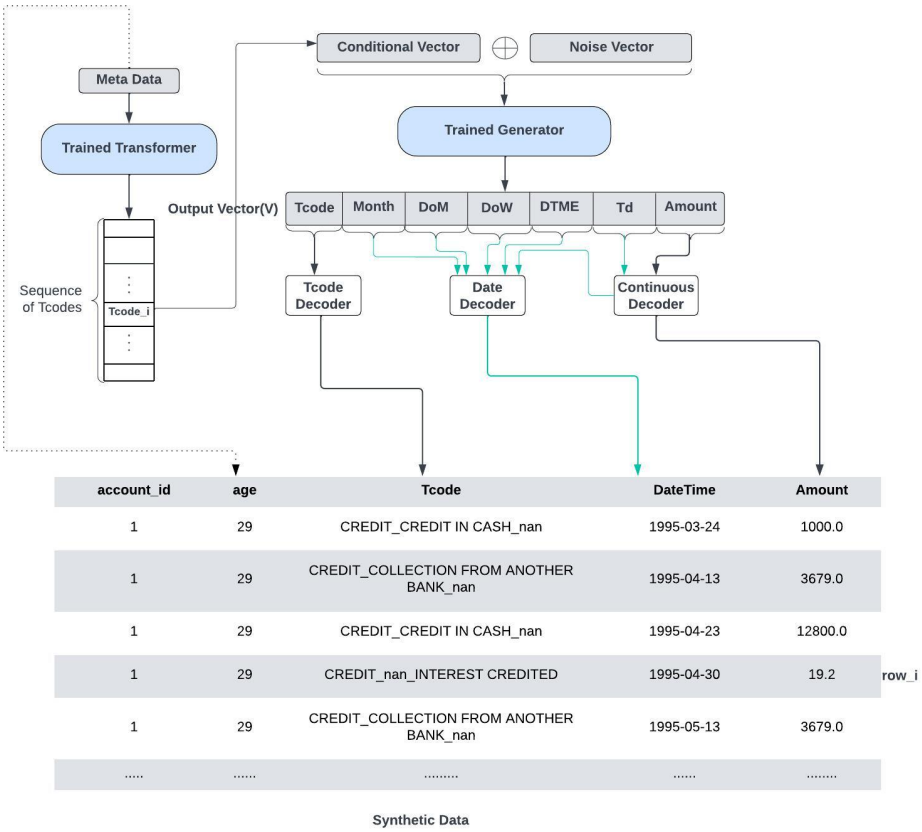

Figure 2.2: Overview of the BankGAN Synthetic Data Generation Workflow. The process begins with a trained Transformer that generates a sequence of transaction codes, $Tcodes$. These are transformed into a conditional vector that, when combined with a noise vector, feeds into the Generator. The output vector, V, is then decoded into transaction features, yielding a synthetic dataset that mimics real transactional patterns.

trained generator in a sequential manner. They are combined with a noise vector, sampled from a normal distribution. The output from the generator is a multi-feature vector, denoted as V. In this vector, distinct segments are responsible for representing different data features. The raw outputs from these segments are subsequently decoded through tailored processes for each seg-

ment to construct the final synthetic dataset. The following is an explanation of how the various segments of the vector, V, are decoded to represent distinct data features in the synthetic dataset:

Continuous Decoder. The decoding process for the continuous variables is the inverse of its encoding method as described in Equation 2.3. The Gaussian distribution index i is identified by $i = \arg\max(\text{Softmax}(V_{\text{continuous}}[1:]))$. The generated value is then $\tanh(V_{\text{continuous}}[0]) \times 4 \times \sigma_i + \mu_i$, where μ_i and σ_i are the mean and standard deviation of the i-th Gaussian mode.

Tcode Decoder. The decoding process for the *transaction code (Tcode)* is straightforward. First, the index i is determined by $i = \arg\max(\text{Softmax}(V_{\text{Tcode}}))$. Then, the corresponding *Tcode* is extracted based on its original encoding method.

Date Decoder. For a *date* feature represented by V_{date} in the output vector, we transform it into a probability vector p_{date} of dimension n, where n corresponds to the total units in the date feature (e.g., $n = 12$ for months). The transformation method varies depending on which encoding technique was used during training. For one-hot encoded *date* features, $p_{\text{date}} = \text{Softmax}(V_{\text{date}})$. For cyclical encoding, p_{date} is computed as:

$$p_{\text{date}} = \left[\frac{\frac{1}{d(t)}}{\sum_{u=0}^{n-1} \frac{1}{d(u)}} \right]_{t=0}^{n-1} \tag{2.4}$$

where $d(t)$ is the squared distance between the cyclical encoding of unit t and point pt, with $pt = \tanh(V_{\text{date}})$, and $\epsilon = 0.01$ is a small constant for stability.

The distance $d(t)$ is defined as:

$$d(t) = \sum_{i=1}^{2} \left(\mathbf{E}(t)_i - pt_i + \epsilon\right)^2 \tag{2.5}$$

with $\mathbf{E}(t) = \mathbf{C}(t)$ from Equation 2.1 for clock encoding and $\mathbf{E}(t) = \mathbf{R}(t)$ from Equation 2.2 for RBF encoding. After calculating probability distributions for each date component, we apply the Banksformer *date mechanism* [15] to determine the probability of transaction dates. This is achieved with the following formula [15]:

$$p(\text{transaction date}) \propto p_{\text{month}}(m) \times p_{\text{DoM}}(d) \times p_{\text{DoW}}(w) \times p_{\text{DTME}}(e) \times p_{\text{td}}(td) \tag{2.6}$$

where m, d, w, and e represent the values for month, day, day of the week, and days to month end, respectively, for the transaction date. p_{td} is a normal distribution with its mean as the decoded V_{td} from the *Continuous Decoder* and a fixed standard deviation. $p(\text{transaction date})$ is computed for 'max days' starting from the last transaction date, and the date with the highest probability is selected for generating the transaction date.

Tcode Sequence generator. It is a decoder-only transformer, where a fully connected embedding layer first transforms the input features, followed by a positional encoding addition to incorporate sequence order information. The core of the model consists of a decoder stack that applies multiple attention mechanisms. Outputs from the decoder are then passed through a final dense layer. For the generator functionality of this model, drawing inspiration

from Banksformer [15], a specific training and prediction strategy is adopted: Specifically, metadata invariant across the dataset for each customer (e.g., customer's age) is embedded as the initial element of the input sequence, ensuring uniform feature dimensionality. During training, the model is designed to forecast subsequent sequence elements based on prior ones. For prediction, it employs an iterative approach, commencing with the metadata vector and progressively appending the most recent output to the input sequence to achieve the desired length.

Chapter 3

Results

3.1 Hyperparameters

The Banksformer model adheres to the configurations outlined in its originating study, featuring fully connected layers each with 128 units, and employing a dropout rate of 0.1. It is structured with four decoder layers, each equipped with two attention heads. Training procedures specify a batch size of 64, spanning a maximum of 80 epochs, with an early stopping mechanism activated if no improvement is observed over a span of 5 epochs. The optimization of this model is facilitated by the Adam optimizer, utilizing its default parameters.The StackedLSTM model adopts 128 units for its LSTM cells, maintaining the same training regimen in terms of batch size, epoch count, and early stopping criteria. It too leverages the Adam optimizer with default settings,

ensuring consistency and efficiency in the training process of both models.

The BankGAN model introduces a nuanced variation in its transformer configuration for generating Tcode sequences, aligning closely with the Banksformer but adjusting the unit count in fully connected layers to 64, diverging from the latter's 128-unit specification. The fine-tuning of the generator's and discriminator's hyperparameters in the BankGAN model emerged from extensive experimentation aimed at achieving convergence between the Generator and Discriminator. These experiments, detailed in Appendix A, led to a set of optimal hyperparameters for BankGAN convergence. First let us describe the hyperparameters in BankGAN:

- **Embedding Dimension**: This parameter defines the dimensionality of the noise vector that's fed into the generator.

- **Generator Dimension (d1, d2, ...)**: This tuple determines the architecture of the generator's fully connected layers. The length of the tuple indicates how many layers the generator possesses, while each specific value within the tuple denotes the number of neurons or units in that particular layer. By varying these, one can experiment with the capacity and complexity of the generator's neural network.

- **Discriminator Dimension (d1, d2, ...)**: Similar to the generator dimension, this hyperparameter governs the architecture of the discriminator.

- **Gradient Penalty Lambda**: This value represents the weight or importance of the gradient penalty in the discriminator's loss function. The gradient penalty is a technique used to stabilize the training of the GAN, ensuring that the discriminator's gradients are bounded and preventing issues like mode collapse.

- **Discriminator step**: During the training process, the discriminator may undergo multiple training iterations for each update of the generator. This hyperparameter defines the number of training iterations allotted to the discriminator for each generator update. This aspect is crucial as it helps in maintaining the balance between the discriminator and generator during training, ensuring neither overpowers the other.

- **PAC** : In the context of the PAC discriminator, the PAC hyperparameter denotes the number of datapoints grouped together in a single pack. This parameter is crucial for training, influencing the discriminator's ability to effectively differentiate between real and synthetic data by assessing them in pack rather than individually. Adjusting this value can impact the granularity at which the discriminator analyzes data, thereby affecting the training process and the model's ultimate performance.

In the BankGAN model, the setup includes an embedding dimension of 100, with the generator's architecture defined by a dimension sequence of (256, 128), and the discriminator's dimension set as (128, 256). Data points are grouped

into packs of 10 for the discriminator's evaluation. Both the generator and discriminator utilize the Adam optimizer, featuring a learning rate of 0.0002, a decay rate of 0.000001, and beta values set at 0.5 for beta_1 and 0.9 for beta_2. Training parameters are established with a batch size of 700 and a training duration capped at 80 epochs.

3.2 Experimental Analysis

In our experimental analysis, we evaluate the effectiveness of BankGAN using the *Czech dataset* and compare it with the StackedLSTM Autoencoder and Banksformer [15]. All models employ Banksformer's *date mechanism* [15] for generating transaction dates, and are evaluated across three date encoding methods (one-hot, clock, and RBF) as detailed in Section 2.3.2. Additionally, we test the models without the *date mechanism*, focusing on three features: *transaction code (Tcode), transaction amount,* and *Time delta (Td)*, treating the latter as continuous. Synthetic transaction dates are then derived by assigning random start dates to each account and calculating subsequent dates using the generated time deltas.

Experimental Setup All models were implemented using TensorFlow2, with BanksFormer, StackedLSTM Autoencoder, and BankGAN containing 450k, 490k, and 320k trainable parameters, respectively. Experiments were run on a system with a NVIDIA Quadro RTX 8000 GPU and 48 GB of RAM.

Evaluation Metrics. We evaluate the effectiveness of BankGAN in terms of the statistical similarity to the real data. Two metrics are used to quantify the statistical similarity between real and synthetic data:

- The *Jensen-Shannon Divergence (JSD)* quantifies the difference between the probability mass distributions of categorical features belonging to the real and synthetic data. This metric is bounded between 0 and 1.
- The *Wasserstein Distance (WD)* quantifies the earth's moving distance on continuous features between real and synthetic data.

We assess our model's performance by comparing feature distributions in synthetic and real data, focusing on transaction codes (Tcode-JSD), transaction amounts (Amount-WD), and the day of the month on which transactions occur (Day-JSD). These comparisons gauge the models' ability to replicate individual feature distributions independently. To evaluate the modeling of feature interactions within the synthetic data, we examine the joint distribution of transaction codes and transaction days (Tcode-Day-JSD) and analyze the distribution of monthly cash flow using WD (Cash Flow-WD). Monthly cash flow, defined as the difference between credits and debits for a customer in a month, reflects how well the model captures the interplay between transaction type, amount, and timing. Additionally, the WD of time delta distributions (Time delta-WD) helps assess how closely the transaction frequency behaviors of customers in synthetic data mirror those in the real dataset. We further

analyze 3-gram distributions of transaction codes (Tcode-3g-JSD) to evaluate the model's capability in capturing the sequence of transactions.

Method	Date Encoding	Amt WD	CF WD	Td WD	Tcode JSD	Day JSD	Tcode-3g JSD	Tcode-Day JSD
BankGAN	**onehot**	2627	8402	2.7	0.09	0.066	0.25	0.26
	clock	2057	9542	0.73	0.09	0.06	0.25	0.26
	rbf	2173	6332	1.6	0.09	0.09	0.25	0.32
	None	1884	8472	1.01	0.09	0.11	0.25	0.37
StackedLSTM	**onehot**	3769	2500	2.86	0.089	0.13	0.29	0.25
	clock	4277	3674	1.93	0.064	0.13	0.28	0.26
	rbf	4039	2392	1.96	0.12	0.15	0.35	0.30
	None	4376	2285	2.5	0.12	0.14	0.34	0.38
BanksFormer	**onehot**	5916	7162	6.1	0.17	0.028	0.42	0.21
	clock	4236	4338	0.15	0.016	0.010	0.058	0.035
	rbf	3383	3394	0.98	0.021	0.026	0.079	0.064
	None	4931	5015	1.32	0.074	0.11	0.21	0.32

Table 3.1: The performance of BankGAN in comparison with StackedLSTM Autoencoder's and Banksformer's performance, while using different mechanisms to represent date. 'None' means the model only uses time delta as a continuous variable.

3.2.1 Assessing Date Encoding Techniques in Banksformer, BankGAN, and StackedLSTM Models

Our analysis contrasts three date encoding strategies and their impacts on the model performance, revealing notable variations across metrics. As shown in Table 3.1, while using the Banksformer model [15], the clock encoding outperforms in all metrics except for the Amount and Cash Flow. Similarly, BankGAN benefits from clock encoding in all metrics except Cash Flow. In the StackedLSTM, RBF encoding stands out for its proficiency in Cash Flow metric yet falls short in Tcode, Tcode-3g, and Tcode-Day metrics. An analysis between one-hot and clock encoding within StackedLSTM highlights clock encoding's marginal advantage in Time delta, Tcode, and Tcode-3g metrics, while one-hot encoding prevails in Amount, Cash Flow, Day, and Tcode-Day. Across the evaluated models, clock and RBF encodings surpass one-hot encoding in capturing the temporal dynamics of customer transaction behavior, as indicated by Time delta distribution metrics. This highlights the effectiveness of cyclical encoding in capturing temporal patterns. Moreover, clock encoding consistently exceeds RBF encoding in all metrics except for Cash Flow and Amount, establishing it as the superior method for further analysis. Consequently, models utilizing clock encoding are selected for more in-depth evaluation.

Inclusion/Exclusion Date Mechanism. In examining the effectiveness

of the *date mechanism* derived from Banksformer [15], a comparison between the optimal date encoding method for each model and the scenario where the date mechanism is not utilized ('None' in Date encoding column of Table 3.1) reveals distinct patterns across the models. For Banksformer [15], the absence of the date mechanism consistently leads to inferior performance across all evaluated metrics, underscoring the mechanism's crucial role in enhancing model accuracy. For StackedLSTM, excluding the date mechanism generally leads to worse performance compared to the clock encoding, except for Cash Flow metrics. BankGAN demonstrates enhanced performance in metrics like Amount and Cash Flow upon excluding the date mechanism; however, it performs comparably or worse in other metrics. These outcomes suggest that the inclusion of the *date mechanism* significantly contributes to capturing the time patterns, and the overall accuracy of modeling both individual features and their interactions.

In the overall evaluation of models utilizing clock encoding, as referenced in Table 3.1, BankGAN outperforms StackedLSTM and Banksformer [15] in capturing transaction Amount distribution, attributed to its mode specific normalization technique, which contrasts with the single distribution assumption of the other two models. Interestingly, BankGAN outperforms StackedLSTM across nearly all metrics. This underscores the efficacy of incorporating a sequential conditional vector within BankGAN's architecture over the RNN-

based approach. Despite these strengths, BankGAN does not surpass Banksformer [15] across a broad range of metrics.

3.2.2 Comparative Analysis of Banksformer, BankGAN, and StackedLSTM Models

In the evaluation of the three models, those utilizing clock encoding outperformed alternative encoding methods. Consequently, we specifically focused on models incorporating clock encoding for an in-depth analysis. This examination not only compared the models' abilities to generate synthetic data with comprehensive transaction codes but also delved into their ability to replicate recurring versus non-recurring transactions, which constitute approximately 58% and 42% of our dataset, respectively. Transactions were classified as recurring if they occurred regularly each month around the same date, and as non-recurring otherwise. For the purpose of clarity and precision in transaction classification, the following criteria are established:

1. **Temporal Consistency:** A transaction is considered recurring if it occurs within a ± 3 day window around a consistent date each month. This range accommodates slight variations in transaction timing due to weekends, holidays, or banking operations.

2. **Frequency Threshold**: A transaction must be observed for at least six consecutive months to qualify as recurring. This threshold helps in dis-

tinguishing genuinely regular transactions from those that are sporadic or occasional.

Statistical similarities between synthetic and real data were then assessed separately for recurring transactions and non-recurring transactions. This approach aims to pinpoint BankGAN's specific areas of strength and weakness in modeling transaction patterns.

3.2.2.1 Comparison of Univariate Distributions

In this section, we visually compare the performance of three models—BankGAN, StackedLSTM, and Banksformer—across three distinct univariate distributions: transaction log amounts, monthly cash flows, and transaction code (Tcode) distributions, validated using real data. Analyses are conducted for the entire dataset and separately for subsets containing non-recurring and recurring transactions.

Entire Dataset. As depicted in Figure 3.1, while both BankGAN and Banksformer exhibit proficiency in density estimation, BankGAN may have a slight edge in mimicking the actual distribution closely. Although Banksformer also performs well, the visual comparison suggests that BankGAN's density estimation aligns more closely with real data, particularly in capturing the high-density areas. In the monthly cash flow distributions, the Banksformer and StackedLSTM models show a sharp peak that closely mirrors the real

data's distribution, indicating effective simulation of typical cash flow values. BankGAN also replicates the real peak, but with slightly less accuracy compared to Banksformer and StackedLSTM, suggesting a minor deviation in capturing typical monthly cash flows. For the frequency distributions of transaction codes (Tcode), both BankGAN and StackedLSTM achieve moderate accuracy. However, Banksformer demonstrates a better fit for the transaction code (Tcode) distribution compared to the other two models, but still falls short compared to the real data, indicating potential challenges in capturing the full complexity of transaction types.

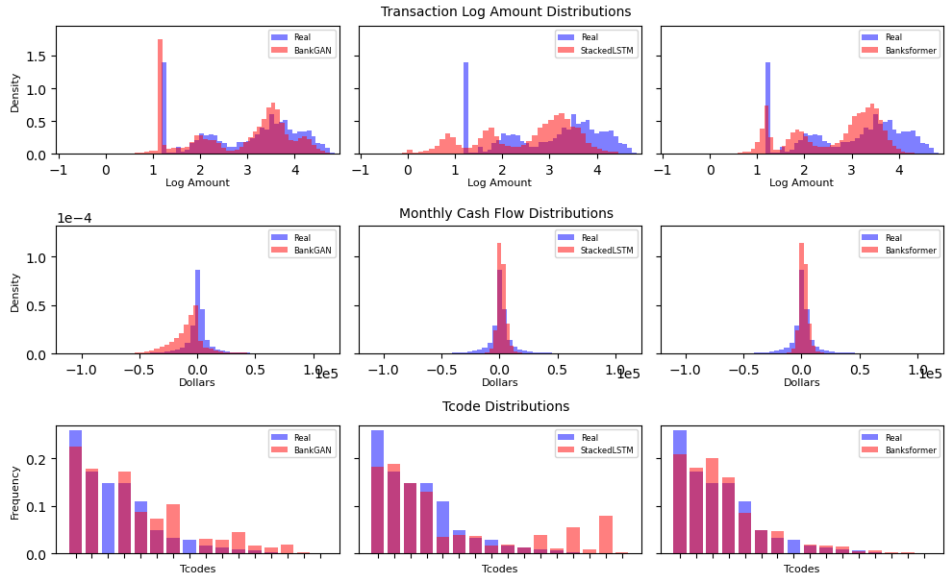

Figure 3.1: Comparison of models for all transactions.

Recurring Transactions. As Figure 3.2 shows, BankGAN excels in replicating the real data's patterns for recurring transactions, in transaction log

amount and monthly cash flow. However, it exhibits a less accurate fit in Tcode distribution, where Banksformer [15] demonstrates the closest resemblance to the actual data. The accurate replication of recurring transactions is crucial for financial models, as these patterns are indicative of stable customer behavior and are essential for risk assessment. This underscores the significance of BankGAN's strengths in modeling these transaction types accurately.

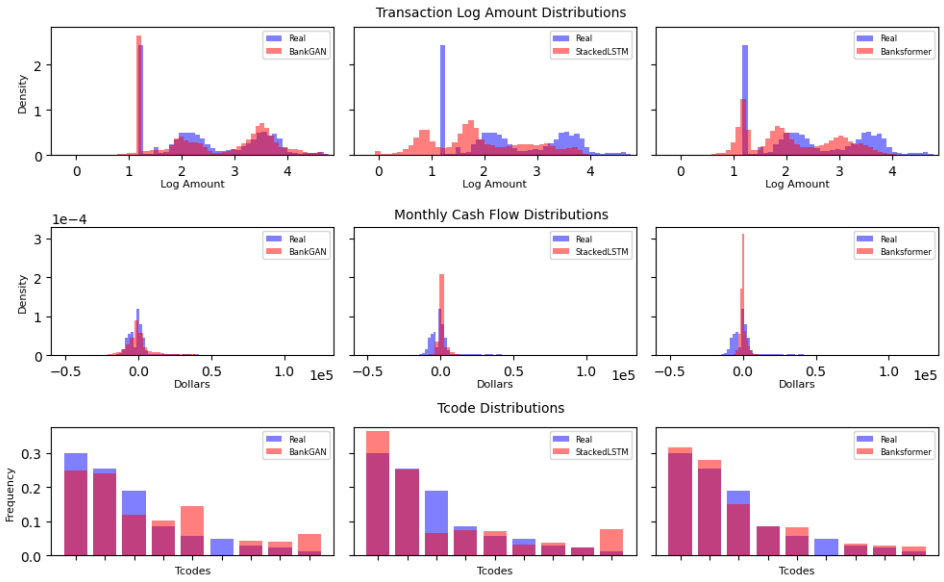

Figure 3.2: Comparison of models for recurring transactions.

Non-recurring Transactions. In the analysis of non-recurring transactions (Figure 3.3), BankGAN's synthetic data generation notably struggles with Tcode distributions. The real data consists predominantly of debit transactions, with a singular credit transaction category evident in the bar plots as the second bar. BankGAN fails to replicate this credit transaction category, resulting in exclusively negative cash flow values in the generated data, as depicted in the cash flow distribution plot. While the peak log amounts are comparably captured by BankGAN, the model introduces a pronounced tail in the distribution that deviates from the actual data.

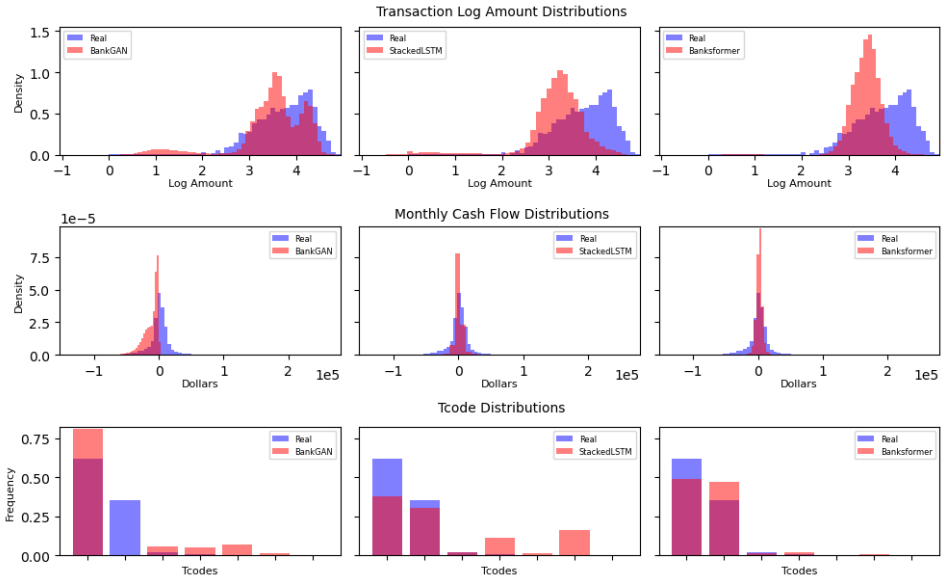

Figure 3.3: Comparison of models for non-recurring transactions.

3.2.2.2 Comparative Analysis of PCA-Transformed Synthetic and Real Datasets

In this section, we compare the distribution of synthetic data generated by three distinct models against real data using Principal Component Analysis (PCA).

Initially, we select relevant features for analysis: transaction codes (Tcodes), transaction amounts, and time deltas between transactions. The next step involves processing the real transaction data to obtain an aggregate view. The transaction data, which is recorded on a per-transaction basis, is aggregated at the account level by grouping the data by account_id. This means that for each unique account, the data is consolidated, and the mean (average) value of each selected feature (Tcodes, transaction amounts, and time deltas) is calculated. This includes transaction codes, which are presumably one-hot encoded, transaction amounts, and time delta. This aggregation step reduces the dataset from a list of individual transactions to a summary of average transaction characteristics for each account. The purpose of this aggregation is to simplify the dataset, making it easier to analyze and interpret by reducing noise and ensuring consistency across accounts. By transforming detailed transaction data into a summary of average characteristics for each account, we can more effectively identify patterns and differences in behavior, enhancing the clarity and effectiveness of the analysis without the overwhelming detail

of individual transactions.

The data is then standardized to ensure a fair comparison during PCA. Standardization adjusts each feature so that it has a mean of zero and a standard deviation of one. This standardization is crucial for PCA, which relies on variance to determine the principal components. Standardizing the data prevents any single feature from disproportionately influencing the results due to its scale.

Following standardization, PCA is applied to the real dataset, yielding a new set of orthogonal variables, the principal components, which systematically capture decreasing proportions of the data's total variance. This transformation not only simplifies the data's complexity by reducing its dimensionality but also retains significant structural information, making it more amenable to visualization and analysis.

The transformation derived from the real data is then consistently applied to various synthetic datasets. This procedure ensures that the real and synthetic datasets are directly comparable, as they share the same scale and dimensional framework.

The last step includes creating visual representations of these datasets in the space defined by the principal components, utilizing kernel density estimation plots for a continuous perspective. This method enables a thorough examination of how well the models mimic actual data structures and trends.

For this purpose, the distribution across the reduced-dimensionality space delineated by the first two principal components (PC1 and PC2) is visualized. To assess which model's output is closest to the real data, we consider various factors within the density plots. First, we examine the shape and topology, evaluating how closely the overall shapes and the arrangements of the contour lines in the generated plots align with those in the 'Real' plot. Next, we assess the density distribution by comparing how well the areas of dense contour lines (indicating higher concentrations of data points) match between the models and the real data in terms of both location and size. Lastly, we look at the spread of the data, which is how well the extent or range covered by the contour lines in the model-generated plots matches that of the real data. By analyzing these aspects, we can determine which model best captures the underlying structure and distribution of the real dataset.

Similar to the previous section, analyses are performed on the entire dataset as well as on specific subsets categorized by non-recurring and recurring transactions.

Entire Dataset. As shown in Figure 3.4, when comparing the models' outputs to the real data in terms of shape and topology, the Banksformer model stands out as having the closest resemblance. It effectively mirrors the real data's complex structure, with multiple peaks and valleys, although it falls short in capturing the isolated data points present in the real dataset.

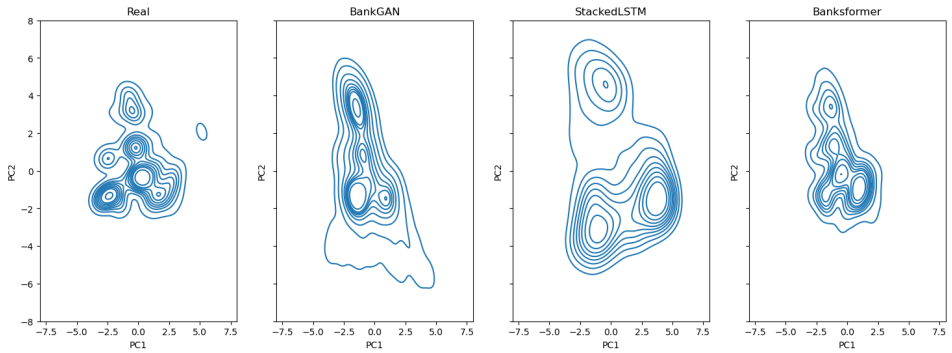

Figure 3.4: PCA visualization of real data and synthetic data generated by the three models.

BankGAN also attempts to mimic the real data's shape but lacks the precision of Banksformer, particularly in replicating the isolated area. StackedLSTM, on the other hand, shows a significant deviation, presenting a more linear and elongated shape that strays far from the intricate design of the real data.

In assessing the density distribution, which examines the concentration and placement of data points, the Banksformer model again proves to be superior. It more accurately reflects the dense regions found in the real data, capturing the essence of the data's clustering. BankGAN provides a fair attempt but does not offer the precision seen in Banksformer. StackedLSTM falls behind in this category as well, with misplaced dense areas and a less accurate portrayal of the real data's clustering.

Regarding the spread of the data, which looks at the overall range and distribution of the points, the Banksformer model again leads in replicating the real data's spread, although it does not perfectly capture the isolated areas seen in the real plot. BankGAN offers a somewhat similar spread but does

not fully align with the real data's nuances. StackedLSTM displays a broader and more linear spread, diverging significantly from the compact and varied nature of the real data's distribution.

Overall, the Banksformer model consistently provides the closest approximation to the real data across all evaluated aspects, including shape, topology, density distribution, and spread, despite none of the models perfectly capturing every characteristic of the real data.

Recurring Transactions. The PCA distribution of recurring transactions is depicted in Figure 3.5. BankGAN seems to capture the general shape of the data distribution but struggles with the nuances, such as the separation between clusters and the exact shape of each cluster. This could indicate that while BankGAN is somewhat effective in understanding the dataset's global structure, it may lack in capturing the finer, local details. StackedLSTM might be capturing some linear relationships in the data but fails to model the more complex, non-linear interactions. The significant deviation from the real data's distribution indicates that StackedLSTM might not be suitable for datasets that exhibit complex, non-linear patterns or require the preservation of multimodal distributions. Banksformer appears to have a better grasp of the data's underlying structure compared to StackedLSTM but still falls short of completely capturing the real data's complexity. It represents an improvement in terms of understanding both linear and non-linear relationships within the

data, although it still struggles with accurately defining the boundaries and densities of different clusters.

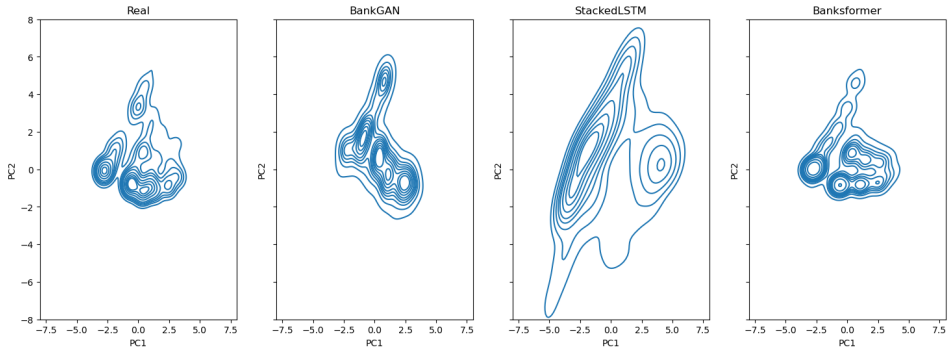

Figure 3.5: PCA visualization of real data and synthetic data generated by the three models, only focusing on recurring transactions.

Non-Recurring Transactions. As shown in Figure 3.6, the models exhibit varying degrees of success in replicating the real data distribution of non-recurring transactions. The BankGAN distribution is elongated and spans a larger range on PC1 and PC2. Unlike the real data, it shows a single, stretched cluster, indicating a different or more generalized data representation. The StackedLSTM shows a very elongated, single cluster with a clear direction, suggesting a significant variance along one principal axis. It differs from the real data's compact and separated clusters. The Banksformer distribution is closer to the real data with distinct, although not completely separated, clusters. However, it covers a broader area than the real data, indicating higher variance within clusters.

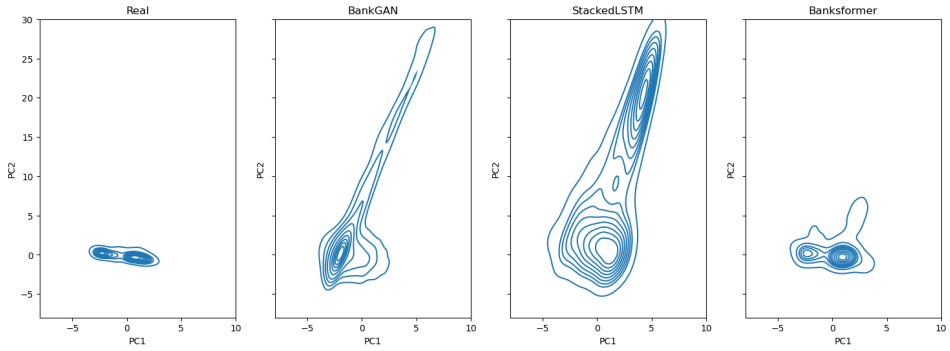

Figure 3.6: PCA visualization of real data and synthetic data generated by the three models, only focusing on non-recurring transactions.

3.2.3 Privacy Preservability

Our analysis extends to assessing the privacy preservation capabilities of the synthetic data generated by the models. We adopted a differentially private version of each model, utilizing a modified training algorithm that employs differentially private stochastic gradient descent (DP-SGD) [50]. This method enhances privacy by clipping gradients and injecting noise into them. The efficacy of differential privacy is quantified by two parameters: epsilon (ϵ) and delta (δ), where lower ϵ values signify stronger privacy assurances, and δ represents the likelihood of these assurances not being met [51]. For our evaluation, we selected the optimal model variants employing clock encoding from Table 3.1. The differential privacy parameters were set with $\delta = 10^{-5}$, and ϵ values were calculated with the tools provided by the Tensorflow Privacy library[1]. The evaluation of synthetic data quality from models employing

[1]https://github.com/tensorflow/privacy

Method	Differential Privacy	Amt WD	CF WD	Td WD	Tcode JSD	Day JSD	Tcode-3g JSD	Tcode-Day JSD
BankGAN	NO	2057	9542	0.73	0.09	0.06	0.25	0.26
	YES($\epsilon = 0.60$)	2033	6703	0.8	0.09	0.06	0.25	0.25
Banksformer	NO	4236	4338	0.15	0.016	0.010	0.058	0.035
	YES($\epsilon = 1.16$)	4918	5241	0.6	0.036	0.016	0.11	0.058
StackedLSTM	NO	4277	3674	1.96	0.064	0.15	0.28	0.26
	YES($\epsilon = 1.29$)	4174	4226	10.25	0.035	0.15	0.24	0.21

Table 3.2: Performance comparison of BankGAN, Banksformer, and StackedLSTM with and without differential privacy (DP).

differential privacy (DP) versus their original counterparts is summarized in Table 3.2. In the Banksformer study [15], differential privacy (DP) degrades data quality, but the StackedLSTM model under DP either matches or surpasses the original across most metrics, except for a notable decrease in Time Delta, reflecting divergence from the original in its ability to replicate customer transaction frequencies. BankGAN, with the lowest epsilon, improves synthetic data quality in all aspects, barely affecting Time Delta, showing no degradation in data quality. While adding privacy guarantees typically compromises performance [50], they can also serve as an effective regularization mechanism and improve results [52], a phenomenon observed in BankGAN. However, Adding DP to Banksformer degrades the quality of generated data, it does so only to the level of the BankGAN generated data. It looks like the

noise added by DP is similar in magnitude to the inherent generation noise of BankGAN. That may be why these metrics are not significantly affected by adding DP to BankGAN.

In conclusion, the robustness of our models, BankGAN, towards differential privacy marks it as a promising candidate for future advancements in the field of privacy-preserving synthetic data generation.

Chapter 4

Discussion

4.1 Discussion & Future Work

Our experimentation highlights the effectiveness of BankGAN, which integrates a date mechanism from Banksformer [15] and a sequential conditional vector derived from transformer-generated sequences into a fully connected conditional tabular GAN architecture. This fusion enables BankGAN to surpass StackedLSTM, which is a sequence-specific model, on most metrics.

In our comparative analysis of BankGAN and Banksformer [15] , it becomes evident that each model excels in different aspects of synthetic data generation. Our exploration into differential privacy-enhanced variant of BankGAN reveals a significant advancement in maintaining data utility while ensuring robust privacy protections. Unlike Banksformer [15] , where the application of dif-

ferential privacy techniques typically results in a degradation of performance, the differential privacy-enhanced BankGAN either maintains or improves performance across all metrics. This distinct characteristic of the BankGAN model showcases its exceptional ability to balance data utility and privacy—a critical requirement in the financial sector. The successful integration of differential privacy without compromising data quality highlights the potential of BankGAN to set a new benchmark for privacy-preserving synthetic data generation in sensitive applications.

Additionally, BankGAN has demonstrated superior performance in generating transaction amount and cash flow values for recurring transactions. This suggests that BankGAN can be particularly effective in scenarios where the accuracy of simulating recurring transactions is critical.

However, BankGAN appears to have limitations in modeling time-based patterns effectively, as indicated by its performance in capturing the time delta(TD) distribution and the joint distribution of transaction code and day(Tcode-Day). This observation points to potential weaknesses in the model's ability to handle non-recurring transactions and more complex temporal sequences.

Given these observations, BankGAN is highly effective in specific contexts, particularly in generating high-quality data for recurring transactions. On the other hand, Banksformer shows more robust performance in capturing the

intricacies of time-based patterns, making it more suitable for applications requiring detailed temporal analysis.

To enhance the utility of BankGAN, future enhancements should focus on refining its ability to model sequences of non-recurring transactions. This could involve improvements to the transaction code (Tcode) generator to produce more accurate and realistic sequences for non-recurring transactions. Additionally, considering the bifurcation of BankGAN into two distinct models—one tailored for recurring and another for non-recurring transactions—could allow for more specialized and accurate data generation, reflecting the unique characteristics of each transaction type.

Moreover, the synthetic data generated by BankGAN could be utilized in training machine learning models where real data is scarce or too sensitive to use, such as in fraud detection algorithms. By training on both real and synthetic data, these models could potentially achieve higher accuracy and robustness.

The implications of the BankGAN model extend beyond the generation of synthetic data for privacy preservation. In the financial sector, the ability to replicate complex transaction patterns can enhance algorithm testing and financial monitoring systems without compromising user confidentiality. Furthermore, this model can be adapted for stress testing financial systems against various economic scenarios by generating data under hypothetical conditions,

thus providing insights into the resilience of financial institutions under potential crises.

Future research could focus on improving the robustness and diversity of the synthetic data generation by integrating multi-modal learning approaches, which could help the model capture a broader range of patterns and nuances in transaction data. Additionally, exploring hybrid models that combine the strengths of GANs with other deep learning architectures could address some of the training stability issues mentioned earlier.

Incorporating feedback loops from domain experts in finance to iteratively refine the model's output could also enhance the practical utility of the synthetic data, ensuring that it meets the specific needs of financial institutions more effectively.

Appendix

Epoch-wise Generator Loss and Discriminator Loss in BankGAN Experiments Training

In these experiments, we trained the BankGAN using different sets of hyper-parameters to find ones that allow successful GAN convergence.

In the following plots, we define several key metrics to characterize the performance and behavior of both the generator and the discriminator.

By plotting these terms we aim to characterize the performance and behavior of both the generator and the discriminator. Let us delve deeper into each of these metrics to better understand their significance and the insights they offer into the training dynamics of our GAN model:

- **Discriminator Loss_real** $(-D(x))$: This term represents the average score that the discriminator gives to real data samples. When we plot it as is, a more negative value indicates that the discriminator assigns higher scores (or values) to real samples. As the GAN training progresses, ideally, we want this value to be negative and become more negative, indicating that the discriminator is getting better at recognizing and assigning higher scores to real data.

- **Discriminator Loss_fake** $(D(G(z)))$: This represents the average score that the discriminator gives to fake data samples generated by the gen-

erator. Initially, this value might be negative if the generator produces poor samples, and the discriminator can easily distinguish them. However, as training progresses, the generator gets better, and $D(G(z))$ should increase. Ideally, upon convergence, this value should be close to the Discriminator Loss_real (without the negative sign) if the generator produces realistic samples.

- **Discriminator Loss_ave** $(D(G(z)) - D(x) + \text{Gradient penalty})$: This is the overall loss for the discriminator, including the gradient penalty to ensure the 1-Lipschitz continuity. The gradient penalty is crucial for the theoretical properties of WGANs to hold and prevents mode collapse and vanishing/exploding gradients. As the GAN converges, the value of $D(G(z)) - D(x)$ should be close to 0, meaning the discriminator finds it equally challenging to differentiate between real and fake samples.

- **Generator Loss** $(-D(G(z)) + CrossEntropy)$: Initially, as the generator might produce samples that are easy for the discriminator to distinguish as fake, the generator loss might be relatively high (in terms of absolute value). However, as the generator improves and starts producing more realistic samples, the value of this loss should decrease. The reason being that the discriminator would find it harder to distinguish the fake samples from real, hence it gives them a higher score.

The table below lists the hyperparameters used in various experiments, accom-

panied by plots showing the generator and discriminator loss at each epoch for each experiment. It's evident from Experiment_v4 that the GAN model has successfully converged.

	Emb_Dim	*Gen_Dim*	*Disc_Dim*	*GP Lambda*	*Disc_Step*	*pac*
Experiment 1	100	(256,128)	(128,256)	10	1	10
Experiment 2	100	(256,128)	(128,256)	10	3	10
Experiment 3	128	(256,256)	(256,256)	10	1	10
Experiment 4	**100**	**(256,128)**	**(128,256)**	**1**	**1**	**10**
Experiment 5	100	(256,128)	(128,256)	0.1	1	10
Experiment 6	128	(256,256)	(256,256)	1	1	10

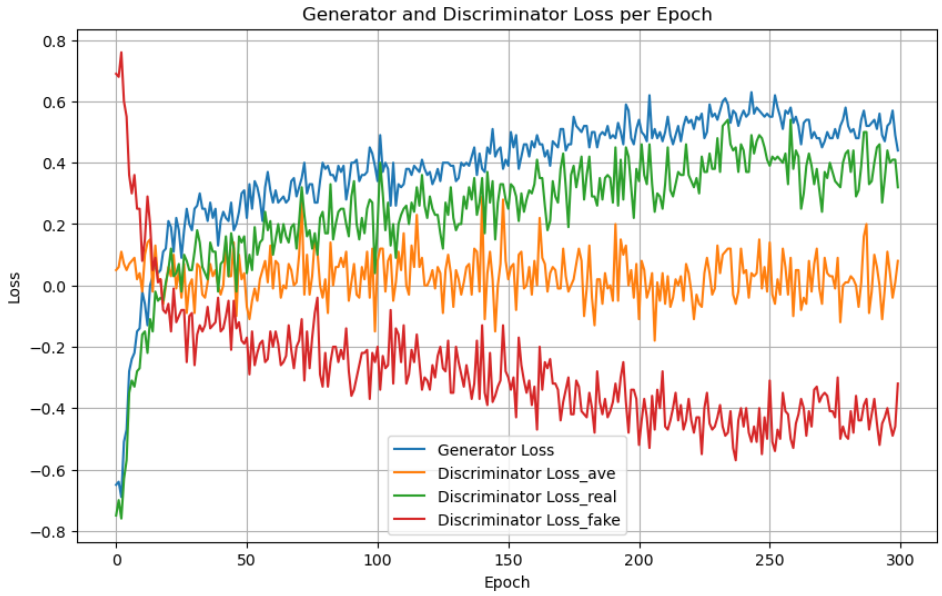

Figure 1: Experiment1

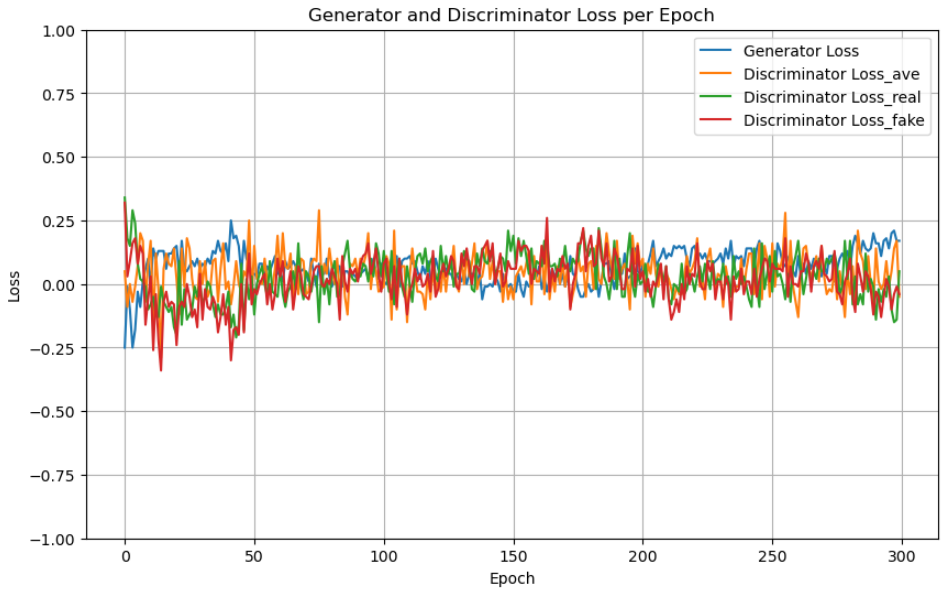

Figure 2: Experiment2

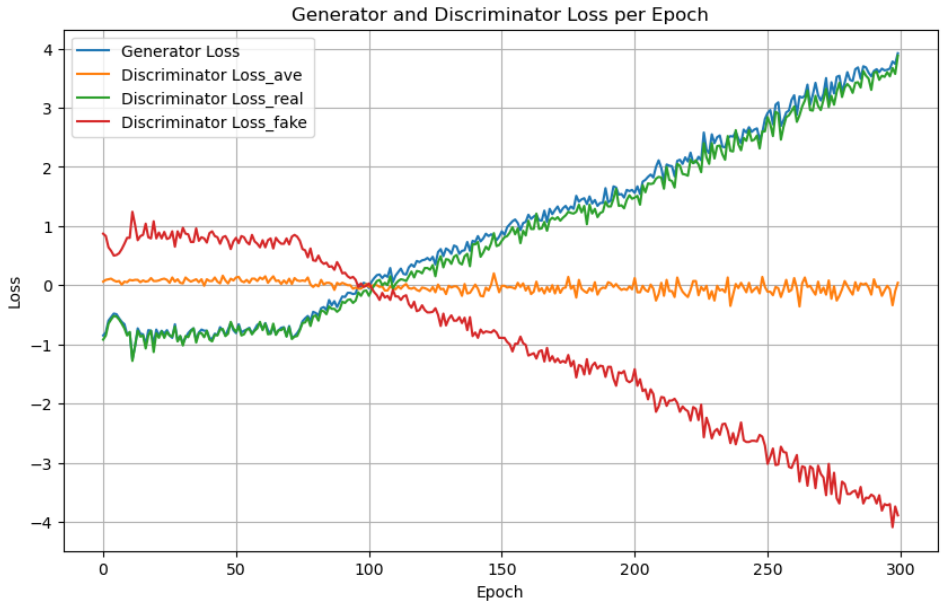

Figure 3: Experiment3

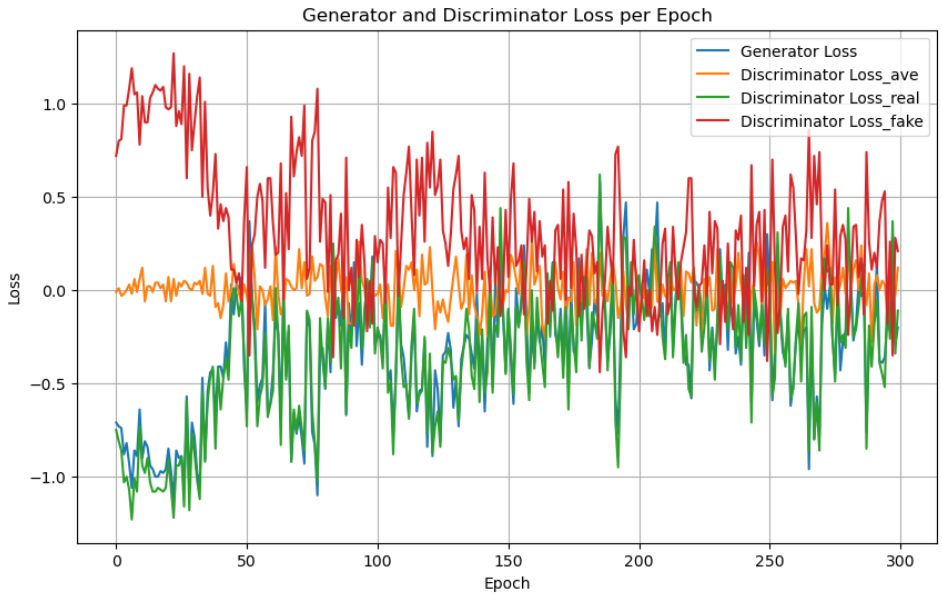

Figure 4: Experiment4

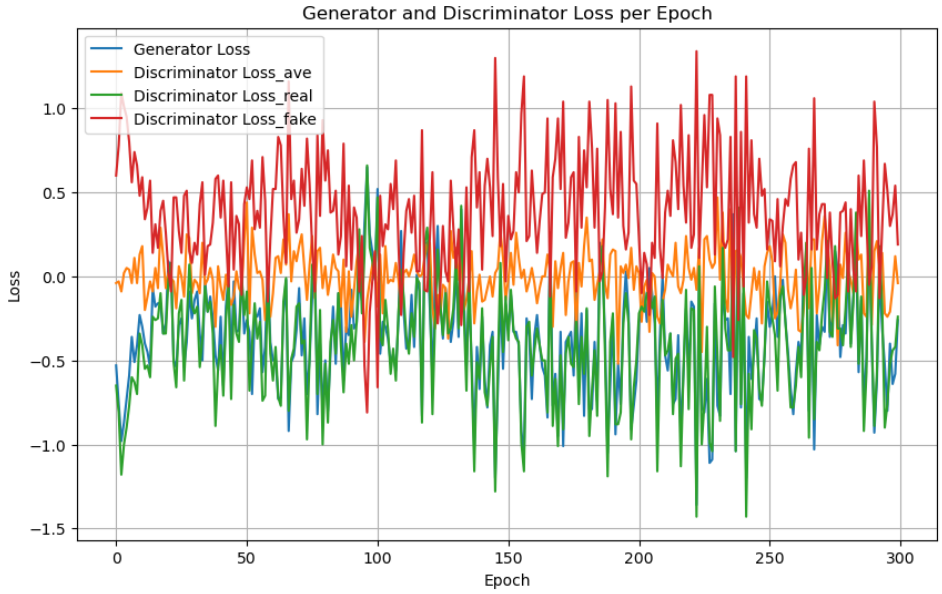

Figure 5: Experiment5

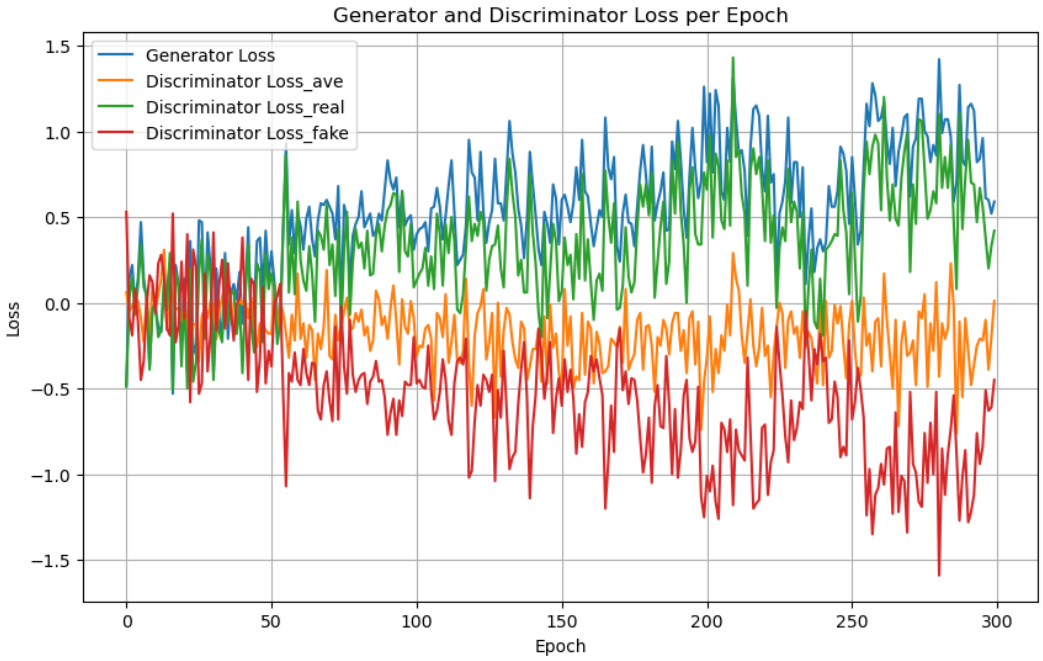

Figure 6: Experiment6

Bibliography

[1] Samuel A Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8, 2020.

[2] JAMIE COLLIER. Insights: The financial sector remains a popular target for cybercriminals, 2022. Accessed: 2023-10-12.

[3] Alejandro Mottini, Alix Lheritier, and Rodrigo Acuna-Agost. Airline passenger name record generation using generative adversarial networks. *arXiv preprint arXiv:1807.06657*, 2018.

[4] James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N Cohen, and Adrian Weller. Synthetic data–what, why and how? *arXiv preprint arXiv:2205.03257*, 2022.

[5] CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[6] Alastair Gregory, F. Din-Houn Lau, Mark Girolami, Liam J. Butler, and Mohammed Z.E.B. Elshafie. The synthesis of data from instrumented structures and physics-based models via gaussian processes. *Journal of Computational Physics*, 392:248–265, 2019.

[7] Edgar Alonso Lopez-Rojas and Stefan Axelsson. Banksim: A bank payments simulator for fraud detection research. In *26th European Modeling and Simulation Symposium, EMSS*, volume 2014, 2014.

[8] Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. Paysim: A financial mobile money simulator for fraud detection. In *28th European Modeling and Simulation Symposium, EMSS, Larnaca*, pages 249–255. Dime University of Genoa, 2016.

[9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[14] Gary Stafford. Unlocking the potential of generative ai for synthetic data generation, 2019. Accessed: 2023-10-12.

[15] Kyle Nickerson, Terrence Tricco, Antonina Kolokolova, Farzaneh Shoeleh, Charles Robertson, John Hawkin, and Ting Hu. Banksformer: A deep generative model for synthetic transaction sequences. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 121–136. Springer, 2022.

[16] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using gan for improved

liver lesion classification. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*, pages 289–293. IEEE, 2018.

[17] A Langevin, T Cody, S Adams, and P Beling. Synthetic data augmentation of imbalanced datasets with generative adversarial networks under varying distributional assumptions: A case study in credit card fraud detection. *J. Oper. Res. Soc*, pages 1–28, 2021.

[18] John T Guibas, Tejpal S Virdi, and Peter S Li. Synthetic medical images from dual generative adversarial networks. *arXiv preprint arXiv:1709.01872*, 2017.

[19] Engin Dikici, Matthew Bigelow, Richard D White, Barbaros S Erdal, and Luciano M Prevedello. Constrained generative adversarial network ensembles for sharable synthetic medical images. *Journal of Medical Imaging*, 8(2):024004–024004, 2021.

[20] Antonio J Rodriguez-Almeida, Himar Fabelo, Samuel Ortega, Alejandro Deniz, Francisco J Balea-Fernandez, Eduardo Quevedo, Cristina Soguero-Ruiz, Ana M Wägner, and Gustavo M Callico. Synthetic patient data generation and evaluation in disease prediction using small and imbalanced datasets. *IEEE Journal of Biomedical and Health Informatics*, 2022.

[21] Emilija Strelcenia and Simant Prakoonwit. Improving classification performance in credit card fraud detection by using new data augmentation.

AI, 4(1):172–198, 2023.

[22] Braden Hurl, Robin Cohen, Krzysztof Czarnecki, and Steven Waslander. Trupercept: Trust modelling for autonomous vehicle cooperative perception from synthetic data. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 341–347. IEEE, 2020.

[23] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2522–2529. IEEE, 2019.

[24] Andre Goncalves, Priyadip Ray, Braden Soper, Jennifer Stevens, Linda Coyle, and Ana Paula Sales. Generation and evaluation of synthetic patient data. *BMC medical research methodology*, 20(1):1–40, 2020.

[25] Denish Azamuke, Marriette Katarahweire, and Engineer Bainomugisha. Scenario-based synthetic dataset generation for mobile money transactions. In *Proceedings of the Federated Africa and Middle East Conference on Software Engineering*, pages 64–72, 2022.

[26] E. A. Lopez-Rojas and Stefan Axelsson. Banksim: a bank payments simulator for fraud detection research. 2014.

[27] Edgar Lopez-Rojas, Ahmad Elmir, and Stefan Axelsson. Paysim: A financial mobile money simulator for fraud detection. In *28th European*

Modeling and Simulation Symposium, EMSS, Larnaca, pages 249–255. Dime University of Genoa, 2016.

[28] Mark Weber, Jie Chen, Toyotaro Suzumura, Aldo Pareja, Tengfei Ma, Hiroki Kanezashi, Tim Kaler, Charles E Leiserson, and Tao B Schardl. Scalable graph learning for anti-money laundering: A first look. *arXiv preprint arXiv:1812.00076*, 2018.

[29] Edgar Alonso Lopez-Rojas, Dan Gorton, and Stefan Axelsson. Retsim: A shoestore agent-based simulation for fraud detection. In *25th European Modeling and Simulation Symposium, EMSS 2013; Athens; Greece*, pages 25–34, 2013.

[30] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[31] Mathijs van Bree. *Unlocking the Potential of Synthetic Tabular Data Generation with Variational Autoencoders*. PhD thesis, Tilburg University Tilburg, The Netherlands, 2020.

[32] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *Advances in neural information processing systems*, 32, 2019.

[33] L Vivek Harsha Vardhan and Stanley Kok. Generating privacy-preserving synthetic tabular data using oblivious variational autoencoders. In *Proceedings of the Workshop on Economics of Privacy and Data Labor at the 37 th International Conference on Machine Learning (ICML)*, 2020.

[34] Yandan Tan, Hongbin Zhu, Jie Wu, and Hongfeng Chai. Dptvae: Data-driven prior-based tabular variational autoencoder for credit data synthesizing. *Expert Systems with Applications*, 241, 2024.

[35] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. In *International Conference on Machine Learning*, pages 17564–17579. PMLR, 2023.

[36] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *arXiv preprint arXiv:1806.03384*, 2018.

[37] Zhengping Che, Yu Cheng, Shuangfei Zhai, Zhaonan Sun, and Yan Liu. Boosting deep learning risk prediction with generative adversarial networks for electronic health records. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 787–792. IEEE, 2017.

[38] Alejandro Mottini, Alix Lheritier, and Rodrigo Acuna-Agost. Airline passenger name record generation using generative adversarial networks.

arXiv preprint arXiv:1807.06657, 2018.

[39] Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y Chen. Ctab-gan: Effective table data synthesizing. In *Asian Conference on Machine Learning*, pages 97–112. PMLR, 2021.

[40] Ying Yu, Bingying Tang, Ronglai Lin, Shufa Han, Tang Tang, and Ming Chen. Cwgan: Conditional wasserstein generative adversarial nets for fault data generation. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2713–2718. IEEE, 2019.

[41] Georgios Douzas and Fernando Bacao. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with applications*, 91:464–471, 2018.

[42] Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

[43] Zinan Lin, Alankar Jain, Chen Wang, Giulia Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference*, pages 464–483, 2020.

[44] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation. *arXiv preprint arXiv:2111.08095*, 2021.

[45] Andrea Coletta, Sriram Gopalakrishan, Daniel Borrajo, and Svitlana Vyetrenko. On the constrained time-series generation problem. *arXiv preprint arXiv:2307.01717*, 2023.

[46] Haksoo Lim, Minjung Kim, Sewon Park, and Noseong Park. Regular time-series generation using sgm. *arXiv preprint arXiv:2301.08518*, 2023.

[47] Sujan Ghimire, Ravinesh C Deo, Hua Wang, Mohanad S Al-Musaylh, David Casillas-Pérez, and Sancho Salcedo-Sanz. Stacked lstm sequence-to-sequence autoencoder with feature selection for daily solar radiation prediction: a review and new modeling results. *Energies*, 15(3):1061, 2022.

[48] KU Jaseena and Binsu C Kovoor. A hybrid wind speed forecasting model using stacked autoencoder and lstm. *Journal of Renewable and Sustainable Energy*, 12(2), 2020.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[50] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[51] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

[52] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908*, 2018.