TabPFGen – Tabular Data Generation with TabPFN

Junwei Ma, Apoorv Dankar, George Stein, Guangwei Yu, Anthony Caterini
Layer 6 AI
Toronto, Canada
`{jeremy, apoorv, george, guang, anthony}@layer6.ai`

Abstract

Advances in deep generative modelling have not translated well to tabular data. We argue that this is caused by a mismatch in structure between popular generative models, and *discriminative* models of tabular data. We thus devise a technique to turn TabPFN – a highly performant transformer initially designed for in-context discriminative tabular tasks – into an energy-based generative model, which we dub *TabPFGen*. This novel framework leverages the pre-trained TabPFN as part of the energy function and does not require any additional training or hyperparameter tuning, thus inheriting TabPFN's in-context learning capability. We can sample from TabPFGen analogously to other energy-based models. We demonstrate strong results on standard generative modelling tasks, including data augmentation, class-balancing, and imputation, unlocking a new frontier of tabular data generation.

1 Introduction

Tabular data is pervasive and important across various domains [3, 35, 10, 33, 36], yet the application of deep generative modelling – successful in modalities like images [6, 37, 30] and text [11, 24] – has lagged behind in the tabular setting [25]. Previous works [34, 17, 28] have argued that attempts in this direction have not used high performing *discriminative* models of tabular data effectively. These discriminative models can assume various forms, encompassing tree-based and deep learning models. The non-differentiability of tree-based models makes it challenging to integrate them into gradient-based methods. Conversely, conventional deep learning-based models demand substantial time and effort in terms of training and hyperparameter tuning, rendering them hard to use across diverse datasets [31]. An exception to this trend is TabPFN [19], a transformer-based model designed for tabular data, which has demonstrated remarkable in-context learning capability for discriminative tasks. It is thus worth considering if TabPFN can be leveraged for *generative* tasks.

We answer this in the affirmative by introducing *TabPFGen*, a novel energy-based model that harnesses the power of TabPFN for tabular tasks in generative modelling. TabPFGen defines a class-conditional energy within the pre-trained TabPFN, and employs the workhorse stochastic gradient Langevin dynamics algorithm [38] for sample generation. Notably, TabPFGen inherits TabPFN's in-context learning capabilities, requiring no additional training or hyperparameter tuning. We conduct experiments on 18 well-established datasets from OpenML-CC18 [4]. Our results show a substantial improvement in downstream model performance through the utilization of TabPFGen for data augmentation, surpassing competitive baselines. Moreover, TabPFGen also proves valuable for class balancing and imputation by producing samples that closely align with the training data distribution, showcasing its exciting potential for tabular data generation in practice.

2 Background & Related Work

TabPFN, introduced by Hollmann et al. [19], is a transformer-based architecture designed for in-context learning of discriminative tabular data tasks. It is trained using a prior-fitting procedure [27] that exposes the network to a massive number of possible inductive biases which may be observed in the tabular setting. After training, the learned TabPFN model accepts training samples $(x_{\text{train}}, y_{\text{train}})$ and test features x_{test}, and yields predictions $\hat{y}_{\text{test}}$ for the entire test set in a single forward pass. In our work, we probabilistically invert these inductive biases in order to generate synthetic data x_{synth} conditioned on synthetic labels y_{synth}.

The field of **generative modeling for tabular data** has witnessed significant advancements. Initially, GAN-based approaches [20, 40, 14, 28] dominated, followed by diffusion models [42, 21] and large language models (LLMs) [5, 32]. However, surprisingly, the simplest interpolation methods such as SMOTE [8, 26] remain competitive [7]. We conjecture that the aforementioned generative techniques may not have adequately captured the inductive biases of successful *discriminative* approaches, undermining their effectiveness. Meanwhile, **generative modelling using discriminators** has expanded mainly outside of the tabular domain. Early work by Tu [34] showed promising results on computer vision tasks, and recent investigations have further demonstrated efficacy in image synthesis [1, 22]. Nock and Guillame-Bert [28] also shed light on this strategy for tabular data synthesis using decision trees.

Energy-based models (EBMs) have also gained significant traction across machine learning domains. For example, Florence et al. [15] utilized EBMs in the context of robot behavioral cloning, while Liu et al. [23] applied an energy score for out-of-distribution detection. It is worth noting that energy scores inherently lack class information; our contribution builds upon the existing work by introducing a class-conditional energy for sample generation. Grathwohl et al. [17] highlighted that any classifier can be treated as an EBM, a concept that has been applied to various image generation works [39, 16, 12, 41]. Notably, our work stands out by generating tabular data while leveraging a pre-trained model without any additional training or hyperparameter tuning.

3 Method

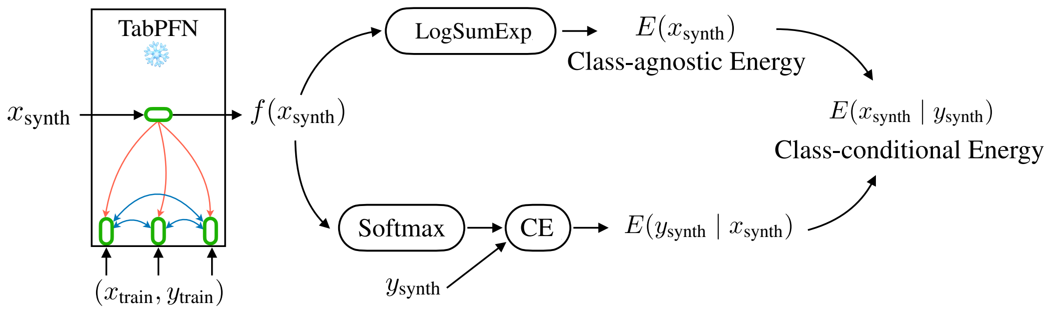


Figure 1: TabPFGen overview. We backpropagate from the class-conditional energy to x_{synth} for gradient generation. CE denotes cross entropy; blue and red arrows represent attention.

We leverage the strong in-context discriminative performance of TabPFN to devise a class-conditional generative model. In particular, given a synthetic label $y_{\text{synth}} \in \{1, \ldots, K\}$, we seek to define a generative model $p(x_{\text{synth}} \mid y_{\text{synth}})$ which can synthesize new samples $x_{\text{synth}} \in \mathbb{R}^D$ while maintaining a link to the classification task solved by TabPFN.

TabPFN, like many classification models, induces a conditional distribution $p(y \mid x)$ given by $\sigma(f(x))[y]$, where x is the network input, y is the label for x, $f : \mathbb{R}^D \to \mathbb{R}^K$ represents the TabPFN, σ is the softmax function, and $[y]$ denotes an indexing operation. The training data $(x_{\text{train}}, y_{\text{train}})$ is also passed to TabPFN, but we omit it in the notation for simplicity.

Next, recalling Bayes' rule, we have $p(x \mid y) \propto p(y \mid x) \cdot p(x)$. To specify $p(x)$ we take inspiration from Grathwohl et al. [17] and employ an energy function $E(x)$ – termed the *class-agnostic energy* – so that $p(x) \propto \exp(-E(x))$:

$$E(x) = -\operatorname{LogSumExp}_{y'}(f(x)[y']) \tag{1}$$

The class-agnostic energy also closely resembles the energy score introduced in Liu et al. [23]. However, since we are interested in class-*conditional* generation, we also require *class-specific* information. To do this, first rewrite $p(y \mid x)$ as

$$p(y \mid x) = \frac{\exp\left(f(x)[y]\right)}{\sum_{y'} \exp\left(f(x)[y']\right)} = \exp\left(f(x)[y] - \text{LogSumExp}_{y'}(f(x)[y'])\right), \tag{2}$$

and thus $p(x \mid y) \propto p(y \mid x) \cdot p(x) \propto \exp(f(x)[y])$, as the LogSumExp terms cancel out. As a result, the *class-conditional energy* can be defined as simply:

$$E(x \mid y) \coloneqq -f(x)[y] \tag{3}$$

We have thus arrived at an energy-based class-conditional generative model $p(x \mid y)$ which relies on TabPFN in a principled manner. As depicted in Figure 1, we first obtain $f(x_{\text{synth}})$ using $(x_{\text{train}}, y_{\text{train}})$ as training data. Then, we can directly compute (3) using $E(x_{\text{synth}} \mid y_{\text{synth}}) = -f(x_{\text{synth}})[y_{\text{synth}}]$, eliminating the need to calculate the class-agnostic energy and cross entropy separately.

Algorithm 1: TabPFGen$_{\text{core}}$: Given TabPFN model f, SGLD step size α, SGLD noise σ, SGLD steps η, manually defined synthetic labels y_{synth}

```
1: Input: $x_{\text{train}}, y_{\text{train}}$
2: Initialize $x^0_{\text{synth}}$                                  ▷ SGLD Initialization; detailed in Appendix C.2
3: for $t \in [1, 2, ..., \eta]$ do
4:     $E(x^t_{\text{synth}} \mid y_{\text{synth}}) = -f(x^t_{\text{synth}} \mid (x_{\text{train}}, y_{\text{train}}))[y_{\text{synth}}]$     ▷ Class-conditional Energy
5:     $x^{t+1}_{\text{synth}} = x^t_{\text{synth}} - \alpha \cdot \frac{\partial E(x^t_{\text{synth}} \mid y_{\text{synth}})}{\partial x^t_{\text{synth}}} + \sigma \cdot \mathcal{N}(0, I)$     ▷ SGLD
6: end for
7: Output: $x^\eta_{\text{synth}}$
```

In order to sample from this energy-based model, we employ the stochastic gradient Langevin dynamics (SGLD) method [38]. We are able to generate a batch of x_{synth} every η steps, taking advantage of their independence. The choice of η is determined based on the classification performance on x_{train} using a frozen TabPFN. This method is referred to as TabPFGen$_{\text{core}}$ and is detailed in Algorithm 1.

To further enhance the core methodology, we propose an alternative configuration by switching $(x_{\text{synth}}, y_{\text{synth}})$ and $(x_{\text{train}}, y_{\text{train}})$ in e.g. Figure 1, and integrating the resulting class-conditional energy $E(x_{\text{train}} \mid y_{\text{train}})$ – which implicitly depends on $(x_{\text{synth}}, y_{\text{synth}})$ – into $E(x_{\text{synth}} \mid y_{\text{synth}})$. We posit this modification introduces a regularization effect, although we leave the proof of this to future work. Empirically, this extended approach exhibits slightly superior performance and greater stability. An ablation analysis can be found in Appendix D.2. The configuration details are detailed in Appendix C.2. We henceforth denote the full approach as *TabPFGen*.

4 Experiments & Analysis

We conduct a comprehensive set of experiments utilizing the 18 numerical datasets without missing values, which are previouly employed by Hollmann et al. [19]. These datasets are obtained from the OpenML-CC18 suite [4] and their specifics are outlined in Appendix B.

Experimental Setup: We partition each dataset into equal-sized training and test sets, and then train or utilize generative models to synthesize samples given the training data. To evaluate the efficacy of a generative model we use its synthetic samples to either replace, augment, or class-balance the training data, then train a variety of discriminative models to predict the class label and evaluate the AUC (Area Under the ROC Curve) performance on the test set. We present the mean and standard deviation of the above evaluation process over three runs with unique seeds.

Downstream Models: For all experiments we train and evaluate 4 distinct downstream models: *XGBoost* [9], *random forest* [18], *logistic regression*, and *TabPFN* [19]. To ensure fair comparisons we adopt the optimized hyperparameters of downstream models previously published by Hollmann et al. [19], if available. Experiments with a range of alternative hyperparameter configurations consistently demonstrate that TabPFGen outperforms baseline methods, as detailed in Appendix D.5.

Baseline Models: We employ a diverse set of baseline generative models, including the traditional interpolation approach SMOTE [8], generative adversarial networks represented by CTGAN [40], variational autoencoder-based methods including TVAE [40] and RTVAE [2], the normalizing flow-based Neural Spline Flows (NF) [13], and the recent diffusion-based methods TabDDPM [21]. All baseline generative models are trained and evaluated using the publicly available `synthcity` package [29]. The details and hyperparameters of each generative model can be found in Appendix C.1.

4.1 Results

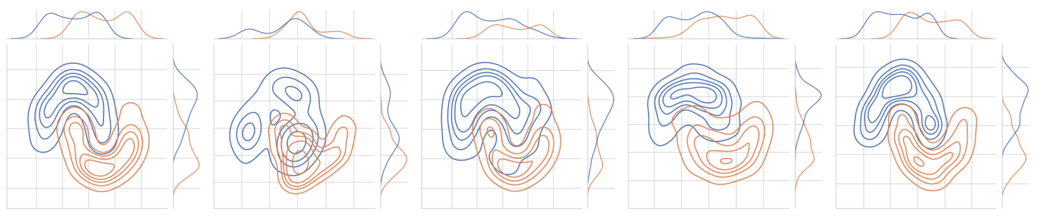
Replacement & Augmentation: We assess the quality of synthetic samples from a given model through two tasks: *augmentation*, which augments the training dataset with an equal volume and class-ratio of synthetic data, and *replacement*, which replaces the original training set with synthetic data. The replacement task assesses whether the generated samples closely resemble the training data, while the augmentation task assesses whether adding synthetic samples improves classification performance. Table 1 displays the downstream model performance on the test sets when utilizing synthetic samples from various generative models during training. We find that TabPFGen facilitates a significant performance enhancement over other generative models – TabPFGen is the only generative model that consistently improves downstream performance when augmenting the original training set. For instance, with XGBoost, the average AUC (0.936) achieved is better than a model trained on the original data (0.924). Individual results on each of the 18 datasets can be found in Appendix D.4. TabPFGen also outperforms all other generative methods on the replacement task, indicating that its generated samples most closely resemble the training data. We highlight that TabPFGen requires no training or fine-tuning, and no hyperparameter tuning to adapt to specific datasets.

Table 1: Average AUC over OpenML-CC18 test sets, with error bars over 3 runs. Top 4 rows: synthetic data as augmentation. Bottom 4 rows: synthetic data as replacement.

Model	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
XGB	$0.924_{\pm 3e-4}$	$0.926_{\pm 3e-4}$	$0.912_{\pm 2e-4}$	$0.914_{\pm 7e-4}$	$0.912_{\pm 4e-4}$	$0.917_{\pm 3e-4}$	$0.927_{\pm 3e-4}$	$\mathbf{0.934}_{\pm 3e-4}$
RF	$0.906_{\pm 3e-4}$	$0.906_{\pm 2e-3}$	$0.898_{\pm 1e-3}$	$0.904_{\pm 1e-3}$	$0.894_{\pm 3e-4}$	$0.907_{\pm 2e-3}$	$0.911_{\pm 7e-4}$	$\mathbf{0.912}_{\pm 4e-4}$
LR	$0.920_{\pm 7e-4}$	$0.914_{\pm 3e-3}$	$0.904_{\pm 3e-3}$	$0.909_{\pm 6e-3}$	$0.901_{\pm 9e-4}$	$0.906_{\pm 8e-3}$	$0.885_{\pm 3e-4}$	$\mathbf{0.921}_{\pm 2e-4}$
TabPFN	$0.934_{\pm 2e-3}$	$0.927_{\pm 1e-3}$	$0.930_{\pm 1e-3}$	$0.931_{\pm 1e-3}$	$0.928_{\pm 3e-4}$	$0.932_{\pm 1e-3}$	$0.929_{\pm 5e-4}$	$\mathbf{0.935}_{\pm 3e-4}$
XGB	N/A	$0.907_{\pm 4e-4}$	$0.842_{\pm 8e-4}$	$0.858_{\pm 2e-3}$	$0.700_{\pm 6e-4}$	$0.795_{\pm 9e-4}$	$0.812_{\pm 3e-4}$	$\mathbf{0.927}_{\pm 3e-4}$
RF	N/A	$0.894_{\pm 1e-3}$	$0.837_{\pm 6e-4}$	$0.844_{\pm 5e-4}$	$0.676_{\pm 2e-3}$	$0.774_{\pm 3e-4}$	$0.814_{\pm 9e-4}$	$\mathbf{0.906}_{\pm 6e-4}$
LR	N/A	$0.893_{\pm 2e-3}$	$0.843_{\pm 6e-4}$	$0.873_{\pm 1e-3}$	$0.722_{\pm 3e-3}$	$0.854_{\pm 7e-4}$	$0.876_{\pm 3e-4}$	$\mathbf{0.920}_{\pm 1e-3}$
TabPFN	N/A	$0.920_{\pm 8e-4}$	$0.888_{\pm 4e-4}$	$0.887_{\pm 3e-4}$	$0.705_{\pm 2e-3}$	$0.862_{\pm 1e-3}$	$0.894_{\pm 7e-4}$	$\mathbf{0.934}_{\pm 2e-4}$

Table 2: Average AUC using synthetic data for class balancing. Error bars can be found in the Appendix. These datasets are the five most class-imbalanced datasets in the OpenML-CC18 suite.

Dataset	Original	Sampling	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
KC	0.823	0.875	0.872	0.859	0.848	0.862	0.866	0.805	**0.877**
PC	0.824	0.811	0.836	0.827	0.835	0.825	**0.841**	0.825	**0.841**
BL	0.731	0.757	0.756	0.743	0.714	0.755	0.706	**0.774**	0.767
CL	0.925	0.935	0.949	0.793	0.771	0.795	0.909	0.915	**0.955**
DI	0.837	0.832	0.832	0.831	0.813	0.832	0.837	0.843	**0.844**



(a) Original (b) CTGAN (c) NF (d) TabDDPM (e) TabPFGen

Figure 2: Contour and marginal density plots of: (a) original two-moons dataset; (b)-(d) synthetic data generated using baseline methods; (e) synthetic data generated by TabPFGen

Class Balancing: We create class-balanced datasets by generating synthetic data for minority classes until every class has an equal number of samples. We then train an XGBoost model on the class-balanced data, and report its test AUC in Table 2. We find that TabPFGen generally outperforms alternative methods, including even class-balancing the original data by sampling with replacement, which we denote “sampling”. Additional details and an expanded table can be found in Appendix D.3.

Imputation: Promising results on imputation using TabPFGen are shown in Appendix D.1.

Qualitative Assessment: The above analyses demonstrate that TabPFN’s generated samples have a high degree of utility for downstream applications. As a further confirmation of sample quality we display contour and the marginal plots for generative models on the popular two-moons dataset in Figure 2. We show that the synthetic data distribution generated by TabPFGen is more similar to the original data distribution than other baseline methods. Surprisingly, we find that some generative models perform poorly even on such a simple dataset.

5 Conclusion

In this work, we present TabPFGen, an efficient approach harnessing a pre-trained TabPFN as an energy based model for tabular data generation without additional training. TabPFGen outperforms competitive baselines across augmentation, class balancing, and imputation tasks. Nonetheless, it is essential to recognize that TabPFN’s current limitations, such as input size constraints and a focus on numerical datasets, limits our method’s current applicability for large-scale datasets. We continue this discussion in Appendix A, but anticipate that these limitations will gradually recede as transformers and extensions of TabPFN advance, and argue that re-purposing powerful pretrained discriminative models for tabular generation is an impactful and promising avenue for future research.

References

[1] Image synthesis with a single (robust) classifier. In *Santurkar, Shibani and Ilyas, Andrew and Tsipras, Dimitris and Engstrom, Logan and Tran, Brandon and Madry, Aleksander*, 2019.

[2] Haleh Akrami, Anand A. Joshi, Jian Li, Sergül Aydöre, and Richard M. Leahy. A robust variational autoencoder using beta divergence. *Knowledge-Based Systems*, 238:107886, 2022.

[3] Omar Benjelloun, Shiyu Chen, and Natasha Noy. Google dataset search by the numbers. In *The Semantic Web – ISWC 2020*, pages 667–682, 2020.

[4] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael Gomes Mantovani, Jan van Rijn, and Joaquin Vanschoren. OpenML benchmarking suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[5] Vadim Borisov, Kathrin Sessler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators. In *International Conference on Learning Representations*, 2022.

[6] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[7] Ramiro D. Camino, Radu State, and Christian A. Hammerschmidt. Oversampling tabular data with deep generative models: Is it worth the effort? In *Proceedings on "I Can’t Believe It’s Not Better!" at NeurIPS Workshops*, pages 148–157, 2020.

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16: 321–357, 2002.

[9] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SigKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[10] Jillian M Clements, Di Xu, Nooshin Yousefi, and Dmitry Efimov. Sequential deep learning for credit risk monitoring with tabular financial data. *arXiv preprint arXiv:2012.15330*, 2020.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[12] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. In *Advances in Neural Information Processing Systems*, 2019.

[13] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *Advances in Neural Information Processing systems*, 2019.

[14] Justin Engelmann and Stefan Lessmann. Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications*, 174:114582, 2021.

[15] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168, 2022.

[16] Ruiqi Gao, Yang Lu, Junpei Zhou, Song-Chun Zhu, and Ying Nian Wu. Learning energy-based models as generative ConvNets via multi-grid modeling and sampling. *arXiv preprint arXiv:1709.08868*, 2017.

[17] Will Grathwohl, Kuan-Chieh Wang, Joern-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. In *International Conference on Learning Representations*, 2020.

[18] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 278–282, 1995.

[19] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations*, 2023.

[20] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2018.

[21] 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, 2023.

[22] Haoyang Li. Use classifier as generator. *arXiv preprint arXiv:2209.09210*, 2022.

[23] Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems*, pages 21464–21475, 2020.

[24] Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pages 4282–4292, 2019.

[25] Dionysis Manousakas and Sergül Aydöre. On the usefulness of synthetic tabular data generation. In *Data-centric Machine Learning Research (DMLR) Workshop at the 40th International Conference on Machine Learning*, 2023.

[26] Ajinkya More. Survey of resampling techniques for improving classification performance in unbalanced datasets. *arXiv preprint arXiv:1608.06048*, 2016.

[27] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. In *International Conference on Learning Representations*, 2022.

[28] Richard Nock and Mathieu Guillame-Bert. Generative trees: Adversarial and copycat. In *International Conference on Machine Learning*, pages 16906–16951, 2022.

[29] Zhaozhi Qian, Bogdan-Constantin Cebere, and Mihaela van der Schaar. Synthcity: Facilitating innovative use cases of synthetic data in different data modalities. *arXiv preprint arXiv:2301.07573*, 2023.

[30] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[31] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[32] Aivin V. Solatorio and Olivier Dupriez. REaLTabFormer: Generating realistic relational and tabular data using transformers. *arXiv preprint arXiv:2302.02041*, 2023.

[33] Qi Tang, Guoen Xia, Xianquan Zhang, and Feng Long. A customer churn prediction model based on XGBoost and MLP. In *2020 International Conference on Computer Engineering and Application (ICCEA)*, pages 608–612, 2020.

[34] Zhuowen Tu. Learning generative models via discriminative approaches. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[35] Dennis Ulmer, Lotta Meijerink, and Giovanni Cinà. Trust issues: Uncertainty estimation does not enable reliable OOD detection on medical tabular data. In *Machine Learning for Health*, pages 341–354, 2020.

[36] Christopher J Urban and Kathleen M Gates. Deep learning: A primer for psychologists. *Psychological Methods*, 26(6):743, 2021.

[37] Arash Vahdat and Jan Kautz. NVAE: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, 2020.

[38] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *International Conference on Machine Learning*, pages 681–688, 2011.

[39] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative ConvNet. In *International Conference on Machine Learning*, pages 2635–2644, 2016.

[40] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *Advances in Neural Information Processing Systems*, 2019.

[41] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via GradInversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16337–16346, 2021.

[42] Shuhan Zheng and Nontawat Charoenphakdee. Diffusion models for missing value imputation in tabular data. In *NeurIPS 2022 First Table Representation Workshop*, 2022.

A Limitations and Impact

Our method builds upon TabPFN, and consequently inherits certain limitations. Presently, TabPFN imposes restrictions on input size, allowing for a maximum of 2000 tokens, 100 features, and 10 classes. These constraints arise from the quadratic complexity inherent in the underlying transformer architecture. It is important to note that TabPFN and TabPFGen algorithms are designed to be architecture-agnostic, therefore the improvement of transformer architectures will directly translate to our method. As the research of transformer architectures continues to advance, we anticipate these constraints will diminish.

B Datasets Details

Our experiments are conducted using the 18 numerical datasets from OpenML-CC18 [4] (available at `http://openml.org`). Similar to Hollmann et al. [19], we use datasets with maximum 2000 samples, 100 numerical features, and 10 classes, without missing values. The details of the datasets are listed in Table 3.

Table 3: 18 datasets from OpenML-CC18

Name	#Feat.	#Inst.	#Class.	Minor. Class Size	OpenML ID
balance-scale	5	625	3	49	11
mfeat-fourier	77	2000	10	200	14
mfeat-karhunen	65	2000	10	200	16
mfeat-morphological	7	2000	10	200	18
mfeat-zernike	48	2000	10	200	22
diabetes	9	768	2	268	37
vehicle	19	846	4	199	54
analcatdata_auth...	71	841	4	55	458
pc4	38	1458	2	178	1049
pc3	38	1563	2	160	1050
kc2	22	522	2	107	1063
pc1	22	1109	2	77	1068
banknote-authenti...	5	1372	2	610	1462
blood-transfusion-...	5	748	2	178	1464
qsar-biodeg	42	1055	2	356	1494
wdbc	31	569	2	212	1510
steel-plates-fault	28	1941	7	55	40982
climate-model-simu...	21	540	2	46	40994

C Experimental Details

C.1 Baseline Details

We use the `synthcity` package [29] to run all of our baselines. The `synthcity` code repository can be found at: `https://github.com/vanderschaarlab/synthcity`. We keep the same hyperparameters as Manousakas and Aydöre [25] when possible. The exact hyperparameters and training details can be found in Table 13.

C.2 TabPFGen Details

We use a pre-trained TabPFN for all of our experiments. The pre-trained TabPFN weights are available for download from the following link: `https://github.com/automl/TabPFN/raw/main/tabpfn/models_diff/prior_diff_real_checkpoint_n_0_epoch_42.cpkt`.

We first initialize x_{synth} using the training data x_{train} injected with Gaussian noise. We fix the mean of the noise to be 0 and standard deviation to be 0.01 throughout all of our experiments. We then perform SGLD for η steps guided by the classification AUC on x_{train} with x_{synth} as in-context training data. Since TabPFN accepts independent inputs, we are able to generate a batch of independent samples for each iteration in the SGLD process. Therefore, we return the best batch of x_{synth} at the end of the SGLD process.

D Additional Results

D.1 Imputation with TabPFGen

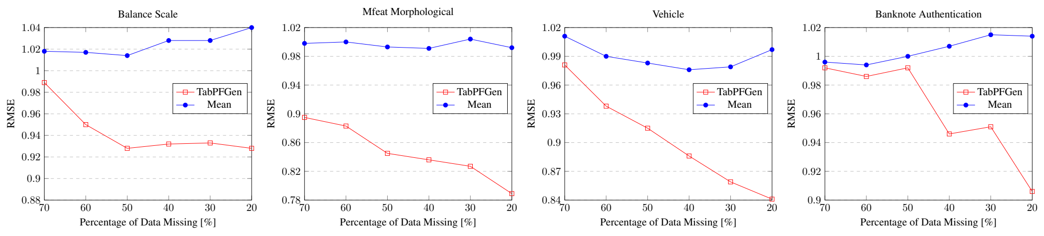


Figure 3: Imputation results. TabPFGen consistently has lower RMSE than the baseline method and also has decreasing RMSE with a decreasing fraction of missing data.

We conduct imputation analysis on 4 randomly selected datasets including: balance scale, mfeat morphological, vehicle, and banknote authentication. Details of the datasets can be found in Appendix B. For all datasets, we uniform randomly mask out a certain percentage of the entries in the table as NaNs. For the baseline method, we simply use the mean of the entire column as the predicted value.

For our method, we initialize the missing entries as the mean value of the feature. Then we start sampling with TabPFGen on x_{synth}, but we fix the values of the non-missing features. Finally, we measure the RMSE between the original values and the sampled values for all missing entries. As shown in Figure 3, for all datasets, TabPFGen results in lower RMSE than using the mean feature value as imputation. We can also observe that TabPFGen is able to reduce RMSE further as the percentage of data missing decreases, while the mean imputation method does not improve. Even though these experiments do not compare to a robust set of baselines, they nevertheless demonstrate that using TabPFGen for imputation is a fruitful direction for further study.

D.2 Ablation Studies

Table 4: Average AUC over OpenML-CC18 with error bars over 3 runs. Results shown across different TabPFGen variants. Top 4 rows: synthetic data as augmentation. Bottom 4 rows: synthetic data as replacement.

Model	TabPFGen$_{\text{core}}$	TabPFGen
XGB	$\mathbf{0.934}\pm5e{-}4$	$\mathbf{0.934}\pm3e{-}4$
RF	$\mathbf{0.913}\pm9e{-}4$	$0.912\pm4e{-}4$
LR	$0.919\pm6e{-}4$	$\mathbf{0.921}\pm2e{-}4$
TabPFN	$\mathbf{0.937}\pm3e{-}4$	$0.935\pm3e{-}4$
XGB	$\mathbf{0.927}\pm1e{-}3$	$\mathbf{0.927}\pm3e{-}4$
RF	$\mathbf{0.906}\pm7e{-}4$	$\mathbf{0.906}\pm6e{-}4$
LR	$0.918\pm6e{-}4$	$\mathbf{0.920}\pm1e{-}3$
TabPFN	$\mathbf{0.934}\pm1e{-}3$	$\mathbf{0.934}\pm2e{-}4$

We conduct ablation studies over the 2 TabPFGen variants in Table 4, recalling that the Core variant is the basic technique outlined in Algorithm 1, and the full TabPFGen variant also includes the swapping of $(x_{\text{synth}}, y_{\text{synth}})$ and $(x_{\text{train}}, y_{\text{train}})$ as described previous. The top 4 rows indicate the performance of different TabPFGen variants when augmenting the training set with synthetic samples, and the bottom 4 rows show the downstream model performance when trained only on synthetic data. We see that the Core variant is generally competitive with the full TabPFGen – even outperforming in some instances – but we decided to use the full TabPFGen as the main approach because of its performance on the replacement task.

Table 5: Average AUC using synthetic data for class balancing. Error bars shown over 3 runs.

	Original	Sampling	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
KC	$0.823_{\pm 1e-3}$	$0.875_{\pm 4e-4}$	$0.872_{\pm 2e-4}$	$0.859_{\pm 7e-4}$	$0.848_{\pm 6e-4}$	$0.862_{\pm 9e-4}$	$0.866_{\pm 2e-3}$	$0.805_{\pm 8e-4}$	$\mathbf{0.877}_{\pm 6e-4}$
PC	$0.824_{\pm 4e-3}$	$0.811_{\pm 4e-3}$	$0.836_{\pm 4e-4}$	$0.827_{\pm 4e-4}$	$0.835_{\pm 5e-3}$	$0.825_{\pm 1e-3}$	$\mathbf{0.841}_{\pm 5e-4}$	$0.825_{\pm 5e-3}$	$\mathbf{0.841}_{\pm 8e-4}$
BL	$0.731_{\pm 4e-4}$	$0.757_{\pm 4e-3}$	$0.756_{\pm 3e-3}$	$0.743_{\pm 7e-4}$	$0.714_{\pm 4e-4}$	$0.755_{\pm 5e-4}$	$0.706_{\pm 4e-3}$	$\mathbf{0.774}_{\pm 4e-4}$	$0.767_{\pm 4e-4}$
CL	$0.925_{\pm 5e-4}$	$0.935_{\pm 3e-3}$	$0.949_{\pm 1e-3}$	$0.793_{\pm 4e-4}$	$0.771_{\pm 4e-4}$	$0.795_{\pm 4e-4}$	$0.909_{\pm 4e-4}$	$0.915_{\pm 5e-4}$	$\mathbf{0.955}_{\pm 3e-3}$
DI	$0.837_{\pm 4e-3}$	$0.832_{\pm 4e-4}$	$0.832_{\pm 7e-4}$	$0.831_{\pm 5e-4}$	$0.813_{\pm 5e-4}$	$0.832_{\pm 3e-3}$	$0.837_{\pm 5e-4}$	$0.843_{\pm 4e-4}$	$\mathbf{0.844}_{\pm 4e-3}$

Table 6: Information of datasets used for class-balancing experiments

Dataset	Full Name	MinMaj
KC	kc2	0.26
PC	pc3	0.11
BL	blood-transfusion-service-center	0.31
CL	climate-model-simulation-crashes	0.09
DI	diabetes	0.54

D.3 Additional Class Balancing Experiment Details

Table 5 shows class balancing results with multiple runs for each experimental setup. The datasets KC, PC, BL, CL, and DI represent kc2, pc3, blood-transfusion-service-center, climate-model-simulation-crashes, and diabetes datasets, respectively. All of them are chosen from the OpenML-CC18 suite. We find that TabPFGen generally outperforms baseline methods, with only TabDDPM on blood-transfusion outperforming TabPFGen throughout the entire suite of experiments.

Table 6 shows the full name for each dataset along with the ratio of minority class size to majority class size (MinMaj).

D.4 Synthetic Data as Augmentation Details

Table 7: Detailed "synthetic data as augmentation" experiment using XGBoost

Dataset	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
balance-scale	**0.9939**	0.9931	0.8374	0.8236	0.8135	0.8236	0.9620	0.9931
mfeat-fourier	**0.9803**	0.9801	0.9761	0.9773	0.9754	0.9770	0.9768	**0.9803**
mfeat-karhunen	0.9983	**0.9989**	0.9977	0.9974	0.9961	0.9977	0.9977	0.9985
mfeat-morphological	0.9612	0.9608	0.9574	0.9556	0.9562	0.9600	0.9612	**0.9613**
mfeat-zernike	0.9735	**0.9749**	0.9729	0.9731	0.9725	0.9733	0.9721	0.9740
diabetes	0.8378	0.8109	0.8390	0.8273	0.8384	**0.8401**	0.8295	0.8378
vehicle	**0.9282**	**0.9282**	0.9219	0.9248	0.9147	0.9074	0.9234	0.9272
analcatdata_authorship	0.9997	**1.0000**	0.9999	0.9999	0.9997	0.9998	0.9999	0.9999
pc4	0.9291	0.9291	0.9245	0.9259	0.9209	0.9230	0.9279	**0.9351**
pc3	0.8288	0.8261	0.8187	0.8251	0.8304	0.8323	0.8279	**0.8392**
kc2	0.8227	0.8567	0.8760	0.8762	0.8793	**0.8831**	0.8801	0.8716
pc1	0.8489	0.8683	0.8428	0.8465	0.8602	0.8556	0.8800	**0.8971**
banknote-authentication	**1.0000**	0.9999	0.9990	0.9974	0.9990	0.9997	**1.0000**	**1.0000**
blood-transfusion-service-center	0.7312	0.7495	0.7221	0.7662	0.7626	0.7580	**0.7651**	0.7579
qsar-biodeg	0.9191	0.9151	0.9120	0.9091	0.9078	0.9113	0.9181	**0.9212**
wdbc	**0.9904**	0.9800	0.9512	0.9491	0.9687	0.9674	0.9798	0.9855
steel-plates-fault	0.9656	0.9653	0.9597	**0.9668**	0.9627	0.9604	0.9585	0.9654
climate-model-simulation-crashes	0.9255	0.9447	0.9081	0.9106	0.8586	0.9389	0.9347	**0.9586**

Table 8: Detailed "synthetic data as augmentation experiment" using Random Forest.

Dataset	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
balance-scale	0.8001	0.8142	0.7491	0.7717	0.7125	0.7717	0.7990	**0.8294**
mfeat-fourier	0.9794	**0.9806**	0.9766	0.9774	0.9775	0.9765	0.9793	0.9785
mfeat-karhunen	0.9968	**0.9974**	0.9955	0.9970	0.9946	0.9963	0.9971	0.9973
mfeat-morphological	0.9509	0.9482	0.9471	0.9459	0.9449	0.9510	**0.9529**	0.9502
mfeat-zernike	**0.9714**	0.9672	0.9662	0.9664	0.9677	0.9684	0.9694	0.9707
diabetes	0.8159	0.8020	0.8068	0.7813	0.8087	0.8070	0.8209	**0.8307**
vehicle	0.9188	0.9153	0.9155	0.9183	0.9109	0.9167	**0.9237**	0.9175
analcatdata_authorship	0.9998	**0.9999**	0.9998	**0.9999**	0.9996	**0.9999**	0.9995	**0.9999**
pc4	0.9220	0.9225	0.9131	0.9213	0.9145	0.9166	**0.9277**	0.9261
pc3	0.8047	0.8173	0.8101	0.8090	0.8085	**0.8318**	0.7906	0.8200
kc2	0.8348	**0.8594**	0.8259	0.8497	0.8484	0.8230	0.8377	0.8348
pc1	0.8853	0.8851	0.8740	0.9063	0.8902	**0.9066**	0.8919	0.8867
banknote-authentication	0.9996	0.9998	0.9992	0.9992	**1.0000**	0.9994	**1.0000**	0.9998
blood-transfusion-service-center	0.7016	0.6914	0.6868	0.7081	0.6905	0.7136	**0.7261**	0.6984
qsar-biodeg	0.9158	0.9156	0.9076	0.9106	0.9076	0.9072	**0.9198**	0.9158
wdbc	0.9838	0.9856	0.9837	0.9897	0.9844	**0.9906**	0.9860	0.9802
steel-plates-fault	0.9577	**0.9601**	0.9579	0.9600	0.9579	0.9584	0.9578	0.9574
climate-model-simulation-crashes	0.8758	0.8580	0.8581	0.8614	0.7908	0.8951	**0.9301**	0.9261

Table 9: Detailed "synthetic data as augmentation experiment" using Logistic Regression.

Dataset	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
balance-scale	**0.9574**	0.9568	0.8933	0.8778	0.8517	0.8778	0.9556	0.9559
mfeat-fourier	0.9631	0.9612	0.9611	**0.9636**	0.9605	0.9601	0.9570	0.9584
mfeat-karhunen	0.9933	0.9930	0.9903	0.9895	0.9885	0.9907	0.9863	**0.9948**
mfeat-morphological	**0.9664**	0.9654	0.9578	0.9538	0.9534	0.9526	0.9629	0.9660
mfeat-zernike	**0.9759**	0.9750	0.9754	0.9757	0.9708	0.9744	0.9645	0.9757
diabetes	0.8384	0.8292	0.8330	0.8304	**0.8440**	0.8255	0.8132	0.8365
vehicle	0.9369	**0.9404**	0.8914	0.8917	0.8607	0.8803	0.7469	0.9388
analcatdata_authorship	0.9999	0.9999	0.9832	0.9909	0.9976	0.9782	0.9989	**0.9999**
pc4	0.8880	0.8869	**0.8922**	0.8743	0.8597	0.8615	0.8500	0.8830
pc3	**0.8201**	0.8010	0.7649	0.7986	0.8031	0.8129	0.7144	0.8162
kc2	0.8524	0.8084	0.8640	0.8428	**0.8781**	0.8497	0.8283	0.8727
pc1	**0.8233**	0.8015	0.7731	0.8393	0.8072	0.8185	0.7827	0.8224
banknote-authentication	**0.9999**	**0.9999**	0.9981	0.9983	0.9986	0.9918	**0.9999**	**0.9999**
blood-transfusion-service-center	0.7616	0.7610	0.7487	**0.7682**	0.7669	0.7520	0.7610	0.7639
qsar-biodeg	0.9060	0.9046	0.9037	0.9055	0.8789	0.9024	0.8664	**0.9073**
wdbc	0.9834	0.9836	0.9832	0.9805	0.9873	**0.9896**	0.9729	0.9841
steel-plates-fault	0.9375	0.9374	0.9189	0.9295	0.9175	0.9247	**0.9500**	0.9377
climate-model-simulation-crashes	0.9621	0.9544	0.9453	0.9581	0.9037	**0.9687**	0.8192	0.9610

Table 10: Detailed "synthetic data as augmentation experiment" using TabPFN.

Dataset	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
balance-scale	**0.9984**	0.9983	0.9510	0.9582	0.9508	0.9582	0.9792	0.9955
mfeat-fourier	**0.9810**	0.9804	0.9783	0.9774	0.9764	0.9773	0.9805	0.9802
mfeat-karhunen	0.9980	0.9979	0.9976	0.9978	0.9975	0.9976	**0.9986**	0.9978
mfeat-morphological	**0.9646**	0.9637	0.9644	0.9627	0.9638	0.9630	0.9638	0.9638
mfeat-zernike	0.9818	0.9817	0.9806	0.9807	0.9816	0.9810	**0.9819**	**0.9817**
diabetes	0.8428	0.7766	0.8423	0.8307	0.8418	0.8308	0.8224	0.8138
vehicle	0.9568	0.9516	0.9419	0.9477	0.9467	0.9450	**0.9541**	0.9523
analcatdata_authorship	**1.0000**	**1.0000**	**1.0000**	0.9999	0.9999	0.9999	0.9999	0.9999
pc4	0.9233	0.9283	0.9320	0.9326	0.9254	0.9306	0.9146	**0.9329**
pc3	0.8481	0.8183	0.8396	0.8449	0.8369	0.8516	0.8365	**0.8533**
kc2	0.8777	0.8565	0.8816	0.8701	0.8779	0.8615	**0.8831**	0.8750
pc1	**0.8919**	0.8782	0.8709	0.8864	0.8753	0.8888	0.8767	0.8903
banknote-authentication	**1.0000**	**1.0000**	0.9997	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
blood-transfusion-service-center	0.7590	0.7277	0.7444	0.7599	0.7601	0.7622	**0.7687**	0.7423
qsar-biodeg	0.9306	0.9242	0.9291	0.9220	0.9242	0.9214	0.9287	**0.9293**
wdbc	0.9904	0.9909	0.9897	0.9883	0.9922	0.9896	**0.9933**	0.9892
steel-plates-fault	0.9634	0.9623	0.9596	0.9613	0.9623	0.9631	0.9627	0.9636
climate-model-simulation-crashes	0.9625	0.9563	0.9354	0.9343	0.8915	0.9546	0.8946	**0.9649**

In Table 7, 8, 9 and 10, we show the performance of TabPFGen and baseline methods on each of the 18 datasets individually. We display the average values over three runs, but omit error bars to preserve space. We find that TabPFGen outperforms the baseline methods on the majority of the datasets. Also notably, SMOTE performs quite well even when compared against modern deep learning methods, which is consistent with the recent work from Manousakas and Aydöre [25].

D.5 Experiments with Different Hyperparameters of Downstream Models

Table 11: Synthetic data as augmentation with various downstream model hyperparameters. Bolding is done here just to highlight the best result up to three significant figures – only one run of each setting was conducted.

Model	Original	SMOTE	CTGAN	TVAE	NF	RTVAE	TabDDPM	TabPFGen
XGB 0	0.903	0.897	0.901	0.897	0.891	0.903	0.909	**0.914**
RF 0	0.902	0.895	0.900	0.895	0.892	0.900	0.903	**0.920**
LR 0	0.888	0.894	0.884	0.886	0.884	0.888	0.871	**0.905**
XGB 1	0.906	0.902	0.903	0.904	0.896	0.906	0.911	**0.915**
RF 1	0.908	0.902	0.897	0.897	0.891	0.904	0.911	**0.919**
LR 1	**0.920**	0.913	0.901	0.908	0.905	0.905	0.885	**0.920**
XGB 2	0.915	0.914	0.906	0.909	0.909	0.911	0.914	**0.928**
RF 2	0.906	0.905	0.902	0.898	0.897	0.905	0.911	**0.922**
LR 2	0.868	0.870	0.863	0.865	0.866	0.869	0.852	**0.885**
XGB 3	0.894	0.897	0.864	0.862	0.853	0.871	0.881	**0.912**
RF 3	0.909	0.909	0.904	0.903	0.900	0.908	0.914	**0.921**
LR 3	0.897	0.894	0.866	0.898	0.882	0.893	0.871	**0.922**

We also experiment with different hyperparameters of the downstream models to verify the robustness of our approach to the specific choice of downstream model, ensuring that the superior performance of TabPFGen over other generative models is not sensitive to such choices. Table 11 shows the results of the "synthetic data as augmentation" experiment with 4 different sets of hyperparameters each for downstream XGBoost (XGB), random forest (RF), and logistic regression (LR) models. We have found that TabPFGen has the best result in all of the hyperparameter settings, showing that the results presented in the main text are robust to such choices. The parameters are randomly chosen using ParameterSampler from the `scikit-learn` library to avoid biasing the algorithm. We list the different hyperparameters we have used in Table 12.

Table 12: Different downstream model hyperparameters for Table 11.

	XGB	RF	LR
`Set 0`	alpha: 1.274e-10, colsample_bylevel: 0.960, colsample_bytree: 0.785, gamma: 5.737e-07, lambda: 3.493e-14, learning_rate: 1.235e-06, max_depth: 8, min_child_weight: 3.737e-11, n_estimators: 971, subsample: 0.766	max_depth: 39, max_features: sqrt, min_samples_split: 5, n_estimators: 34	C: 1.362e-03, fit_intercept: True, max_iter: 320, penalty: l2
`Set 1`	alpha: 2.165e-16, colsample_bylevel: 0.975, colsample_bytree: 0.865, gamma: 2.892e-13, lambda: 9.203e-14, learning_rate: 1.922e-06, max_depth: 6, min_child_weight: 1.312e-16, n_estimators: 956, subsample: 0.432	max_depth: 43, max_features: sqrt, min_samples_split: 5, n_estimators: 40	C: 0.2776, fit_intercept: True, max_iter: 152, penalty: none
`Set 2`	alpha: 9.415e-07, colsample_bylevel: 0.311, colsample_bytree: 0.433, gamma: 9.377e-11, lambda: 2.719e-09, learning_rate: 0.0313, max_depth: 3, min_child_weight: 2.436e-10, n_estimators: 343, subsample: 0.673	max_depth: 39, max_features: sqrt, min_samples_split: 5, n_estimators: 94	C: 7.744e-05, fit_intercept: True, max_iter: 137, penalty': l2
`Set 3`	alpha: 5.717e-16, colsample_bylevel: 0.686, colsample_bytree: 0.336, gamma: 1.149e-15, lambda: 0.293, learning_rate: 0.574, max_depth: 2, min_child_weight: 2.729e-10, n_estimators: 445, subsample: 0.278	max_depth: 11, max_features: sqrt, min_samples_split: 5, n_estimators: 119	C: 0.0266, fit_intercept: False, max_iter: 180, penalty: none

Table 13: Hyperparameters of baseline models.

Model	Hyperparameters

`TVAE`	n_units_embedding=500,
	lr=5e-4,
	weight_decay=1e-5,
	batch_size=1000,
	decoder_n_layers_hidden=2,
	decoder_n_units_hidden=256,
	decoder_nonlin="leaky_relu",
	decoder_dropout=0.1,
	encoder_n_layers_hidden=3,
	encoder_n_units_hidden=256,
	encoder_nonlin="leaky_relu",
	encoder_dropout=0.1,
	loss_factor=1,
	data_encoder_max_clusters=10,
	clipping_value=1,
	sampling_patience=500

`RTVAE`	n_units_embedding=500,
	lr=0.001,
	weight_decay=1e-5,
	batch_size=200,
	decoder_n_layers_hidden=3,
	decoder_n_units_hidden=500,
	decoder_nonlin="leaky_relu",
	decoder_dropout=0,
	encoder_n_layers_hidden=3,
	encoder_n_units_hidden=500,
	encoder_nonlin="leaky_relu",
	encoder_dropout=0.1,
	data_encoder_max_clusters=10,
	robust_divergence_beta=2,
	clipping_value=1,
	sampling_patience=500

`CTGAN`	generator_n_layers_hidden=2,
	generator_n_units_hidden=256,
	generator_nonlin="relu",
	generator_dropout=0.1,
	generator_opt_betas=(0.9, 0.999),
	discriminator_n_layers_hidden=2,
	discriminator_n_units_hidden=256,
	discriminator_nonlin="leaky_relu",
	discriminator_n_iter=1,
	discriminator_dropout=0.1,
	discriminator_opt_betas=(0.9, 0.999),
	lr=5e-4,
	weight_decay=1e-3,
	batch_size=1000,
	clipping_value=1,
	lambda_gradient_penalty=10,
	encoder_max_clusters=10,
	sampling_patience=500

`NF`	n_layers_hidden=2,
	n_units_hidden=256,
	batch_size=1000,
	num_transform_blocks=1,
	dropout=0.1,
	batch_norm=False,
	num_bins=8,
	tail_bound=3,
	lr=5e-4,
	apply_unconditional_transform=True,
	base_distribution="standard_normal",
	linear_transform_type="permutation",
	base_transform_type="rq-autoregressive",
	encoder_max_clusters=10,
	n_iter_min=100,
	sampling_patience=500

`TabDDPM`	is_classification=True,
	lr=0.002,
	weight_decay=0.0001,
	batch_size=1024,
	gaussian_loss_type=’mse’,
	scheduler=’cosine’,
	model_type=’mlp’,
	dim_embed=128,
	continuous_encoder=’quantile’,
	sampling_patience=500