TabularARGN: An Auto-Regressive Generative Network for Tabular Data Generation

Andrey Sidorenko* **Ivona Krchova** **Mariana Vargas Vieyra** **Paul Tiwald**
Mario Scriminaci **Michael Platzer**

MOSTLY AI

Abstract

Synthetic data generation for tabular datasets requires a balance of fidelity, efficiency, and adaptability to address real-world applications. We present a synthetic data framework based on the Tabular Auto-Regressive Generative Network (TabularARGN), a flexible architectural framework for modeling tabular data. TabularARGN learns the joint distribution of tabular features by training on encoded representations of the original data and using randomly sampled variable orderings to fit an auto-regressive model. This design naturally supports conditional sampling across arbitrary feature subsets, enabling use cases such as class rebalancing, missing value imputation, and controllable sampling strategies. Although the architectural flexibility of the framework allows for these extensions, our focus in this work is on evaluating the fidelity and efficiency of the generated data. We demonstrate state-of-the-art utility performance with low computational overhead across established benchmarks, making TabularARGN a practical solution for scalable tabular data generation. The framework code is available at https://github.com/mostly-ai/mostlyai-engine.

1 Introduction

Deep neural networks enable the modeling of complex distributions of original data by capturing its underlying patterns, dependencies, and relationships. This capability forms the foundation of synthetic data generation, where models are trained to produce samples that are structurally consistent, statistically representative, and novel. Synthetic data has emerged as a powerful tool for privacy-preserving data sharing and analysis. Beyond privacy, it has potential for scaling data volume, addressing class imbalance, imputing missing values, and supporting fairness-aware evaluations [van der Schaar et al., 2021]. Although not all of these use cases are explored in this work, they highlight the versatility that synthetic data generation frameworks must support [van Breugel and van der Schaar, 2023].

Despite significant progress in tabular data synthesis, most existing approaches are narrowly tailored to specific data regimes. Many methods focus on flat, i.i.d. tabular datasets using GANs [Park et al., 2018, Xu et al., 2019, Zhao et al., 2021, Qian et al., 2023, Li et al., 2023, Zhao et al., 2024] or VAEs [Akrami et al., 2022, Liu et al., 2023], while others target time-series synthesis with sequence models [Yoon et al., 2019, Lin et al., 2020, Desai et al., 2021, Zhicheng et al., 2024, Yuan and Qiao, 2024, Suh et al., 2024]. Approaches for multi-table synthesis remain comparatively scarce and are often designed for predictive tasks [Fey et al., 2023]. As a result, few solutions can accommodate the diversity of real-world tabular data within a unified framework.

In this work, we introduce a simple and flexible generative modeling approach capable of handling a wide variety of tabular data structures and types. Our framework supports flat and sequential

*Corresponding author: andrey.sidorenko@mostly.ai

AI for Tabular Data workshop at EurIPS 2025.

tables, accommodates mixed-type features, including categorical, numerical, datetime, geolocation, and text fields and can model multi-table scenarios with arbitrary relationships. To ensure privacy, training incorporates regularization, early stopping, value range constraints, and can optionally apply differentially private stochastic gradient descent (DP-SGD) for formal privacy guarantees [Dwork et al., 2014, Abadi et al., 2016]. Rather than relying on architectural complexity, the framework leverages a modular design that enables broad adaptability. In this paper, we focus our evaluation on two settings: single-table generation for flat data and a multi-table scenario involving two linked tables, one of which contains sequential records. We assess the performance of our model by comparing it to established baselines using low-order marginal distributions as a proxy for data fidelity, along with runtime efficiency. Additionally, we report results under differential privacy constraints to highlight our framework's practical trade-offs between privacy and utility. We also provide a tested and maintained reference implementation of TabularARGN under a fully permissive Apache v2 Open Source license[2].

2 TabularARGN Framework

In this section, we introduce TabularARGN, an auto-regressive generative model framework designed for flexible and scalable tabular data synthesis. The framework supports any-order (column-order agnostic) training and sampling along the column dimension, enabling it to adapt to a variety of conditional generation tasks, including data imputation and generating datasets where some variables are already provided.

2.1 Model Overview

In the simplest scenario, we are given a d-dimensional tabular dataset of n observations, where rows represent independent and identically distributed (i.i.d.) records, and each variable corresponds to a column in the dataset. Prior to training, we pre-process the original records to derive a privacy-aware representation of the data domain. For categorical columns, infrequent values are grouped into a common category. Numerical features are either binned into percentiles or decomposed into individual digits, while datetime values are broken down into components such as year, month, and day. To reduce sensitivity to outliers, numerical and datetime ranges are clipped. Geospatial data, represented by latitude and longitude, is encoded as sequences of quad-tiles. As a result, each original column is transformed into one or more categorical sub-columns, which serve as inputs to the model.

Let D denote the total number of sub-columns in the pre-processed dataset. During training, the model employs an auto-regressive mechanism that factorizes the joint distribution $p(\mathbf{x})$ of a record $\mathbf{x} \in \mathbb{R}^D$ using the chain rule:

$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i \mid x_1, \ldots, x_{i-1}).$$

This reduces joint density estimation to a series of simpler conditional predictions. To support flexible conditioning, we adopt an any-order training scheme that randomly permutes column orderings for each training batch. This strategy enables the model to learn conditional distributions over arbitrary feature subsets, that is a key requirement for tasks such as imputation, conditional sampling, and fairness-aware generation.

Architecture. The model architecture consists of three main components: an embedding layer, a regressor block, and a predictor layer. Each sub-column is assigned an embedding whose dimension is heuristically determined based on its cardinality. These embeddings are concatenated into a single feature vector. A permutation masking layer then zeroes out entries not permitted by the current auto-regressive context, based on a randomly sampled feature ordering. Each regressor is implemented as a feed-forward neural network with ReLU activations and dropout regularization. The predictor layer applies a softmax transformation to produce a probability distribution over the valid categories, with output dimensionality matching the feature's cardinality. An illustrative example of the architecture is provided in Appendix A.

[2]https://github.com/mostly-ai/mostlyai-engine

Training. TabularARGN is trained to minimize the sum of categorical cross-entropies across all sub-columns. With randomly sampled feature orderings, this corresponds to minimizing the expected negative log-likelihood:

$$\max_{\theta} \mathbb{E}_{\sigma \in \text{Uniform}(S_D)} \left[\sum_{i=1}^{D} \log p_\theta\big(x_{\sigma(i)} \mid x_{\sigma(<i)}\big) \right],$$

where σ is a random permutation over the D features. Teacher forcing is used during training, with ground-truth values from preceding features provided as inputs. This setup naturally casts training as a multi-task prediction problem, one per sub-column. To ensure efficient convergence, we monitor validation loss at each epoch on a held-out split. We apply a learning rate scheduler with patience when progress stalls, and training stops when further reductions yield no improvement. The model parameters with the lowest validation loss are retained as the final checkpoint.

2.2 Extension to Sequential Tables

For sequential tables, TabularARGN extends its auto-regressive modeling across the time dimension. It estimates conditional probabilities of the form:

$$\hat{p}_{x_j, t=T}(x_j \mid x_{i<j, t=T}, \mathcal{H}_{0:T-1}),$$

where $\mathcal{H}_{0:T-1}$ denotes the encoded history of all previous time steps. It uses an LSTM-based encoder to capture the history of previous time steps, applying a causal temporal shift to ensure proper auto-regressive conditioning. The encoded history is combined with masked current-step sub-column embeddings to condition predictions at each time step.

Training proceeds similarly to the flat table scenario, but the loss is aggregated across both columns and time steps, treating each feature at each time step, $x_i^{(t)}$, as a separate predictive task. This enables TabularARGN to model complex temporal dependencies while maintaining flexibility.

The model supports sequences of arbitrary and irregular lengths without requiring equidistant time steps or specific timestamp formats and allowing it to model both time series and unordered event sequences. A diagram and further architectural details are provided in Appendix B.

2.3 Conditional Sequence Generation

TabularARGN supports multi-table setups, where specific tables are generated conditionally based on related tables previously defined as parent tables or *context*. In this work, we focus on a simple two-table scenario involving a sequential table linked to a flat table. In this setting, the model generates each sequential record conditioned not only on column values and time-series history, but also on the associated flat-table row, which provides time-independent context (e.g., a customer profile linked to their transaction history).

A context processor module encodes the flat table into a context embedding $\mathbf{c}_{\text{context}}$, which is concatenated with the column and history embeddings of the sequential table. The full input to the regressor at time step $t = T$ for feature j is then $\left[\mathbf{e}_{x_{i<j}; t=T}, \mathcal{H}_{0:T-1}, \mathbf{c}_{\text{context}}\right]$, where $i, j = 1, \ldots, D$ and $\mathbf{e}_{x_i}$ is the embedding corresponding to x_i. This pipeline is described in more detail in Appendix B, and depicted in Figure 3.

In a two-table setup, the flat and sequential tables can be trained independently. However, during generation, the flat table must be sampled first to provide context for the sequential model. In this sense, the auto-regressive approach is extended along the table dimension, complementing its application across columns and time.

3 Empirical Results

We evaluate the performance of TabularARGN through a comparative study against state-of-the-art (SOTA) baselines. This section focuses on two scenarios: a single flat-table setting and a two-table setting involving linked sequential data. For each case, we select a representative dataset and report results in terms of data quality and computational efficiency.

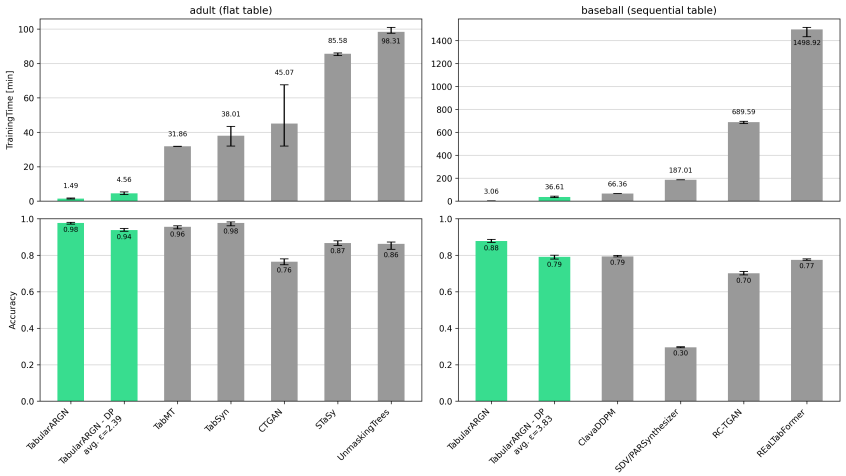


Figure 1: Training time (top) and accuracy (bottom) for the flat *Adult* (left) and sequential *Baseball* (right) datasets. Values are averaged over three or more runs; error bars indicate min/max. TabularARGN is shown both with and without DP training.

Data quality is assessed using low-order statistics that capture univariate and bivariate marginal accuracy, along with a coherence metric for sequential data. Our evaluation metrics follow the approach proposed in Sidorenko et al. [2025]. A detailed description of these metrics is provided in Appendix D. In addition to quality metrics, we report runtime for all models. Core results for the flat and sequential datasets appear in Figure 1, with extended benchmarks presented in Appendix E.

We also include results for TabularARGN trained with DP-SGD. Although a comprehensive study of privacy guaranties is beyond the scope of this work, these results illustrate that good performance can be retained even under strong privacy constraints, as discussed in Sidorenko and Tiwald [2025]. Further experimental details, including information about datasets and benchmarks for both scenarios, are provided in Appendix C.

Flat Tables We conducted experiments on the *Adult* dataset [Dua and Graff, 2019], a census dataset comprising 48k rows with eight categorical and six numerical variables. As baseline models, we include CTGAN [Xu et al., 2019], STaSy [Kim et al., 2023], TabSyn [Zhang et al., 2024], TabMT [Gulati and Roysdon, 2023], and Unmasking Trees [McCarter, 2024].

Results are shown in Figure 1, left. Notably, our model is on-a-par with TabSyn and TabMT, which achieve top accuracy, while outperforming all baselines in terms of runtime. We note that when employing DP-SGD($\varepsilon \sim 2.5$), accuracy drops slightly and runtime increases modestly, yet remains far below that of other methods.

Sequential Tables For the sequential table scenario we use the *Baseball* dataset [Lahman, 2023], which comprises a flat table that serves as context for a second sequential table. We compare TabularARGN to REalTabFormer [Solatorio and Dupriez, 2023], RC-TGAN [Gueye et al., 2023], and ClavaDDPM [Pang et al., 2024].

Results are displayed in Figure 1, right. We observe that TabularARGN outperforms all baselines by nearly 10 percentage points in terms of accuracy. In terms of runtime we remain the top performing model along with ClavaDDPM.

4 Conclusion

We introduced TabularARGN, an auto-regressive framework for generating synthetic data from flat and sequential tables. Its modular design supports flexible conditional generation, enabling use cases such as rebalancing, imputation, and sequence modeling. Empirical results show that the model delivers competitive data quality with strong runtime performance, making it a practical tool for synthetic data generation in real-world analytical workflows.

References

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.

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.

Petr Berka. Pkdd'99 financial dataset, 1999. URL `https://www.kaggle.com/datasets/marceloventura/the-berka-dataset`. Accessed: 2025-01-31.

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.

Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Dheeru Dua and Casey Graff. UCI machine learning repository: Adult (census income) data set, 2019. URL `http://archive.ics.uci.edu/ml/datasets/Adult`. University of California, Irvine, School of Information and Computer Sciences.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023.

Sarah Flood, Miriam King, Renae Rodgers, Steven Ruggles, and J. Robert Warren. Integrated public use microdata series, current population survey: Version 8.0 [dataset]. Minneapolis, MN: IPUMS, 2020. URL `https://doi.org/10.18128/D030.V8.0`.

Mohamed Gueye, Yazid Attabi, and Maxime Dumas. Row conditional-tgan for generating synthetic relational databases. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

Manbir Gulati and Paul Roysdon. Tabmt: Generating tabular data with masked transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46245–46254. Curran Associates, Inc., 2023.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Jayoung Kim, Chaejeong Lee, and Noseong Park. Stasy: Score-based tabular data synthesis. In *The Elenventh International Conference on Learning Representations*, 2023.

Sean Lahman. Sean lahman baseball database, 2023. URL `http://www.seanlahman.com/`. Accessed: January 12, 2025.

Jin Li, Benjamin J Cairns, Jingsong Li, and Tingting Zhu. Generating synthetic mixed-type longitudinal electronic health records for artificial intelligent applications. *NPJ Digital Medicine*, 6(1):98, 2023.

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*, IMC '20, page 464–483, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381383. doi: 10.1145/3419394.3423643. URL `https://doi.org/10.1145/3419394.3423643`.

Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. Goggle: Generative modelling for tabular data by learning relational structure. In *The Eleventh International Conference on Learning Representations*, 2023.

Calvin McCarter. Unmasking trees for tabular data. *arXiv preprint arXiv:2407.05593*, 2024.

MOSTLY AI. MOSTLY AI - Quality Assurance, 2024. URL https://github.com/mostly-ai/mostlyai-qa.

R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, 33(3):291–297, 1997. doi: 10.1016/S0167-7152(96)00140-X.

Wei Pang, Masoumeh Shafieinejad, Lucy Liu, Stephanie Hazlewood, and Xi He. Clavaddpm: Multi-relational data synthesis with cluster-guided diffusion models, 2024. URL https://arxiv.org/abs/2405.17724.

Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks, 2018.

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, Oct 2016. doi: 10.1109/DSAA.2016.49.

Michael Platzer and Thomas Reutterer. Holdout-based empirical assessment of mixed-type synthetic data. *Frontiers in big Data*, 4:679939, 2021.

Zhaozhi Qian, Bogdan-Constantin Cebere, and Mihaela van der Schaar. Synthcity: facilitating innovative use cases of synthetic data in different data modalities, 2023, 2023. URL https://doi.org/10.48550/arXiv.

Cemal Okan Sakar and Yomi Kastro. Online shoppers purchasing intention dataset, 2018. URL https://doi.org/10.24432/C5F88Q. Accessed: 2025-01-31.

Andrey Sidorenko and Paul Tiwald. Privacy-preserving tabular synthetic data generation using tabularargn, 2025. URL https://arxiv.org/abs/2508.06647.

Andrey Sidorenko, Michael Platzer, Mario Scriminaci, and Paul Tiwald. Benchmarking synthetic tabular data: A multi-dimensional evaluation framework, 2025. URL https://arxiv.org/abs/2504.01908.

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

Namjoon Suh, Yuning Yang, Din-Yin Hsieh, Qitong Luan, Shirong Xu, Shixiang Zhu, and Guang Cheng. Timeautodiff: Combining autoencoder and diffusion model for time series tabular data synthesizing. *arXiv preprint arXiv:2406.16028*, 2024.

Boris van Breugel and Mihaela van der Schaar. Beyond privacy: Navigating the opportunities and challenges of synthetic data, 2023. URL https://arxiv.org/abs/2304.03722.

Mihaela van der Schaar, Boris van Breugel, Trent Kyono, and Jeroen Berrevoets. Decaf: Generating fair synthetic data using causally-aware generative networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 22221–22233. Curran Associates, Inc., 2021.

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

I-Cheng Yeh. Default of credit card clients, 2009. URL https://doi.org/10.24432/C55S3H. Accessed: 2025-01-31.

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

Xinyu Yuan and Yan Qiao. Diffusion-ts: Interpretable diffusion for general time series generation, 2024. URL https://arxiv.org/abs/2403.01742.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *The Twelfth International Conference on Learning Representations*, 2024.

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.

Zilong Zhao, Aditya Kunar, Robert Birke, Hiek Van der Scheer, and Lydia Y Chen. Ctab-gan+: Enhancing tabular data synthesis. *Frontiers in big Data*, 6:1296508, 2024.

Chen Zhicheng, FENG SHIBO, Zhong Zhang, Xi Xiao, Xingyu Gao, and Peilin Zhao. Sdformer: Similarity-driven discrete transformer for time series generation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

A The Flat Table Model

This appendix illustrates the flat table model in TabularARGN using a simplified three-column example and accompanying architecture diagram, shown for a particular training batch where the column order is fixed as x_1, x_2, x_3.

In a three-column example (see fig. 2a), the regressor for column x_1 receives a vector of zeros, $\mathbf{0}_{[\mathbf{e}_{x_1},\mathbf{e}_{x_2},\mathbf{e}_{x_3}]}$, with dimensions matching $[\mathbf{e}_{x_1}, \mathbf{e}_{x_2}, \mathbf{e}_{x3}]$. The regressor for column x_2 receives $\left[\mathbf{e}_{x_1}, \mathbf{0}_{\mathbf{e}_{x_2}}, \mathbf{0}_{\mathbf{e}_{x_3}}\right]$, where $\mathbf{0}_{\mathbf{e}_{x_2}}$ and $\mathbf{0}_{\mathbf{e}_{x_3}}$ represent zero vectors of sizes equal to $\mathbf{e}_{x_2}$ and $\mathbf{e}_{x_3}$, respectively. The regressor for column x_3 receives $\left[\mathbf{e}_{x_1}, \mathbf{e}_{x_2}, \mathbf{0}_{\mathbf{e}_{x_3}}\right]$, allowing it to condition on the embeddings of both preceding columns.

Outputs of the regressor layers are passed through predictor layers—dense layers with softmax activation that output conditional probabilities. Each predictor's output size matches the cardinality of its corresponding feature.

During generation (see fig. 2b), a zero vector initializes the process. Given a selected feature order (defaulting to the original), the model predicts the distribution of the first feature, samples a category, embeds it, and passes it to the next regressor. This process continues sequentially, with each feature conditioned on embeddings of all previously sampled features, generating one row feature by feature.

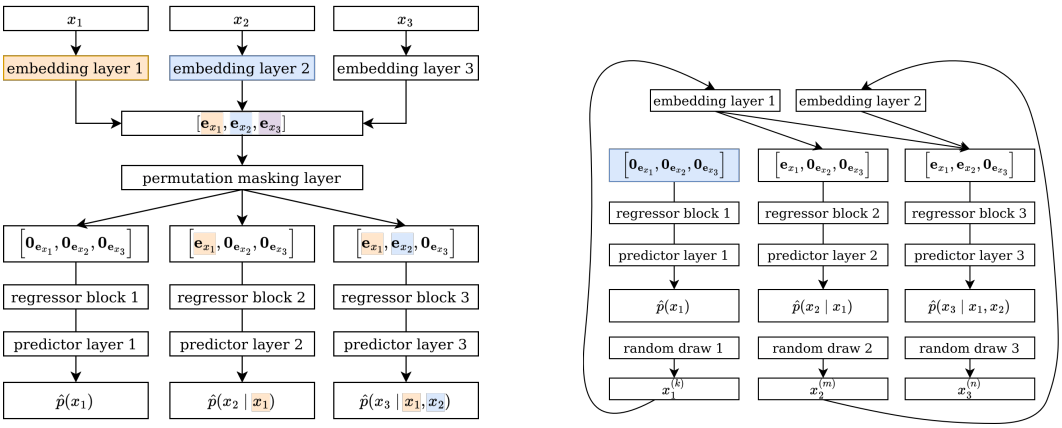
B The Sequential Table Model

For sequential tables, TabularARGN estimates conditional probabilities of the form:

$$\hat{p}_{x_j, t=T}(x_j \mid x_{i<j, t=T}, \mathcal{H}_{0:T-1}),$$

where $\mathcal{H}_{0:T-1}$ denotes the encoded history of all previous time steps (see blue part in fig. 3). This history is produced by an LSTM-based encoder Hochreiter and Schmidhuber [1997], which receives temporally shifted embeddings of all sub-columns from $t = 0$ to $T - 1$. The sequence is padded at the beginning ($t = 0$), and the final time step ($t = T$) is excluded from the encoder input.

The resulting history vector is concatenated with masked feature embeddings at $t = T$ and passed to the column-specific regressors, enabling conditioning on both time and feature dimensions.



(a) (b)

Figure 2: (a) Model components and information flow in the **training phase** of a three-column TabularARGN flat model with the current column order [1,2,3]. Input features x_i are embedded and sent through the permutation masking layer to condition predictions on preceding columns. The permutation masking layer randomly shuffles the column order for each training batch. (b) Model components and information flow in the **generation phase**. The input to the model and starting point of the generation is a vector of zeros (blue) triggering the successive generation of synthetic features. Due to the permutation of column orders during training, any column order can be realized in the generation phase.

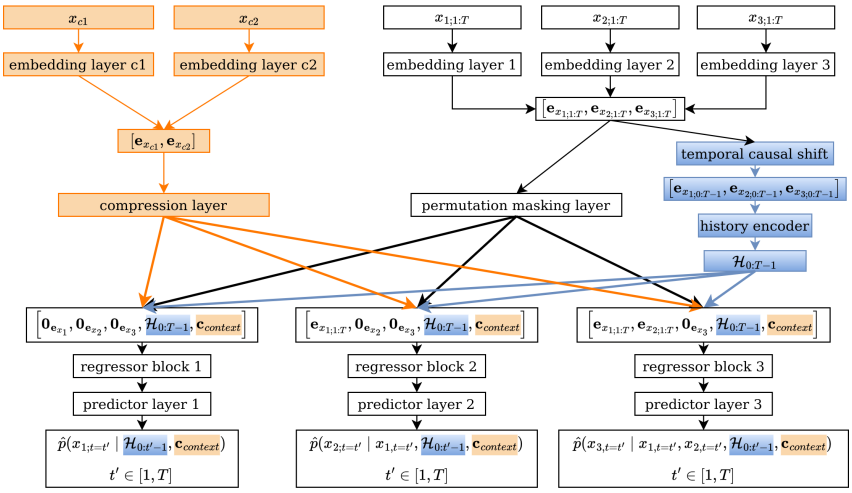


Figure 3: Model components and information flow in the training phase of a three-column sequential table with a two-column flat context. Feature embeddings pass through a permutation mask and LSTM-based history encoder (blue) for column- and time-wise conditioning. Flat context features (orange) are processed and used to condition all sequence steps.

For the conditional sequence generation, the context processor module (see orange part in fig. 3) encodes the flat table into a compressed context embedding. This context is combined with the temporal and feature-level embeddings before being passed to the regressor layers, allowing the model to capture correlations between the flat and sequential tables.

C Experimental setup

Compute: We perform almost all experiments on `AWS g5.2xlarge` instances with 8 vCPUs, 32 GiB system RAM, and one NVIDIA A10G (24 GiB of GPU memory). As a non-deep learning method, UnmaskingTrees is specifically designed to run on CPU and we opted for an `AWS p3.2xlarge` instance with 8 vCPUs and 61 GiB of system RAM.

Data sets: The *Adult* data set [Dua and Graff, 2019] has around 48k rows with eight categorical and six numerical features. *ACS-Income* Ding et al. [2021], Flood et al. [2020] has about 1.5 million rows, containing 28 categorical and four numerical features—making it roughly 30 times larger and twice as wide as *Adult*, and thus a challenging real-world benchmark. *Default* [Yeh, 2009] contains 30,000 financial records of credit card users labeled by default status, and *Shoppers* [Sakar and Kastro, 2018] consists of 12,330 user sessions, capturing browsing behavior and transaction outcomes on an e-commerce platform. We initially considered an even larger data set with more columns, but all baseline methods were either too slow or required excessive GPU memory.

For sequential modeling, we present results on a two-table subset (*players* and *fielding*) of the *Baseball* dataset [Lahman, 2023], the *California* dataset [Pace and Barry, 1997] consisting of a table of *households* and a table of *individuals*, and *Berka* dataset [Berka, 1999], where we only consider the *accounts* and *transactions* tables. For *Baseball*, the *players* table (flat context) has about 20 thousand rows, and the sequential *fielding* table has 140 thousand, resulting in an average sequence length of seven. With over 600 thousand rows, the *household* table has about 30 times more records than the *players* table, resulting in almost 1.7 million rows and an average sequence length of 2.8 in the *individuals* table. The *accounts* table has only 4,500 records, but the *transactions* table is notably larger, encompassing around 1 million rows. All sequence tables have a comparable number of features: 9 (*Berka*), 11 (*Baseball*) and 15 (*California*)

Baselines: The baseline methods included in this study can be grouped depending on whether they process flat tables or sequential tables.

For flat tables, we include CT-GAN, a generative model introduced by Xu et al. [2019] that applies mode-specific normalization to handle multi-modal data while generating numerical features conditioned on categorical ones. STaSy [Kim et al., 2023] builds on score-based generative modeling,

incorporating self-paced learning and fine-tuning strategies to improve the stability of denoising score matching. TabSyn [Zhang et al., 2024] adopts a VAE to project mixed data types into a common latent space, allowing a diffusion model to better capture the underlying data distribution. Unmasking Trees, proposed by McCarter [2024], is an alternative route to deep learning models. This framework leverages gradient-boosted decision trees to incrementally unmask individual features.

For sequential tables, we consider REalTabFormer [Solatorio and Dupriez, 2023], a transformer-based framework that generates flat records auto-regressively using a GPT-2 model while employing a sequence-to-sequence approach to synthesize child tables conditioned on parent data. SDV [Patki et al., 2016] takes a different approach by iteratively modeling relationships across a relational database, enabling flexible data synthesis. ClavaDDPM [Pang et al., 2024] introduces a cluster-guided diffusion mechanism that captures inter-table dependencies via latent representations, improving the modeling of complex multi-relational structures. Finally, RC-TGAN [Gueye et al., 2023] conditions the generation of child table rows on parent data, extending its modeling capacity to high-order relationships across multiple interconnected tables.

Code repositories: Table 1 lists the repositories containing implementations of all methods used for the benchmarks in this paper. Throughout this benchmark, we use the default settings of *TabularARGN* as implemented in x. *TabSyn* and *STaSy* are implemented in `https://github.com/amazon-science/tabsyn`. For implementation details of these baselines, we refer to appendix G.2 of Zhang et al. [2024].

Method	Repository
TabSyn	`https://github.com/amazon-science/tabsyn`
STaSY	`https://github.com/amazon-science/tabsyn`
CT-GAN	`https://github.com/vanderschaarlab/synthcity`
UnmaskingTrees	`https://github.com/calvinmccarter/unmasking-trees`
SDV	`https://github.com/sdv-dev/SDV`
RC-TGAN	`https://github.com/croesuslab/RCTGAN`
REaLTabFormer	`https://github.com/worldbank/REaLTabFormer`
ClavaDDPM	`https://github.com/weipang142857/ClavaDDPM`

Table 1: List of benchmark methods and their repositories.

D Metrics

The metrics used for the evaluation of synthetic-data quality follow the general approach of Platzer and Reutterer [2021] and are available in a well-maintained and documented open-source GitHub repository MOSTLY AI [2024]. We include metrics for measuring low-dimensional marginal statistics and a distance-based metric to measure the novelty, i.e. privacy of the synthetic data.

D.1 Low-Order Marginal Statistics

Low-order marginal statistics are evaluated by comparing univariate (column-wise) distributions and the pairwise correlations between columns. To handle mixed-type data, numerical and datetime columns are discretized by grouping their values into deciles defined by the original training data, resulting in 10 groups per column (equally sized for the original data). For categorical columns, only the 10 most frequent categories are retained, with the remaining categories disregarded. This approach ensures comparability across data types while focusing on the most significant features of the data.

For each feature, we derive a vector of length 10 from the training (original) data and another from the synthetic data. For numerical and DateTime columns, the vectors represent the frequencies of the groups defined by the original deciles. For categorical columns, the vectors reflect the frequency distribution of the top 10 categories after re-normalization. These feature-specific vectors are denoted as $\mathbf{X}_{\text{trn}}^{(m)}$ and $\mathbf{X}_{\text{syn}}^{(m)}$, corresponding to the training and synthetic data, respectively. m is the feature index, running from 1 to d.

The **univariate accuracy** of column m is then defined as

$$acc_{\text{univariate}}^{(m)} = \frac{1}{2}\Big(1 - \|\mathbf{X}_{\text{trn}}^{(m)} - \mathbf{X}_{\text{syn}}^{(m)}\|_1\Big) \tag{1}$$

and the overall univariate accuracy, as reported in the results section, is defined by

$$acc_{\text{univariate}} = \frac{1}{D}\sum_{m}^{D} acc_{\text{univariate}^{(m)}} \ , \tag{2}$$

where D is the number of columns.

For bi-variate metrics, we evaluate the relationships between pairs of columns by constructing normalized contingency tables. These tables capture the joint distribution of two features, m and n, allowing us to assess pairwise dependencies.

The contingency table between columns m and n is denoted as $\mathbf{C}_{\text{trn}}^{(m,n)}$ for the training data and $\mathbf{C}_{\text{syn}}^{(m,n)}$ for the synthetic data. Each table has a maximum dimension of 10×10, corresponding to the (discretized) values or the top 10 categories of the two features. For columns with fewer than 10 categories (categorical columns with cardinality <10), the dimensions of the table are reduced accordingly.

Each cell in the table represents the normalized frequency with which a specific combination of categories or discretized values from columns m and n appears in the data. This normalization ensures that the contingency table is comparable across features and datasets, regardless of their absolute scale or size.

The **bivariate accuracy** of the column pair m, n is defined as

$$acc_{\text{bivariate}}^{(m,n)} = \frac{1}{2}\Big(1 - \|\mathbf{C}_{\text{trn}}^{(m,n)} - \mathbf{C}_{\text{syn}}^{(m,n)}\|_{1,\text{entrywise}}\Big) = \frac{1}{2}\left(1 - \sum_i \sum_j = \left|\mathbf{C}_{\text{trn}}^{(m,n)} - \mathbf{C}_{\text{syn}}^{(m,n)}\right|_{i,j}\right) \tag{3}$$

and the overall bivariate accuracy, as reported in the results section, is given by

$$acc_{\text{bivariate}} \frac{2}{D(D-1)} \sum_{1 \le m < n \le D} acc_{\text{bivariate}}^{(m,n)} \ , \tag{4}$$

the average of the strictly upper triangle of $acc_{\text{bivariate}}^{(m,n)}$.

Note that due to sampling noise, both $acc_{\text{univariate}}$ and $acc_{\text{bivariate}}$ cannot reach 1 in practice. The software package reports the theoretical maximum alongside both metrics.

There is no difference in calculating the univariate and bivariate accuracies between flat and sequential data. In both cases, vectors $\mathbf{X}^{(m)}$ and contingency tables $\mathbf{C}^{(m,n)}$ are based on all entries in the columns, irrespective of which data subject they belong to.

The coherence metric, specific to sequential data, evaluates the consistency of relationships between successive time steps or sequence elements. It is conceptually similar to the bi-variate accuracy metric but adapted for sequential datasets. The process is as follows:

- For each data subject, we randomly sample two successive sequence elements (time steps) from their sequential data.
- These pairs of successive time steps are transformed into a wide-format dataset. To illustrate, consider a sequential dataset of N subjects and original columns A, B, C, represented as $K > N$ rows. After processing, the resulting dataset has six columns: A, A', B, B', C, C'. The unprimed columns correspond to the first sampled sequence element, the primed columns correspond to the successive sequence element. The number of rows in this wide-format dataset is equal to N, irrespective of the sequence lengths in the original dataset.

Using this wide-format dataset, we construct contingency tables $\mathbf{C}^{(m,m')}$ for each pair of corresponding unprimed and primed columns (m, m'). These tables are normalized and used to calculate the **coherence metric** for column m as:

$$acc_{\text{coherence}}^{(m,m')} = \frac{1}{2}\Big(1 - \|\mathbf{C}_{\text{trn}}^{(m,m')} - \mathbf{C}_{\text{syn}}^{(m,m')}\|_{1,\text{entrywise}}\Big) \tag{5}$$

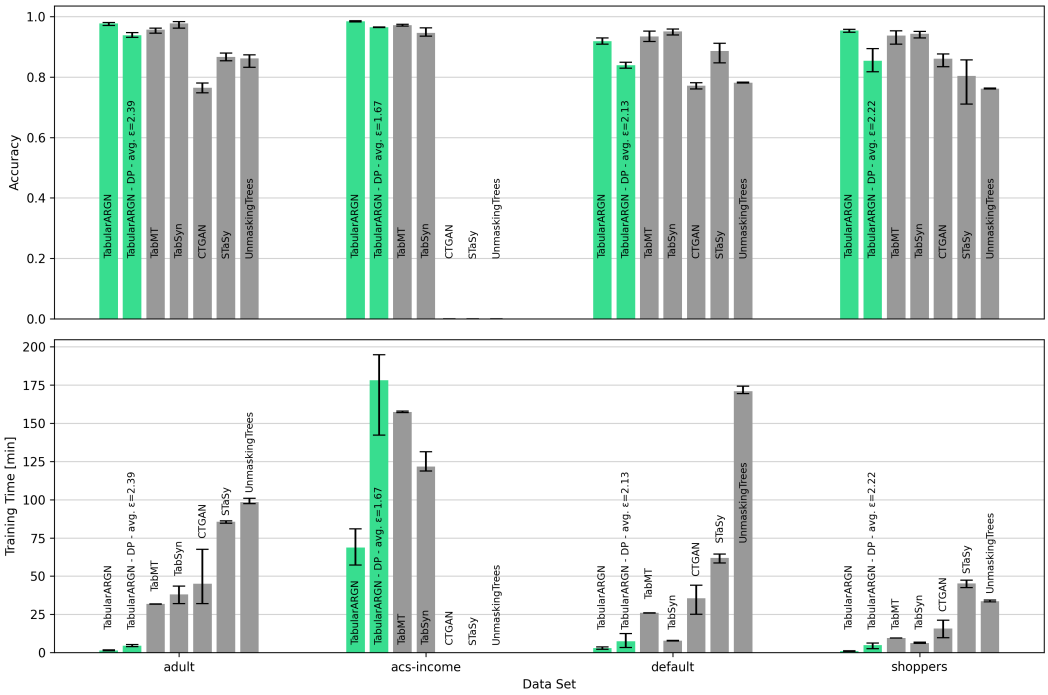


Figure 4: Accuracy (top row) and Training Time (bottom row) for the flat datasets *Adult*, *ACS-Income*, *Default*, and *Shoppers* and various generative models - TabularARGN, TabularARGN with DP, TabMT, TabSyn, CTGAN, STaSy, and UnmaskingTree. Reported values are averages over five full training and generation runs, error bars indicate the minimum and maximum values. As UnmaskingTrees is a non-Deep Learning approach, it is run on CPU. For comparison, Training of TabularARGN on the *Adult* with a CPU takes approximately 2.5 minutes.

and the overall coherence metric, as reported in the results section

$$acc_{\text{coherence}} = \frac{1}{D} \sum_{m}^{D} acc_{\text{coherence}}^{(m,m')} \,. \tag{6}$$

We summarize the **overall accuracy** of a data set as

$$\frac{1}{2}(acc_{\text{univariate}} + acc_{\text{bivariate}}) \tag{7}$$

and

$$\frac{1}{3}(acc_{\text{univariate}} + acc_{\text{bivariate}} + acc_{\text{coherence}}) \tag{8}$$

for flat and sequential data, respectively.

E Further Results

This section provides an overview of results not included in the main text. Specifically, we report results on additional data sets (see description in C for both flat and sequential tables.)

Similar to the *Adult* data set, TabularARGN performs on par concerning accuracy on the *ACS-Income*, *Default*, and *Shoppers* data set with SOTA models (see Fig. 4). For some data sets, it marginally surpasses them. Throughout all data sets, the training time of TabularARGN is the smallest, often by a large margin.

For sequential data (see Fig. 5), TabularARGN achieves the highest accuracy values across all methods. In terms of runtime, it outperforms almost all of them with the only exception of ClavaDDPM for

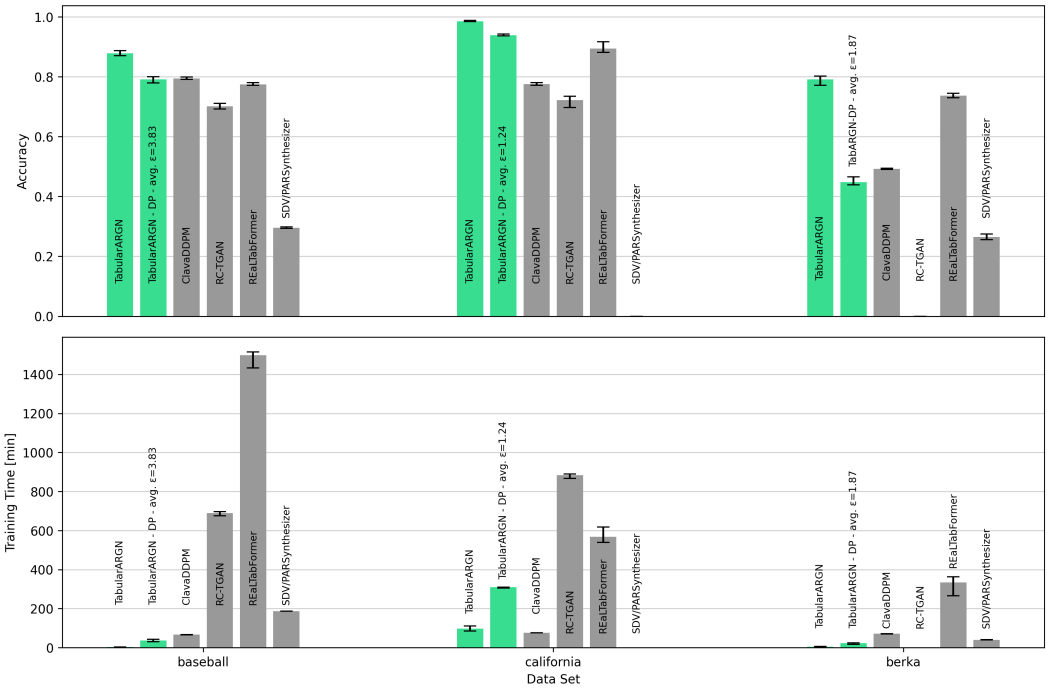


Figure 5: Accuracy (top row) and Training Time (bottom row) for the sequential datasets *Baseball*, *California*, and *Berka* and various generative models - TabularARGN, TabularARGN with DP, ClavaDDPM, RCTGAN, RealTabFormer, and SDV/ParSynthesizer. Reported values are averages over at least three full training and generation runs, error bars indicate the minimum and maximum values. RCTGAN and SDV/ParSynthesizer fail to train on the Berka and California datasets, respectively, due to out-of-memory (OOM) issues.

the *California* data set. However, ClavaDDPM lags in accuracy underpinning the strong accuracy-efficiency trade-off of TabularARGN. RC-TGAN fails on the *Berka* and SDV on the *California* data set with an OOM error.