PFN-Syn: Generating Synthetic Tabular Data with Prior-Data Fitted Networks

Michael Zuo
RPI
Troy, NY, USA
zuom@rpi.edu

Inwon Kang
RPI
Troy, NY, USA
kangi@rpi.edu

Oshani Seneviratne
RPI
Troy, NY, USA
senevo@rpi.edu

Stacy Patterson
RPI
Troy, NY, USA
sep@cs.rpi.edu

ABSTRACT

Synthetic tabular data can help enable data sharing and model development in privacy-sensitive domains, but existing generators often require dataset-specific training and substantial computation. We introduce PFN-Syn, a training-free framework for synthetic tabular data generation based on prior-data fitted networks. We evaluate PFN-Syn against training-based generators on eight binary classification datasets. PFN-Syn achieves the highest average downstream utility among evaluated synthetic generators, with higher generation throughput than training-based baselines, while maintaining competitive distributional similarity and distance-to-closest-record privacy behavior. Our results suggest that tabular foundation models can serve as useful building blocks for practical synthetic tabular data generation.

VLDB Workshop Reference Format:
Michael Zuo, Inwon Kang, Oshani Seneviratne, and Stacy Patterson. PFN-Syn: Generating Synthetic Tabular Data with Prior-Data Fitted Networks. VLDB 2026 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/brains-group/table-agent-toolkit.

1 INTRODUCTION

Tabular data is central to many modern AI applications in domains such as finance and healthcare, where predictive models rely on large volumes of structured data to identify patterns, support decision-making, and improve outcomes. However, these same domains are governed by strict regulations on data privacy and sharing, such as the General Data Protection Regulation (GDPR), Health Insurance Portability and Accountability Act (HIPAA), and Fair Credit Reporting Act (FCRA). Synthetic data generation offers a promising solution to this challenge by creating artificial datasets that preserve the statistical structure and relationships of the original data without directly exposing sensitive records [3].

A variety of synthetic data generation methods have been developed for tabular data. Early approaches relied primarily on statistical methods such as the Gaussian Copula [13]. More modern techniques include generative adversarial networks (GANs) [21], variational autoencoders (VAEs) [21], diffusion-based methods [17], and LLM-based generators [2]. While these methods have demonstrated strong performance in many settings, they often require significant computational resources, careful hyperparameter tuning, and repeated retraining when datasets or tasks change.

To address this challenge, we propose a *training-free* alternative path to high-quality synthetic tabular data generation using a prior-data fitted network (PFN) [12]. A PFN is a model that is pre-trained on synthetic data sampled from a prior family of distributions. At inference time, the model takes both training data and query samples for a given task as input and performs prediction based directly on the in-context training samples. There is no need to retrain the model for each new dataset and/or task.

In this work, we introduce PFN-Syn, a principled framework for converting a PFN-based predictor into a synthetic data generator for tabular data. PFN-Syn inherits the benefits of PFN predictors: it is training-free and computationally efficient. Evaluated on standard benchmark datasets, PFN-Syn achieves higher average downstream task utility than state-of-the-art training-based tabular data generators, while matching these generators on various quality metrics.

Related Work. PFN-Syn builds on existing work on PFN predictors. Recently, specialized PFNs have been developed for in-context learning and prediction on tabular data, TabPFN [7, 9] and TabICL [14, 15] being the most prominent examples. These PFN-based predictors outperform state-of-the-art ML methods on various tabular dataset benchmarks [5].

We apply an autoregressive approach to synthetic data generation which is conceptually similar to an unsupervised imputation-based framing described as an extension to TabPFN [7, 20]. However, rather than extending imputation, our method explicitly generates synthetic records by sampling conditional distributions.

While a prior work uses a PFN backing an energy-based model to predict labels for synthetically generated samples [10], our method generates a fully synthetic table from a real table by using the PFN to predict all columns, conditioned on previous column values.

Outline. In Section 2, we present the details of PFN-Syn. Section 3 gives our experimental results, which explore variations on PFN-Syn and compare PFN-Syn's performance against various benchmark generators. Finally, we conclude in Section 4.

2 PFN-SYN

Given a real tabular dataset $\mathbf{X}$ with n samples and t features per sample, let $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_t$ be the columns of $\mathbf{X}$. We take $\mathbf{X}_t$ to be the target column; we assume target $\mathbf{X}_t$ takes on discrete values. We seek to produce a synthetic table with the same schema and joint distribution as $\mathbf{X}$.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

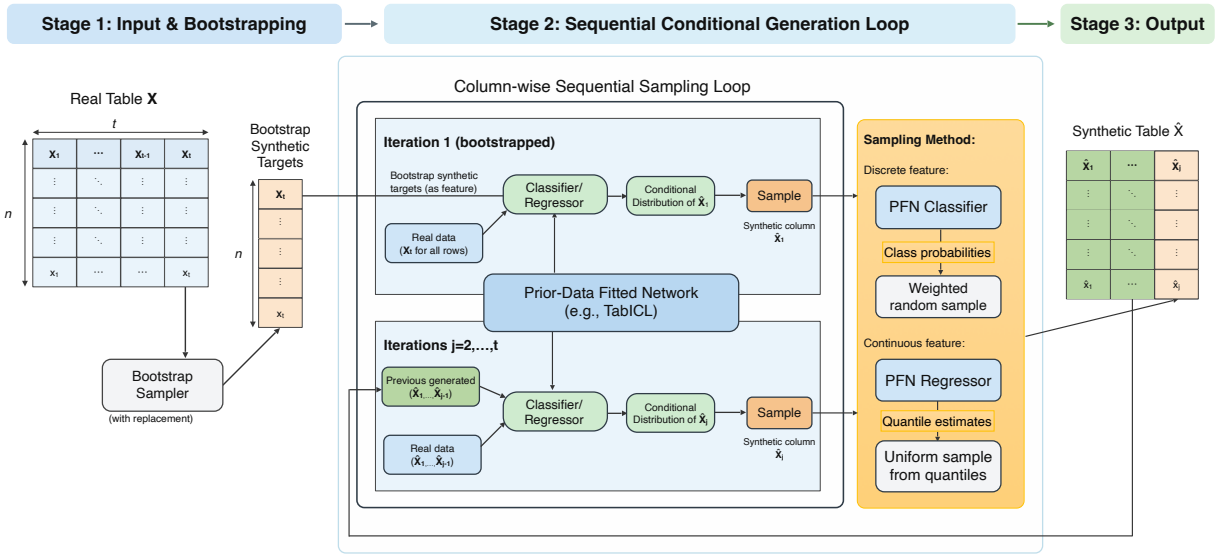


Figure 1: PFN-Syn synthetic data generation framework.

With a true joint distribution over all features, we could generate synthetic data simply by repeatedly sampling new rows from the distribution. However, directly modeling full joint distributions is intractable for practical tabular datasets. PFN-Syn factors the full joint distribution into a sequence of conditional sampling problems:

$$\Pr(\mathbf{X}_1, \ldots, \mathbf{X}_t) = \prod_{j=1}^{t} \Pr(\mathbf{X}_j \mid \mathbf{X}_1, \ldots, \mathbf{X}_{j-1}).$$

This allows us to generate one column at a time instead of modeling all columns at once. We can now approximate each subproblem by using a PFN to estimate conditional distributions for each column given generated values in all previous columns.

To generate a synthetic row, we first bootstrap the conditional generation process by generating synthetic values $\hat{\mathbf{X}}_0$ according to the unconditional distribution of the target column $\mathbf{X}_t$ in the real data. In the case where the target is a discrete categorical feature, this is equivalent to sampling directly from target column $\mathbf{X}_t$ of the real data, with replacement. The bootstrap targets $\hat{\mathbf{X}}_0$ are used only when generating the first synthetic column and are then discarded; they are not included in the final output.

To generate the first synthetic column $\hat{\mathbf{X}}_1$, we construct a PFN training input using the real target values $\mathbf{X}_t$ as the single feature and the real first column $\mathbf{X}_1$ values as the target. We then use the PFN to generate synthetic values for $\hat{\mathbf{X}}_1$ from the bootstrap targets $\hat{\mathbf{X}}_0$ (as a feature) by the sampling procedure described below. For each subsequent column $j = 2, \ldots, t$, we invoke the PFN using real columns $\mathbf{X}_1, \ldots, \mathbf{X}_j$ and synthetic columns $\hat{\mathbf{X}}_1, \ldots, \hat{\mathbf{X}}_{j-1}$ as input. (The bootstrap target $\hat{\mathbf{X}}_0$ is not included in the query for columns $j \geq 2$.) The underlying PFN takes as input a combined table including both real and synthetic data for the fully-known columns $1, \ldots, j-1$, and the partial list of known target values $\mathbf{X}_j$. From these inputs it generates predictions to complete the missing target values, which we use to sample the new synthetic values $\hat{\mathbf{X}}_j$.

The sampling procedure used to generate each column depends on whether that column is discrete or continuous. To generate a discrete feature $\hat{\mathbf{X}}_j$ for each entry, we use a classifier PFN to produce a set of class probabilities conditioned on real data $\mathbf{X}_1, \ldots, \mathbf{X}_j$ and previously generated columns $\hat{\mathbf{X}}_1, \ldots, \hat{\mathbf{X}}_{j-1}$. We determine each entry's class in the new column $\hat{\mathbf{X}}_j$ by sampling randomly with weights given by the classifier's reported probabilities. The selected class is not necessarily the class predicted with the highest probability or the best-represented class in the real data, but rather is randomized to approximate the conditional distribution implied by the classifier prediction.

For continuous features, we instead use a regressor PFN fitted on $\mathbf{X}_1, \ldots, \mathbf{X}_j$ and $\hat{\mathbf{X}}_1, \ldots, \hat{\mathbf{X}}_{j-1}$ to estimate values at a set of quantiles evenly spaced within the full range $[0, 1)$. We sample values for $\hat{\mathbf{X}}_j$ uniformly from among these quantiles. This procedure approximates sampling the value at a uniformly random quantile from the regressor prediction's implied distribution, drawing from the continuous feature space rather than copying any specific value from the real data.

For computational efficiency, synthetic rows are not generated individually. We batch full sets of synthetic data so that each invocation of the underlying PFN produces the statistics required to sample all new values for a single column. This allows for parallelism and reuse of the "training" corpus at each step, which is well-supported by the underlying PFN models.

3 EVALUATION RESULTS

Setup. We instantiate PFN-Syn using TabICL v2 to provide the underlying PFN. We compare PFN-Syn against Gaussian Copula, TabDiff, CTGAN, and TVAE in their reference configurations on eight binary classification tabular datasets (see Table 1). Each generator produces a synthetic table of the same size as the original table. We perform experiments on an Nvidia RTX A6000 GPU with 48

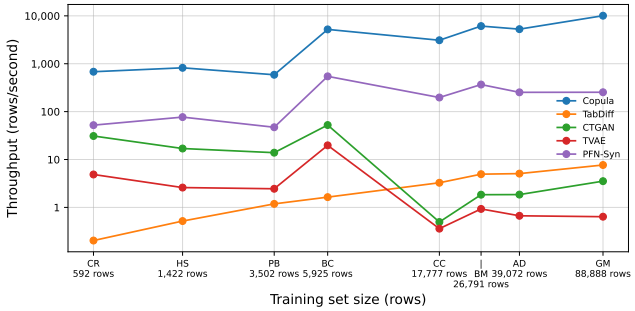
Table 1: Comparison of balanced accuracy of prediction from original training data and synthetic data from various generators. XGB columns show results for predictions with XGBoost, and ICL columns show results for predictions made by TabICL.

Dataset	Original		Copula		TabDiff		CTGAN		TVAE		PFN-Syn	
	XGB	ICL	XGB	ICL	XGB	ICL	XGB	ICL	XGB	ICL	XGB	ICL
Adult Income (AD) [1]	0.818	0.844	0.516	0.507	0.684	0.640	0.718	0.711	**0.769**	**0.758**	0.762	0.684
Bank Customer Churn (BC) [19]	0.721	0.726	0.540	0.508	0.707	0.719	0.670	0.654	0.650	0.633	**0.714**	**0.740**
Bank Marketing (BM) [11]	0.596	0.594	0.506	0.5	0.576	0.580	0.586	0.589	**0.603**	**0.604**	0.584	0.578
Default of Credit Card Clients (CC) [22]	0.658	0.659	0.546	0.503	0.642	0.648	0.609	0.623	0.635	0.631	**0.655**	**0.653**
German Credit Data (CR) [8]	0.686	0.673	0.533	0.497	0.614	**0.659**	0.507	0.5	0.552	0.570	**0.616**	0.620
Give Me Some Credit (GM) [6]	0.594	0.581	0.502	0.5	0.583	0.565	0.628	0.618	**0.666**	**0.662**	0.640	0.650
Hazelnut Spread Contaminant Detection (HS) [16]	0.897	0.965	0.803	0.85	0.872	0.910	0.566	0.487	0.785	0.788	**0.873**	**0.922**
Polish Companies Bankruptcy (PB) [18]	0.760	0.755	0.499	0.5	0.601	0.523	**0.677**	**0.635**	0.502	0.517	0.581	0.574
Mean	0.716	0.725	0.556	0.546	0.660	0.656	0.621	0.602	0.645	0.645	0.678	0.678

Table 2: Average utility, quality, and privacy metrics over eight datasets (higher is better).

Generator	XGB	ICL	Shape	Pair	DCR-B	DCR-O
Original	0.716	0.725	(1)	(1)	(0)	(0)
Copula	0.556	0.546	0.875	0.871	0.258	0.815
TabDiff	0.660	0.656	**0.972**	**0.962**	0.135	0.823
CTGAN	0.621	0.602	0.789	0.815	**0.263**	**0.866**
TVAE	0.645	0.645	0.857	0.802	0.091	0.786
PFN-Syn	**0.678**	**0.678**	0.856	0.841	0.131	0.855

Figure 2: Comparison of data generators by throughput, plotted log-log against training set size (higher is better).



GB memory, under the conditions given in [23], averaging results over 3 generated datasets in each configuration.

3.1 Quality and Utility

We measure the utility of the resulting synthetic datasets by training downstream XGBoost and TabICL classifiers on the synthetic data produced by each generator and evaluating balanced accuracy on held-out real data. These results are shown in Table 1.

PFN-Syn, while not uniformly dominant, is consistently competitive across all datasets. All other methods except TabDiff suffer a discriminative collapse on some dataset, achieving balanced accuracy comparable to random guessing.

Table 2 reports average scores across all datasets for balanced accuracy with the XGBoost and ICL classifiers, as well as column shapes and column pair trends data quality metrics and the distance to closest record baseline (DCR-B) and overfitting (DCR-O) protection metrics from SDMetrics [4]. PFN-Syn achieves the highest average downstream balanced accuracy among the synthetic generators under both downstream classifiers.

On quality and protection metrics, PFN-Syn is solidly middle-of-the-pack. On the column shape and pair trends metrics, PFN-Syn scores within the range of the training-based generators, suggesting that it preserves marginal feature distributions comparably well. The DCR protection metrics suggest that PFN-Syn's strong performance on balanced accuracy is not attributable to merely reproducing the real data. PFN-Syn does not generate synthetic records that are unusually similar to training records compared to the other generators.

Table 3: Comparison of XGBoost balanced accuracy between baseline and scalable variants.

Dataset	Full size	Split		Coreset	
		3:1	10:1	3:1	10:1
AD	0.762	0.744	0.766	0.768	0.667
BC	0.714	0.723	0.685	0.712	0.736
BM	0.584	0.592	0.593	0.585	0.592
CC	0.655	0.665	0.655	0.662	0.662
CR	0.616	0.579	0.586	0.571	0.543
GM	0.640	0.628	0.624	0.632	0.641
HS	0.873	0.860	0.862	0.852	0.833
PB	0.581	0.591	0.572	0.621	0.597
Mean	0.678	0.673	0.668	0.675	0.659

3.2 Scalability

PFN-Syn's strong performance is particularly appealing as it is very fast compared to training-based generators. Figure 2 plots estimated synthetic data generation throughput of each generator when used to generate synthetic datasets of equal size to the training data. For training-based generators, timings include training and generation time. PFN-Syn is outperformed only by Gaussian Copula, a statistical generator with poor output utility. Compared to generators based on dataset-specific model training, PFN-Syn achieves 1.7–74 times higher throughput while maintaining competitive data

Table 4: Comparison of balanced accuracy between baseline and randomized feature generation orders.

Dataset	In-Order		Target Last		Full Random	
	XGB	ICL	XGB	ICL	XGB	ICL
AD	0.762	0.684	0.693	0.663	0.705	0.643
BC	0.714	0.740	0.713	0.729	0.723	0.709
BM	0.584	0.578	0.593	0.579	0.575	0.582
CC	0.655	0.653	0.658	0.658	0.650	0.643
CR	0.616	0.620	0.613	0.635	0.609	0.625
GM	0.640	0.650	0.654	0.626	0.640	0.630
HS	0.873	0.922	0.879	0.920	0.872	0.910
PB	0.581	0.574	0.553	0.535	0.598	0.627
Mean	0.678	0.678	0.670	0.668	0.672	0.671

quality and downstream utility, qualities that make PFN-Syn attractive for rapid exploratory analysis and practical synthetic data generation on small to moderately large datasets.

PFN-Syn inherits the context and query size constraints of its PFN substrate; the TabICL v2 models on which our experiments are based report good performance for up to 100,000 samples, requiring disk offloading for larger datasets due to an intermediate memory bottleneck [14]. However, PFN-Syn also stands to benefit from improvements in the underlying foundation models; the recent TabPFN-3 claims good performance up to 1 million rows with small feature counts [7].

As existing PFNs are primarily memory-constrained on total input size for a single invocation, we explore two approaches to scaling PFN-Syn by applying the underlying PFN model on smaller or partial inputs.

In the "split" settings, we randomly divide the samples evenly in each data set into disjoint batches, using 3 and 10 batches in total, respectively. We then apply PFN-Syn to generate synthetic data from each batch equal to the smaller batch size. Concatenating the synthetic outputs from each batch results in a total output size equal to the original dataset. While the total amount of input to the underlying PFN does not decrease, the input to each individual instance is smaller. Moreover, as there is no interaction between the samples in each batch, data generation from each batch can be performed in parallel.

In the "coreset" settings, we apply K-means clustering to select a representative coreset smaller than the original dataset. We find a number of clusters equal to 1/3 and 1/10 of the dataset size, respectively, then use the cluster centroids as an input dataset for PFN-Syn. This approach does reduce the total amount of input to the underlying PFN, but requires a variable amount of precomputation, depending on the coreset selection method.

We present the performance impact of these variations in Table 3. While we observe the intuitively expected trend of reduced utility as we reduce per-input size overall, the empirical impact is small and performs inconsistently between datasets. This limited performance degradation suggests that scaling of the underlying PFN models is not fundamentally problematic for synthetic data generation.

3.3 Feature Generation Ordering

We examine the impact of feature generation order on downstream performance by comparing the PFN-Syn's straightforward in-order generation scheme with randomly shuffled column orders.

The in-order setting is baseline PFN-Syn, as previously described. In the "target last" setting, we randomly permute the order of non-target features, but ensure that the target feature of the downstream task is used as the bootstrap target and generated as the final column. In the "full random" setting, we include the target column in the permutation, allowing it to be generated at any point during the sampling process. Each trial of the random settings uses a separate column permutation.

We compare the performance of these variations Table 4. While other work suggests that causal structure has an impact on synthetic data quality [20], we do not observe a significant difference in downstream task performance between in-order generation and randomly permuted orders, nor do we find any clear pattern where column order randomization degrades performance.

4 CONCLUSION

PFN-Syn reframes synthetic data generation as a sequence of in-context conditional prediction problems. Instead of training a new generator for each table, we repurpose a pre-trained tabular prior-data fitted network as a conditional sampler. Despite the simplicity of its design, PFN-Syn provides synthetic data with extremely strong downstream predictive utility on classification tasks across eight benchmark datasets, along with competitive distributional similarity and DCR-based privacy scores, while generating data very fast compared to training-based approaches. Our results suggest that pre-trained tabular foundation models are useful not only for prediction, but can also serve as useful building blocks for an efficient, practical alternative to training-based synthetic tabular data generation.

ACKNOWLEDGMENTS

Authors acknowledge the support from NSF IUCRC CRAFT center research grant (CRAFT Grant #22023) for this research. The opinions expressed in this publication do not necessarily represent the views of NSF IUCRC CRAFT.

REFERENCES

[1] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository.
[2] Vadim Borisov, Kathrin Sessler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. 2023. Language Models are Realistic Tabular Data Generators. In *International Conference on Learning Representations*.
[3] Fida K. Dankar, Mahmoud K. Ibrahim, and Leila Ismail. 2022. A Multi-Dimensional Evaluation of Synthetic Data Generators. *IEEE Access* 10 (2022), 11147–11158.
[4] DataCebo, Inc. 2025. Synthetic Data Metrics. https://docs.sdv.dev/sdmetrics/.
[5] Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Desai, David Salinas, and Frank Hutter. 2026. TabArena: A living benchmark for machine learning on tabular data. *Advances in Neural Information Processing Systems* 38 (2026).
[6] Credit Fusion and Will Cukierski. 2011. Give Me Some Credit. https://kaggle.com/competitions/GiveMeSomeCredit.
[7] Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Mihir Manium, Shi Bin Hoo, Magnus Bühler, Anurag Garg, Dominik Safaric, Jake Robertson, Benjamin Jäger, Simone Alessi, Adrian Hayler, Vladyslav Moroshan, Lennart Purucker, Philipp Singer, Alan Arazi, Julien Siems, Jan Hendrik Metzen, Georg Grab, Nick Erickson, Siyuan Guo, Eliott Kalfon, Simon Bing,

David Salinas, Clara Cornu, Lilly Charlotte Wehrhahn, Diana Kriuchkova, Kursat Kaya, Lydia Sidhoum, Marie Salmon, Jerry Chen, Madelon Hulsebos, Yann LeCun, Samuel Müller, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. 2026. TabPFN-3: Technical Report. arXiv:2605.13986 [cs.LG] https://arxiv.org/abs/2605.13986

[8] Hans Hofmann. 1994. Statlog (German Credit Data). UCI Machine Learning Repository.

[9] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. arXiv:2207.01848 [cs.LG] https://arxiv.org/abs/2207.01848

[10] Junwei Ma, Apoorv Dankar, George Stein, Guangwei Yu, and Anthony Caterini. 2024. TabPFGen – Tabular Data Generation with TabPFN. arXiv:2406.05216 [cs.LG] https://arxiv.org/abs/2406.05216

[11] S. Moro, P. Rita, and P. Cortez. 2014. Bank Marketing. UCI Machine Learning Repository.

[12] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. 2022. Transformers Can Do Bayesian Inference. In *International Conference on Learning Representations*.

[13] Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. 2016. The Synthetic Data Vault. In *IEEE International Conference on Data Science and Advanced Analytics*. 399–410.

[14] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. 2026. TabICLv2: A better, faster, scalable, and open tabular foundation model. (2026).

[15] Jingang QU, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. 2025. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In *International Conference on Machine Learning*.

[16] Marco Ricci, Bernardita Štitić, Luca Urbinati, Giuseppe Di Guglielmo, Jorge A. Tobón Vasquez, Luca P. Carloni, Francesca Vipiana, and Mario R. Casu. 2021. Machine-Learning-Based Microwave Sensing: A Case Study for the Food Industry. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 3 (2021), 503–514.

[17] Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. 2025. TabDiff: a Mixed-type Diffusion Model for Tabular Data Generation. In *International Conference on Learning Representations*.

[18] Sebastian Tomczak. 2016. Polish Companies Bankruptcy. UCI Machine Learning Repository.

[19] Gaurav Topre. 2022. Bank customer churn dataset. https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset.

[20] Davide Tugnoli, Andrea De Lorenzo, Marco Virgolin, and Giovanni Cinà. 2026. Improving TabPFN's Synthetic Data Generation by Integrating Causal Structure. arXiv:2603.10254 [cs.LG] https://arxiv.org/abs/2603.10254

[21] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling Tabular Data Using Conditional GAN. In *Advances in Neural Information Processing Systems*, Vol. 32.

[22] I-Cheng Yeh. 2009. Default of Credit Card Clients. UCI Machine Learning Repository.

[23] Michael Zuo, Inwon Kang, Stacy Patterson, and Oshani Seneviratne. 2026. Measuring Privacy Risks and Tradeoffs in Financial Synthetic Data Generation. In *Second International Workshop on Transformative Insights in Multifaceted Evaluation (with the Web Conference)*.