Causal Data Augmentation for Robust Fine-Tuning of Tabular Foundation Models

Magnus Bühler
Department of Computer Science
University of Freiburg
Freiburg im Breisgau, Germany
buehlema@informatik.uni-freiburg.de

Lennart Purucker
Department of Computer Science
University of Freiburg
Freiburg im Breisgau, Germany

Frank Hutter
Prior Labs
ELLIS Institute Tübingen
University of Freiburg

Abstract

Fine-tuning tabular foundation models (TFMs) in the face of scarce data is challenging, as early stopping on even scarcer validation data often fails to capture true generalization performance. We propose CausalMixFT, a method that enhances fine-tuning robustness and downstream performance by generating structurally consistent synthetic samples using Structural Causal Models (SCMs) fitted on the target dataset. This approach augments limited real data with causally informed synthetic examples, preserving feature dependencies while expanding training diversity. Evaluated across 33 classification datasets from TabArena and over 2,300 fine-tuning runs, our CausalMixFT method consistently improves the improvement of median normalized ROC-AUC by fine-tuning from 0.10 (standard fine-tuning) to 0.12, outperforming purely statistical generators such as CTGAN (-0.01), TabEBM (-0.04), and TableAugment (-0.09). Moreover, it narrows the median validation-test performance correlation gap from 0.67 to 0.30, enabling more reliable validation-based early stopping—a key step toward improving fine-tuning stability under data scarcity. These results demonstrate that incorporating causal structure into data augmentation provides an effective and principled route to fine-tuning tabular foundation models in low-data regimes.

1 Introduction

Foundation models have transformed machine learning across vision [1], medicine [17, 2], time series [20], and graphs [38]. Yet the most ubiquitous data type in the real world, namely **tabular data**, has long remained the hardest to model effectively. Recent advances in pre-trained tabular foundation models (TFMs) such as TabPFN [13, 14, 10], TabICL [30], and TabDPT [24] signal a paradigm shift: transformers trained across millions of datasets can now perform in-context learning on unseen tables, rivaling classical methods like XGBoost [6].

While these models demonstrate strong zero-shot generalization, their full potential emerges only after fine-tuning on specific target datasets. Recent models such as Mitra[1], TabPFNv2 [14], and LimiX [41] now offer out-of-the-box fine-tuning capabilities in response to the growing demand for data-efficient model adaptation. However, existing fine-tuning practices implicitly assume abundant labeled data,

[1] https://huggingface.co/autogluon/mitra-classifier

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Tabular Data.

an assumption that is not always met in practice. In fact, more than 39% of OpenML datasets contain fewer than 1,000 samples, representing the largest share of available dataset sizes [26]. Under such constraints, conventional data splits and validation-based early stopping become unreliable, leading to overfitting to the training or validation split and unstable performance estimates. Consequently, fine-tuning towards small and heterogeneous datasets remains an unsolved challenge for tabular foundation models.

To overcome these limitations, we propose **CausalMixFT**, a strategy that augments scarce training samples through *learnable Structural Causal Models (SCMs)*. Unlike statistical generators, SCMs recover and exploit causal dependencies among features, producing *structurally consistent* synthetic samples that preserve the semantics of the target domain. By enriching fine-tuning data with causally coherent examples, TFMs can be fine-tuned effectively without overfitting to limited real observations.

Contributions. Our work makes three key contributions:

1. **Empirical diagnosis:** We provide the first systematic analysis of fine-tuning tabular foundation models under severe data scarcity, revealing that large validation-test discrepancies persist even under strong regularization.
2. **Methodological innovation:** We introduce **CausalMixFT**, an SCM-based approach that learns the underlying causal structure of small datasets to generate *structurally consistent* synthetic samples, enabling data-efficient fine-tuning.
3. **Comprehensive evaluation:** Across diverse benchmarks and over 2,300 fine-tuning runs, our method consistently surpasses conventional fine-tuning and statistical augmentation baselines, establishing a new SOTA for fine-tuning tabular foundation models in low-data regimes.

By combining causal generative modeling with foundation model adaptation, our work provides a principled and data-efficient pathway toward making tabular foundation models fine-tunable in the small-data settings that dominate real-world machine learning.

2 Related Work

PFNs and Variants. Müller et al. [28] introduced *Prior-Data Fitted Networks* (PFNs), using transformers to approximate Bayesian posterior predictive distributions via in-context learning. Subsequent works have extended this paradigm to classification and regression tasks [13, 4, 24, 30, 14, 23, 41, 10], scaling to larger datasets and diverse pre-training regimes on both real [24] and synthetic data [4]. These advances establish PFNs as universal tabular priors that can generalize across domains, forming the basis for most current TFMs.

Fine-Tuning TabPFN & Regularization. Recent work investigates the adaptation of PFNs for large datasets. Approaches include full-weight fine-tuning with prior regularization [4], continued pre-training [24, 10], tokenization-layer adaptation [37], encoder compression and distillation [9, 25], mixture-of-experts routing [33, 39], and batch-ensemble encoders [23]. Additional studies refine context retrieval and conditioning [36, 24, 18], and Rubachev et al. [31] provide a general survey. Classical regularization techniques such as L2-SP [21, 10], stochastic weight averaging [15], and early stopping [29] mitigate overfitting in low-data regimes. However, fine-tuning under *data scarcity*—the regime most common in practice—remains largely unaddressed. While Kadra et al. [16] has shown that strong regularization can improve performance in the tabular domain, our work provides a new regularization through data augmentation for TFMs.

Synthetic Tabular Data Generation. Synthetic data generation supports privacy, data sharing, and augmentation. GAN-based models [40] and diffusion approaches [22] improve fidelity, while privacy-preserving GANs [42] and energy-based models [27] provide interpretability and control. Yet most focus on distributional realism rather than improving downstream model adaptation. In contrast, we evaluate these approaches for their ability to enhance fine-tuning.

Research Gap. Despite progress in tabular foundation models (TFMs) and synthetic data generation, their integration for small-data fine-tuning and regularization remains largely unexplored. Building on Garg et al. [10], we investigate whether combining real and structured synthetic samples can enhance downstream-performance and robustness. Our work addresses this gap by coupling causal data generation with fine-tuning strategies for TFMs.

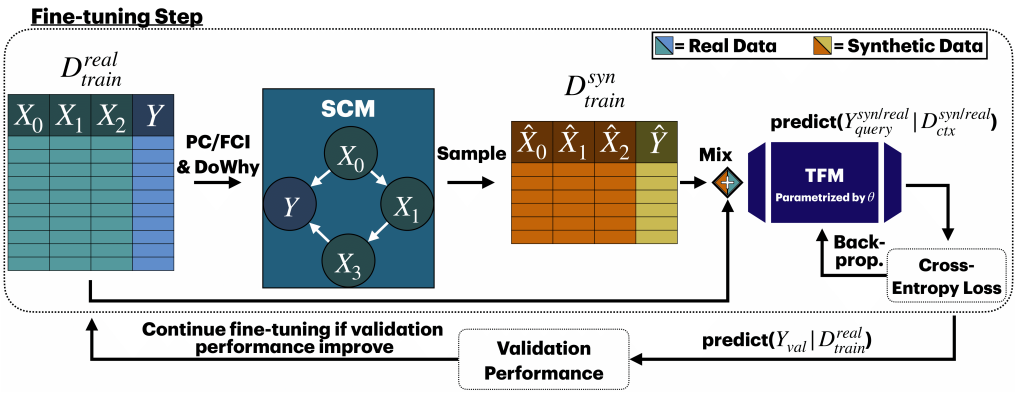


Figure 1: **Overview of the SCM-augmented fine-tuning process.** Real training data ($D^{\text{real}}_{\text{train}}$) are used to fit a Structural Causal Model (SCM) via PC/FCI and DoWhy [32]. The SCM samples synthetic data ($D^{\text{syn}}_{\text{train}}$) that preserve the discovered causal dependencies among features. Real and synthetic samples are mixed in equal proportion to fine-tune the tabular foundation model (TFM), which is optimized by cross-entropy loss. Validation is performed only on real data, and fine-tuning continues as long as validation performance improves.

3 Methodology

Our method extends the fine-tuning framework of Bühler et al. [5] by mixing real and causally grounded synthetic samples into the fine-tuning process, adding more recent baselines, evaluating a TFM optimized for fine-tuning and evaluating across a broad collection of real world datasets. Specifically, we generate synthetic data using SCMs fitted to the target dataset, enabling the model to learn jointly from real samples and causally coherent augmentations. This design preserves feature dependencies while expanding sample diversity, which enhances robustness and generalization under low-data constraints.

SCM-Based Synthetic Augmentation (CausalMixFT). Unlike purely statistical generators, SCMs explicitly encode causal dependencies among features through a directed acyclic graph (DAG) and a set of structural equations, allowing data augmentation to respect the underlying data-generating process. We first estimate the structural relations between the features using the PC and FCI algorithms [34, 35], producing a probabilistic adjacency matrix that encodes edge strengths between variables. DAGs are then sampled and fitted using DoWhy's SCM framework with additive noise models [32]. Numerical features are modeled with regressors, and categorical features with classifiers. The complexity of the internally used model types can be controlled through a quality hyperparameter. Synthetic samples are generated by sampling exogenous noise and propagating it through the fitted SCM, yielding data that captures both causal structure and realistic variability. See Appendix J for more details.

Model & Data Overview. We adopt the *Mitra* foundation model as the tabular backbone, as it is explicitly designed for per-dataset fine-tuning and has achieved state-of-the-art performance on the TabArena benchmark. Further, Mitra is provided with strong default hyperparameters. Our experiments cover 33 classification datasets from the `TabArena` benchmark suite, excluding datasets with more than 200 features (OpenML IDs 46912, 46919, 46939, 46908, 46933) using 10 folds each. Datasets with more than 200 features were excluded to ensure SCM fitting remains within a one-hour runtime limit. Each dataset is split into training, validation, and test subsets using stratified sampling, with training and validation sets capped at 600 and 200 samples respectively to simulate small-data conditions. Further details about the data splitting and light pre-processing are provided in Appendix L and M.

Fine-Tuning and Implementation. Let the downstream dataset be denoted as $D^{\text{real}} = \{(x_i, y_i)\}_{i=1}^{n}$ and the corresponding SCM-generated data as $D^{\text{syn}} = \{(x_j^{\text{syn}}, y_j^{\text{syn}})\}_{j=1}^{m}$. The model is fine-tuned on the combined dataset $D^{\text{mix}} = D^{\text{real}} \cup D^{\text{syn}}$, where both sources are equally represented in each batch. To balance contributions from real and synthetic data, we define a weighted fine-tuning

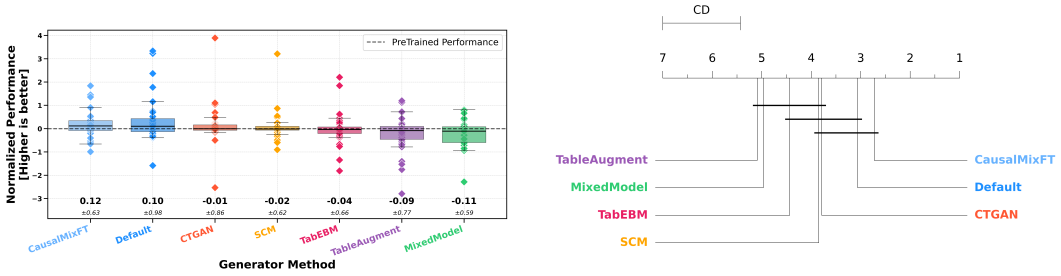


Figure 2: **Performance comparison across data generation strategies.** (left) Normalized ROC-AUC improvements relative to the pre-trained baseline (dashed line). Whiskers represent $1.5\times$ IQR; medians and standard deviations are annotated. **Higher score is better**. (right) Critical difference diagram (significance level = 0.05) **Lower rank is better** [7].

objective:

$$\mathcal{L} = \alpha\, \mathbb{E}_{(x,y)\sim D^{\text{real}}}[\ell(f_{\theta'}(x), y)] + (1-\alpha)\, \mathbb{E}_{(x,y)\sim D^{\text{syn}}}[\ell(f_{\theta'}(x), y)],$$

where $\alpha = 0.5$ unless specified otherwise. Validation is performed exclusively on $D_{\text{val}}^{\text{real}}$ to ensure that improvements reflect genuine generalization rather than memorization of synthetic data. Early stopping is triggered when validation log-loss fails to improve for a fixed number of iterations. Optimization uses the default *Mitra* hyperparameters. A schematic overview of the iterative fine-tuning steps is shown in Figure 1. Additional implementation details and hyperparameter settings are provided in Appendix J.

4 Results

We evaluate whether CausalMixFT improves the robustness and generalization of tabular foundation models under data scarcity. Experiments are conducted on the *Mitra* model across 33 classification datasets with 10 folds each from the `TabArena` benchmark suite, totaling 2,310 fine-tuning runs. Model performance is reported as normalized ROC-AUC relative to the pre-trained model (see Appendix H).

Fine-Tuning Performance. Figure 2 summarizes the normalized test performance across all data generation strategies. On the left plot the proposed CausalMixFT, which combines real and causally generated samples, achieves the highest median improvement of (**+0.12**±0.63) over the pre-trained model, outperforming both the default fine-tuning baseline (+0.10 ±0.98) and all purely synthetic augmentation methods, including CTGAN, SCM, TabEBM, TableAugment and MixedModel (which, in fact, show negative median improvements).

While default fine-tuning occasionally achieves higher peak performance on individual datasets, its variability is substantially larger than that of our method **(Default: ±0.98 vs. CausalMixFT: ±0.63)**. This indicates greater instability across datasets, whereas the *CausalMixFT* configuration acts as a consistent regularizer, improving median performance and reducing variance. These results show that SCM-based augmentation stabilizes fine-tuning under small-data conditions by introducing causally structured synthetic diversity.

Figure 2 (right) presents the average ranks and corresponding critical difference (CD) intervals across datasets. **CausalMixFT ranks first overall**, followed by the default fine-tuning baseline, while purely synthetic generators occupy lower ranks, confirming the results of the boxplot. We further analyze the validation-test performance gap in Appendix A, showing that early stopping based on limited validation data leads to significant validation set overfitting depending on the fine-tuning data mix used.

5 Discussion

We empirically find that fine-tuning tabular foundation models (TFMs) in tiny-to-small data regimes remains highly challenging. To address this, we propose **CausalMixFT**, a causally grounded fine-tuning approach that leverages SCM-based augmentation to improve both stability and generalization.

Across a curated benchmark, **CausalMixFT** consistently outperforms standard fine-tuning, achieving a superior balance between robustness and data efficiency. This provides a principled path forward for more reliable adaptation of TFMs under scarce supervision.

References

[1] Muhammad Awais, Muzammal Naseer, Salman Khan, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, and Fahad Shahbaz Khan. Foundation models defining a new era in vision: a survey and outlook. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025.

[2] Bobby Azad, Reza Azad, Sania Eskandari, Afshin Bozorgpour, Amirhossein Kazerouni, Islem Rekik, and Dorit Merhof. Foundational models in medical imaging: A comprehensive survey and future vision. *arXiv preprint arXiv:2310.18689*, 2023.

[3] Louis Bethune, David Grangier, Dan Busbridge, Eleonora Gualdoni, Marco Cuturi, and Pierre Ablin. Scaling laws for forgetting during finetuning with pretraining data injection. *arXiv preprint arXiv:2502.06042*, 2025.

[4] Felix den Breejen, Sangmin Bae, Stephen Cha, and Se-Young Yun. Fine-tuned in-context learning transformers are excellent tabular data classifiers. *arXiv preprint arXiv:2405.13396*, 2024.

[5] Magnus Bühler, Lennart Purucker, and Frank Hutter. Towards synthetic data for fine-tuning tabular foundation models. In *1st ICML Workshop on Foundation Models for Structured Data*, 2025.

[6] 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.

[7] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[8] Felix den Breejen, Sangmin Bae, Stephen Cha, Tae-Young Kim, Seoung Hyun Koh, and Se-Young Yun. Fine-tuning the retrieval mechanism for tabular deep learning. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

[9] Benjamin Feuer, Robin Schirrmeister, Valeriia Cherepanova, Chinmay Hegde, Frank Hutter, Micah Goldblum, Niv Cohen, and Colin White. Tunetables: Context optimization for scalable prior-data fitted networks. *Advances in Neural Information Processing Systems*, 37:83430–83464, 2024.

[10] Anurag Garg, Muhammad Ali, Noah Hollmann, Lennart Purucker, Samuel Müller, and Frank Hutter. Real-tabpfn: Improving tabular foundation models via continued pre-training with real-world data. *arXiv preprint arXiv:2507.03971*, 2025.

[11] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[12] Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. *arXiv preprint arXiv:2410.24210*, 2024.

[13] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

[14] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025. ISSN 1476-4687. doi: 10.1038/s41586-024-08328-6. URL `https://doi.org/10.1038/s41586-024-08328-6`.

[15] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[16] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021.

[17] Wasif Khan, Seowung Leem, Kyle B See, Joshua K Wong, Shaoting Zhang, and Ruogu Fang. A comprehensive survey of foundation models in medicine. *IEEE Reviews in Biomedical Engineering*, 2025.

[18] Mykhailo Koshil, Thomas Nagler, Matthias Feurer, and Katharina Eggensperger. Towards localization via data embedding for tabpfn. In *NeurIPS 2024 Third Table Representation Learning Workshop*, 2024.

[19] Suhas Kotha, Jacob Mitchell Springer, and Aditi Raghunathan. Understanding catastrophic forgetting in language models via implicit inference. *arXiv preprint arXiv:2309.10105*, 2023.

[20] Siva Rama Krishna Kottapalli, Karthik Hubli, Sandeep Chandrashekhara, Garima Jain, Sunayana Hubli, Gayathri Botla, and Ramesh Doddaiah. Foundation models for time series: A survey. *arXiv preprint arXiv:2504.04011*, 2025.

[21] Xuhong LI, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2825–2834. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/li18a.html`.

[22] Xiaofeng Lin, Chenheng Xu, Matthew Yang, and Guang Cheng. Ctsyn: A foundational model for cross tabular data generation. *arXiv preprint arXiv:2406.04619*, 2024.

[23] Si-Yang Liu and Han-Jia Ye. Tabpfn unleashed: A scalable and effective solution to tabular classification problems. *arXiv preprint arXiv:2502.02527*, 2025.

[24] Junwei Ma, Valentin Thomas, Rasa Hosseinzadeh, Hamidreza Kamkari, Alex Labach, Jesse C Cresswell, Keyvan Golestan, Guangwei Yu, Maksims Volkovs, and Anthony L Caterini. Tabdpt: Scaling tabular foundation models. *arXiv preprint arXiv:2410.18164*, 2024.

[25] Junwei Ma, Valentin Thomas, Guangwei Yu, and Anthony Caterini. In-context data distillation with tabpfn. *arXiv preprint arXiv:2402.06971*, 2024.

[26] Ahmed Mamdouh, Moumen El-Melegy, Samia Ali, and Ron Kikinis. Tab2visual: Overcoming limited data in tabular data classification using deep learning with visual representations. *arXiv preprint arXiv:2502.07181*, 2025.

[27] Andrei Margeloiu, Xiangjian Jiang, Nikola Simidjievski, and Mateja Jamnik. Tabebm: A tabular data augmentation method with distinct class-specific energy-based models. *arXiv preprint arXiv:2409.16118*, 2024.

[28] Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

[29] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[30] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. *arXiv preprint arXiv:2502.05564*, 2025.

[31] Ivan Rubachev, Akim Kotelnikov, and Nikolay Kartashev. On finetuning tabular foundation models. *arXiv preprint arXiv:2506.08982*, 2025.

[32] Amit Sharma and Emre Kiciman. Dowhy: An end-to-end library for causal inference. *arXiv preprint arXiv:2011.04216*, 2020.

[33] N Shazeer, A Mirhoseini, K Maziarz, A Davis, Q Le, G Hinton, and J Dean. The sparsely-gated mixture-of-experts layer. *Outrageously large neural networks*, 2017.

[34] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000.

[35] Peter L Spirtes, Christopher Meek, and Thomas S Richardson. Causal inference in the presence of latent variables and selection bias. *arXiv preprint arXiv:1302.4983*, 2013.

[36] Valentin Thomas, Junwei Ma, Rasa Hosseinzadeh, Keyvan Golestan, Guangwei Yu, Maks Volkovs, and Anthony L Caterini. Retrieval & fine-tuning for in-context tabular models. *Advances in Neural Information Processing Systems*, 37:108439–108467, 2024.

[37] Valentin Thomas, Junwei Ma, Rasa Hosseinzadeh, Keyvan Golestan, Guangwei Yu, Maks Volkovs, and Anthony L Caterini. Retrieval & fine-tuning for in-context tabular models. *Advances in Neural Information Processing Systems*, 37:108439–108467, 2024.

[38] Zehong Wang, Zheyuan Liu, Tianyi Ma, Jiazheng Li, Zheyuan Zhang, Xingbo Fu, Yiyang Li, Zhengqing Yuan, Wei Song, Yijun Ma, et al. Graph foundation models: A comprehensive survey. *arXiv preprint arXiv:2505.15116*, 2025.

[39] Derek Xu, Olcay Cirit, Reza Asadi, Yizhou Sun, and Wei Wang. Mixture of in-context prompters for tabular pfns. *arXiv preprint arXiv:2405.16156*, 2024.

[40] Lei Xu, M Skoularidou, A Cuesta-Infante, and K Veeramachaneni. Modeling tabular data using conditional gan. arxiv 2019. *arXiv preprint arXiv:1907.00503*, 1, 2019.

[41] Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, Ningbo Dai, Renzhe Xu, Shuyang Li, Tianyang Zhang, Yue He, Yuanrui Wang, Yunjia Zhang, Zijing Xu, Dongzhe Li, Fang Gao, Hao Zou, Jiandong Liu, Jiashuo Liu, Jiawei Xu, Kaijie Cheng, Kehan Li, Linjun Zhou, Qing Li, Shaohua Fan, Xiaoyu Lin, Xinyan Han, Xuanyue Li, Yan Lu, Yuan Xue, Yuanyuan Jiang, Zimu Wang, Zhenlei Wang, and Peng Cui. Limix: Fine-tuning tabular foundation models via limited mixture adaptation. *arXiv preprint arXiv:2509.03505*, 2025.

[42] 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.

Acknowledgment The authors acknowledge the use of ChatGPT-5 (OpenAI, 2025) for assistance in refining sentence formulations and in structuring tables and figures to enhance the clarity and presentation of this paper.

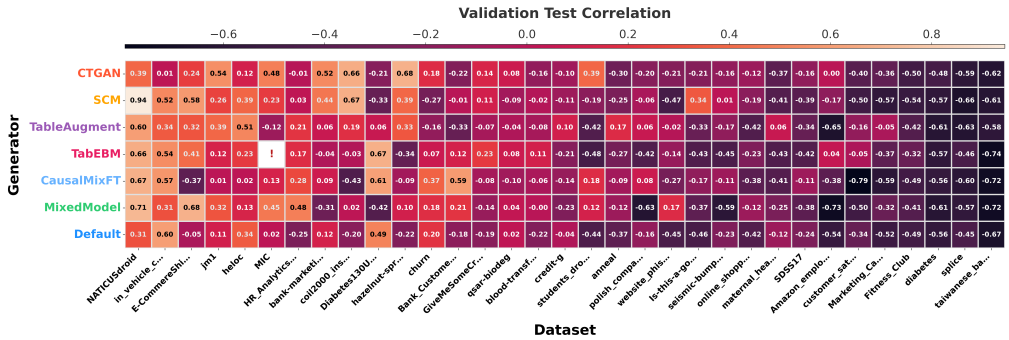


Figure 3: **Validation–test performance correlation across datasets and data generators.** Each cell shows the Pearson correlation between validation and test log-loss for a given dataset and generator configuration. Low or negative correlations indicate that validation performance is not a reliable proxy for generalization under small-data conditions. The columns and rows are sorted by average correlation coefficients from left (higher) to right (lower) and top (higher) to bottom (lower). Incomplete runs due to the time limit of 1h or too many features are marked with "!" (TabEBM uses TabPFNv1 internally, which only allows for 100 features)

Appendix Overview

This appendix provides additional analyses, figures, and implementation details that complement the main text. It includes extended evaluations of validation-test correlations, overfitting behavior, performance heterogeneity across data generators, as well as in-depth analyses of model weight adaptation and normalization procedures.

A Validation–Test Performance Gap

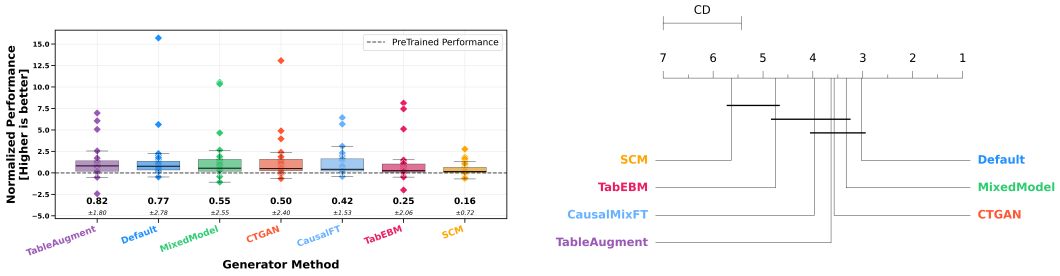
To better understand the relationship between validation and test performance during fine-tuning, we analyze the Pearson correlation between validation log-loss and test log-loss across all generator configurations. While validation metrics often suggest strong improvements, the corresponding test results frequently show diminished gains. This discrepancy indicates that validation performance provides a weak and noisy signal for true generalization, particularly in the low-data regime where validation splits are small.

Figure 3 presents the correlation heatmap between validation and test performance across datasets and generator types. Correlations vary substantially, with several negative or near-zero values, highlighting the instability of validation-based early stopping under data-scarce conditions. Among all methods, the *CausalMixFT* configuration yields relatively higher and more consistent correlations, suggesting that incorporating causally structured synthetic data mitigates some of this instability. Nonetheless, across most settings, validation performance remains an unreliable predictor of test performance, underscoring the need for more robust fine-tuning criteria for tabular foundation models. The general low generalization from validation to test performance is one of the main factors we identified for fine-tuning TFMs to generally be very challenging.

B Validation Performance and Overfitting Analysis

To complement the test-set evaluation, we analyze the normalized ROC-AUC performance on the validation sets across all generator configurations. Comparing validation and test performance provides insight into the degree of overfitting introduced during fine-tuning and the reliability of validation metrics as an early-stopping signal. A smaller discrepancy between validation and test performance indicates a more stable and trustworthy proxy for generalization.

Figure 4 summarizes validation performance and the corresponding ranks across generators. Figure 4a reveals that the *TableAugment* generator achieves the highest median normalized validation ROC-AUC, despite ranking among the weakest methods on the test set (Section C). This discrepancy



(a) **Validation ROC-AUC distribution across generator methods.** Each box represents the normalized validation performance relative to the pre-trained baseline.

(b) **Critical difference diagram (significance level = 0.05).** Lower ranks indicate better average performance across datasets.

Figure 4: **Validation performance comparison across generators.** The results reveal that validation-based ranking can be misleading under small-data conditions, with methods such as *TableAugment* and *Default* showing strong validation performance but large decrease in test performance and thus generalization. The *CausalMixFT* configuration demonstrates a smaller validation-test discrepancy, suggesting more stable and generalizable fine-tuning behavior.

suggests strong overfitting to the validation data, leading to poor generalization. Similarly, the default fine-tuning baseline configuration exhibits the second-highest validation performance but also a pronounced drop on the test set, with a median validation-test difference of 0.67 normalized units. In contrast, the *CausalMixFT* combination shows a much smaller difference of 0.30 units, indicating that SCM-based augmentation produces a more reliable and stable validation signal.

The critical difference diagram in Figure 4b further supports these observations. The *Default* baseline achieves the lowest (best) rank on validation performance, reflecting its overconfident behavior on small validation splits. However, this high validation ranking does not translate into superior test performance, reinforcing the conclusion that validation metrics can be misleading under data-scarce conditions. Across all methods, the majority of validation ROC-AUC scores exceed those of the pre-trained model, which to an extent is an expected outcome due to early stopping. Certain datasets display lower validation performance, relative to the pre-trained model, likely caused by divergence between the early-stopping criterion (log-loss) and the evaluation metric (ROC-AUC).

C Heterogeneity of Test Performance across Generators

To investigate how fine-tuning outcomes differ across data generation strategies, we analyze the normalized test ROC-AUC for each dataset and generator combination. This evaluation highlights the degree of heterogeneity in model performance and the dataset-specific behavior of each augmentation method. We note that the TabEBM generator is not compatible with the "MIC" dataset, as it internally relies on TabPFNv1, which only supports up to 100 features, while the dataset has 112 features.

Figure 5 presents the normalized test performance heatmap across all generators and datasets. Consistent with the findings in Section 4, we observe substantial variability in fine-tuning outcomes. While the *CausalMixFT* and *Default* configurations achieve strong and stable performance on average, their relative advantage varies across datasets. Some datasets favor purely synthetic approaches such as *CTGAN* or *TabEBM*, whereas others benefit most from hybrid or causally informed augmentation. The *CTGAN* generator achieves highes normalized performance of 3.89, while the TableAugment generator yields the lowest normalized performance of -2.79.

This heterogeneity suggests that the effectiveness of a generator is strongly dataset dependent and influenced by the underlying feature distribution, sample size, and causal complexity. The absence of a universally superior generator underscores the importance of adaptive fine-tuning strategies that can leverage multiple synthetic sources or dynamically adjust augmentation ratios based on dataset characteristics.

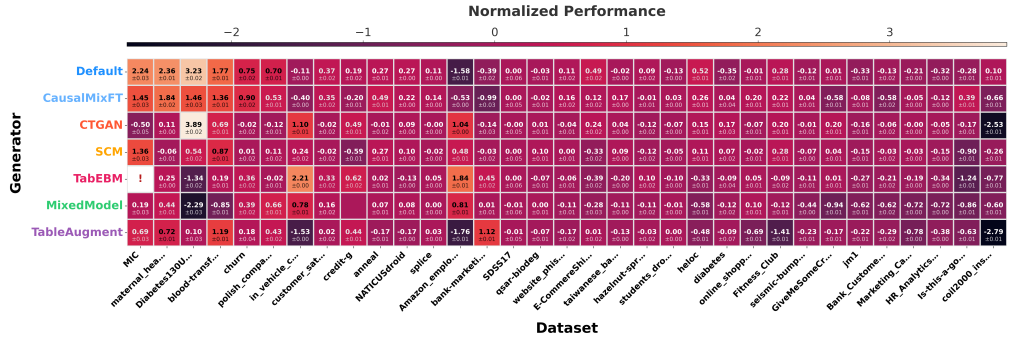


Figure 5: **Normalized test ROC-AUC performance across datasets and generator configurations.** Each cell reports the normalized ROC-AUC (mean ± standard deviation) for a given generator on a specific dataset. The observed heterogeneity indicates that fine-tuning performance varies considerably across generators and datasets, highlighting the need for dataset-adaptive augmentation strategies.

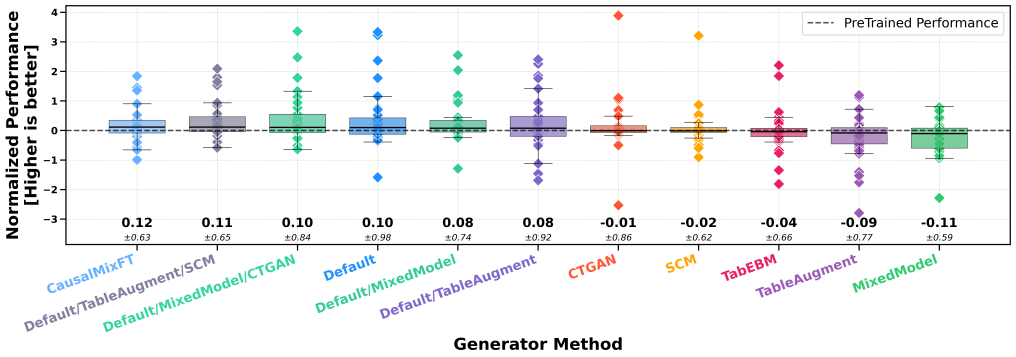


Figure 6: **Normalized test ROC-AUC performance across mixed generator configurations.** Each box represents the distribution of normalized test performance relative to the pre-trained model across datasets. Mixed configurations that combine real and synthetic data consistently outperform single-generator baselines.

D Combinations of Generators

Building on the strong performance of our proposed SCM-based augmentation, we extend our analysis to explore hybrid generator configurations that combine multiple data sources. Given the competitive baseline performance of the default fine-tuning setup (*Default*) relative to single synthetic generators, we systematically pair it with one or more synthetic generation methods to examine potential complementarity effects. We evaluate normalized test ROC-AUC performance across all datasets to assess whether combining generators yields more robust fine-tuning behavior.

As shown in Figure 6, hybrid configurations that mix real and synthetic data achieve consistently strong results across datasets. The five mixed-generator variants occupy the top six median performance positions, with *CausalMixFT* achieving the highest median improvement of **+0.12** (±0.63), followed closely by *Default/TableAugment/SCM* (**+0.11** ±0.65). The latter finding is particularly noteworthy, as *TableAugment* alone performs poorly when used in isolation. This suggests that combining heterogeneous data sources enables the foundation model to leverage complementary structural and distributional properties—an effect reminiscent of ensemble learning.

Furthermore, the *Default/MM* configuration exhibits both competitive median performance (**+0.10**) and a notably small interquartile range (1.5 ×IQR) shown by the whiskers, indicating stable and predictable improvements across datasets. Such stability makes it a strong candidate when reliable gains over the pre-trained baseline are prioritized alongside generalization consistency. Overall, these results demonstrate that mixing real data with diverse synthetic generators enhances fine-tuning

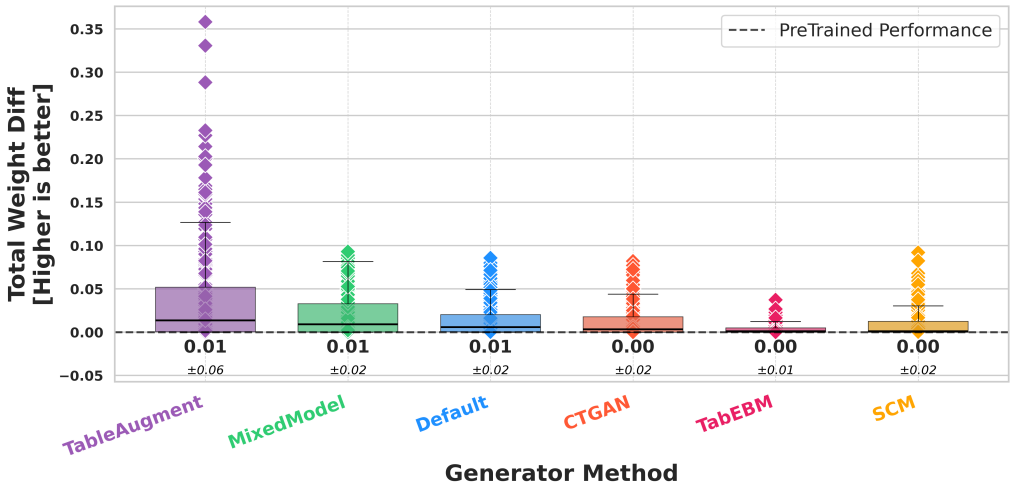


Figure 7: **Total parameter distance between fine-tuned and pre-trained model weights across generator configurations.** Each box represents the distribution of elementwise Euclidean weight differences across all datasets. Smaller values indicate that the fine-tuned model remains closer to the pre-trained parameter space.

robustness and generalization, highlighting the potential of multi-generator augmentation for tabular foundation models operating in the low-data regime.

E Fine-Tuned Weight Distance from the Pre-Trained Model

We analyze how far the fine-tuned model weights deviate from the pre-trained checkpoint across different generator configurations. Specifically, we compute the elementwise Euclidean distance between the fine-tuned parameters and the original pre-trained weights. Prior work has suggested that constraining the degree of weight divergence through regularization (e.g. euclidean distance) can improve fine-tuning stability and generalization [10, 21]. By quantifying the total weight displacement per generator, we aim to understand how different augmentation strategies influence model adaptation dynamics and parameter stability.

As shown in Figure 7, the *TableAugment* generator exhibits the largest variance in weight displacement from the pre-trained model. Although its median distance is relatively low (approximately 0.01), several runs show extreme deviations up to 0.35, indicating unstable optimization behavior. Combined with its previously observed weak generalization performance, this pattern is consistent with phenomena associated with *catastrophic forgetting* [19, 3, 11], where fine-tuning overwrites pre-trained representations, leading to significant loss of learned capabilities.

In contrast, the *MixedModel*, *Default*, and *CTGAN* configurations display similar median distances (around 0.01) but diverge substantially in their downstream performance, suggesting that weight distance alone is an unreliable indicator of fine-tuning success. Notably, the *TabEBM* and *SCM* generators show almost no displacement from the pre-trained weights, implying that their training signals were too weak or misaligned to induce meaningful parameter updates. This observation highlights that small weight changes do not necessarily imply better generalization, emphasizing the importance of evaluating both representational stability and downstream performance jointly.

F Layer wise Weight Adaptation

We next examine which components of the TFM undergo the largest updates relative to the pre-trained checkpoint. The pre-trained model has been optimized on a broad distribution of purely synthetic tasks, which encourages storage of general representational structure rather than dataset-specific heuristics. Fine-tuning, in contrast, repeatedly exposes the model to related (or even equal) samples

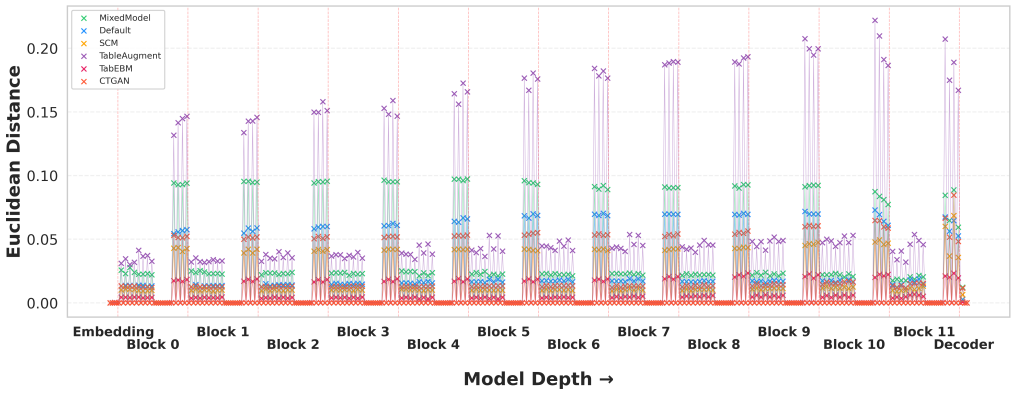
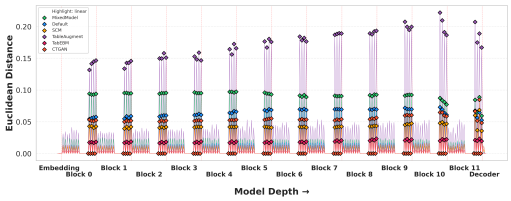
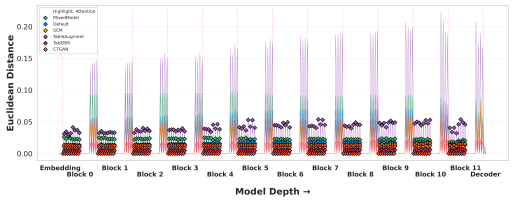


Figure 8: **Layerwise parameter drift by generator.** Euclidean distance between fine-tuned and pre-trained parameters, grouped by module along model depth from left to right (embedding, transformer blocks 0 through 11, decoder head). Markers correspond to distinct parameter groups within each block.



(a) **Feed-forward (linear) layers.** Weight distance across model depth per generator.



(b) **Attention layers.** Weight distance across model depth per generator.

Figure 9: **Component-wise weight deviation across model depth.** Comparison of parameter shifts for feed-forward (linear) and attention layers relative to the pre-trained model. Linear layers show the strongest adaptation across all generators, followed by the attention layers.

from a generator, which promotes specialization towards these samples. Quantifying where parameter drift concentrates within the network can therefore provide insight into how specialization emerges.

Figure 8 shows the distribution of layerwise weight distances for each generator. The *TableAugment* configuration displays a pronounced depth trend: distances increase toward later blocks and the decoder head, indicating strong adaptation in task-specific layers. A similar pattern, though less pronounced, appears for the *Default*, *SCM*, and *CTGAN* settings. In contrast, *MixedModel* and *TabEBM* show relatively flat profiles with smaller shifts across depth. These observations align with the common view that early layers encode general computations while deeper layers encode task-specific transformations; specialization during fine-tuning therefore concentrates in later blocks.

Coupling these results with Section B suggests a practical implication. When depth-wise drift is steep and accompanied by weak test generalization, as observed for *TableAugment*, regularization that limits deviation from the pre-trained manifold may improve stability of early stopping and reduce overfitting.

G Component-Wise Weight Adaptation

The layer wise analysis in Figure 8 indicates that certain model components undergo substantial parameter updates during fine-tuning, whereas others remain largely unchanged across generators. To investigate this in greater detail, we isolate and compare the two component groups exhibiting the strongest deviations from the pre-trained checkpoint.

Figure 9 highlights that the *linear layers*, including the feed-forward networks between each transformer block. We observe that especially the feed-forward networks between the attention computation undergo the largest parameter shifts across all generator configurations, while the decoder

head has (not marked on the plot) only minimal distance compared to the pre-trained model weights. This suggests that most dataset-specific specialization is concentrated in the linear layers, while other components retain more general representations. These findings point to a potential strategy for mitigating overfitting: selectively constraining the deviation of linear layer parameters from the pre-trained checkpoint may preserve adaptability in other components while reducing excessive specialization. Such a targeted regularization offers an interesting direction for future research.

The *attention layers* (key, query, value, and output projection matrices) exhibit the second-largest parameter shifts, consistent with earlier findings that fine-tuning induces measurable but in our case less pronounced adaptation in attention modules [31]. This supports the hypothesis that attention mechanisms encode more generalizable computations that remain relatively stable across datasets. Notably, the extent of attention-layer adaptation varies strongly with the generator type: *TableAugment* produces the largest deviations, followed by the *MixedModel*, while *Default*, *CTGAN*, *TabEBM*, and *SCM* lead to comparatively minor changes. These results further reinforce that the fine-tuning signal introduced by different generators differentially influences the degree to which the model parameters are adjusted.

H Performance Normalization

to compare the performance across different data generators, we apply the normalization strategy suggested by Gorishniy et al. [12]. We choose the base model's (Mitra's) zero-shot performance as performance baseline, to measure the improvement after fine-tuning over the pre-trained model. To normalize the performance of the fine-tuned model we compute: $score_{normalized} = metric_{sign} \times (\frac{score_{method}}{score_{baseline}} - 1) \times 100\%$, where $metric_{sign} = 1$ for metrics, where higher is better (e.g. ROC-AUC) and $metric_{sign} = -1$ for metrics, where lower is better (e.g. Log-loss). If fine-tuning and the pre-trained models achieve the same performance then $score_{normalized} = 0$, if fine-tuning improves over the pre-trained model then $score_{normalized} > 0$ and if fine-tuning decreases performance then $score_{normalized} < 0$. The choice of normalization method allows averaging the normalized performance across datasets and compare the data generating methods.

I Notation.

We denote the real downstream dataset as D^{real} and the synthetically generated dataset as D^{syn}. Each dataset is partitioned into training, validation, and test subsets, represented as $D_{\text{train}}^{\{\text{real/syn}\}}$, $D_{\text{val}}^{\{\text{real/syn}\}}$, and $D_{\text{test}}^{\{\text{real/syn}\}}$, respectively. For context–query splits used during in-context fine-tuning, we write $D_{\text{ctx}}^{\{\text{real/syn}\}}$ and $D_{\text{query}}^{\{\text{real/syn}\}}$. Given that each dataset contains a predefined target column, we represent the non-target feature matrix as $X_{\text{train}}^{\{\text{real/syn}\}} \in \mathbb{R}^{n \times d}$ and the corresponding target vector as $Y_{\text{train}}^{\{\text{real/syn}\}} \in \mathbb{R}^{n \times 1}$. As there is a optional normalization step, between the training samples and the generator data, we sometimes write $D_{\text{generator}}$, to specify this. This notation is used consistently throughout the methodology and experimental sections.

J Generator Details

In this section, we describe the different data-generating methods, which are used for the experiments. We start with our baseline, which uses D_{train}^{real} directly, followed by a heuristic method and then go into the methods, optimized on the dataset.

J.1 Default Generator (baseline)

In our experiments, fine-tuning directly on the raw training data without any form of augmentation represents our baseline. This approach reflects the standard practice of utilizing available data for fine-tuning, as employed in prior work [4, 8, 31]. Since the experiments focus on tiny-to-small datasets, all foundation models considered are capable of processing the entire dataset within one forward pass, without requiring any context retrieval mechanisms.

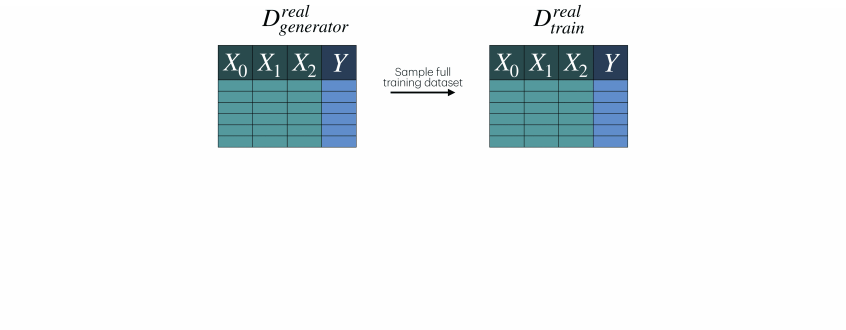


Figure 10: Fine-tuning on training data. The original training dataset is directly utilized for fine-tuning the model.

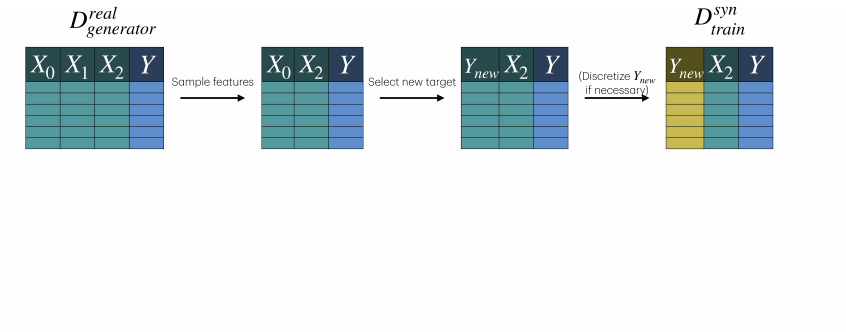


Figure 11: The TableAugment Generator. Real data features are green and blue, while synthetized features are orange and yellow.

J.2 TableAugment Generator

The TableAugment generator is inspired by the data augmentation approach introduced by Ma et al. [24], who used real-world tabular datasets to pre-train a TabPFN variant. Due to the limited number of publicly available tabular datasets, the authors augmented their collection using different views of each dataset by subsampling and shuffling features and random selection of a target column in each iteration.

In contrast to their pre-training method, our goal is to improve the model's performance on one specific target dataset, and therefore we only need to augment the same dataset over and over again. To this end, our implementation supports a range of configurable augmentation strategies involving feature subselection and target column assignment.

Feature Subseletion. Feature subselection can be toggled on or off. If disabled, the model uses all features from the generator dataset $D^{real}_{\text{generator}}$. If enabled, a subset of features is selected in each iteration by uniformly sampling a proportion of features between 50% and 100%. Additionally, we provide control over the inclusion of the original target column (hereafter referred to as the "old target") within the selected feature subset. The old target can be configured to be always included, never included, or included with the same probability as all other features.

Target Column Selection. After the feature subselection, we assign a new target column for the foundation model to predict in this iteration. This functionality can also be enabled or disabled. If disabled, the old target remains the target throughout all iterations. If enabled, a new target column is sampled randomly from the set of selected features. The inclusion of the old target column in the candidate pool for new targets is controllable: it can always be included, never included, or included at random. If the newly selected target column is continuous or exhibits high cardinality, it may need to be discretized. The number of discretized classes, denoted by $\hat{c}$, can either match the number of classes in the old target or be sampled uniformly from a user-specified range (default range is between 2 and 10). To discretize a target column into $\hat{c}$ classes, we assign the $\hat{c} - 1$ most frequent values to individual classes and group all remaining values into the $\hat{c}^{\text{th}}$ class.

The new resulting dataset D^{syn} is the composed of the subselected features and the new selected target column.

J.3 Mixed-Model Generator

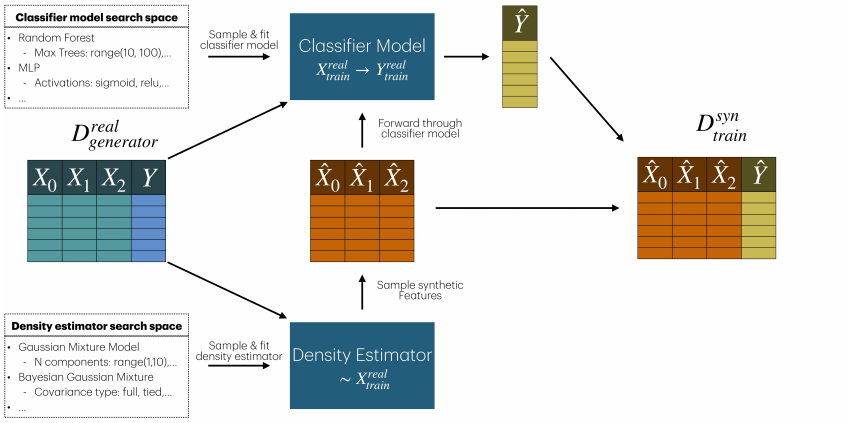


Figure 12: The Mixed-Model generator. First the internal density estimator and classifier model are sampled from predefined search spaces and fitted on $D^{real}_{generator}$. Secondly, to generate synthetic data, we sample features from the density estimator and propagate the samples through the classifier, which yields the labels. The features and labels are concatenated, which results in the synthetic dataset.

The Mixed-Model Generator is the first proposed augmentation method that incorporates learnable internal models to generate synthetic datasets, as shown in figure 12. Breejen et al. [4] showed that during pre-training data generated through machine learning models can capture complex feature-target relations and thus is very efficient for improving in-context learning capabilities. Based on this insight, we create a generator, which leverages internal machine learning models to generate synthetic datasets, which incorporate information about the real feature to target mapping. It consists of two primary components:

- A density estimator, which models the distribution of the feature space.
- A classifier, which learns mapping between the features and targets.

Together, these components are used to produce labeled synthetic datasets. The generator exposes a range of hyperparameters that control the behavior of both components, as described below.

Density Estimator. We support four types of density estimators: Gaussian Mixture Model (GMM), Bayesian Gaussian Mixture (BGM), Kernel Density Estimation (KDE), and Uniform Density Model. The first three options use the implementations provided by *scikit-learn*, while the uniform estimator is custom implemented. For the uniform estimator, continuous features are sampled from a uniform float distribution bounded by the observed range in the training data, while categorical features are sampled uniformly from the set of observed integer values.

A special case is handled for the Bayesian Gaussian Mixture model. When the covariance matrix becomes singular (which can occur in highly imbalanced datasets), we iteratively increase the *cov_reg*

parameter by a factor of 10, up to 10 times. If this fails to resolve the issue, we default to using the uniform density estimator.

Classifier Model. Once the density estimator is selected and fitted, we sample a classifier model from the following set: Decision Tree (DT), Random Forest (RF), Gradient Boosted Trees (GradBoost), Support Vector Classifier (SVC) and Multi-Layer Perceptron (MLP). Each classifier is associated with a predefined set of hyperparameters, which are shown in (TODO).

Synthetic Data Generation. To generate synthetic data, we first sample a density estimator with corresponding hyperparameters and a classifier with corresponding hyperparameters. The density estimator is trained on the generator's real feature set, $X^{\text{real}}_{\text{generator}}$, while the classifier is trained using both the real features $X^{\text{real}}_{\text{generator}}$ and the corresponding real targets $Y^{\text{real}}_{\text{generator}}$. Further, we sample $\hat{n}_i$ new feature samples from the density estimator (default: 20,000). These features are forwarded through the trained classifier to produce the corresponding synthetic labels Y^{syn}, forming a complete synthetic dataset $(X^{\text{syn}}_{train}, Y^{\text{syn}}_{train})$.

J.4 SCM-Based Generator

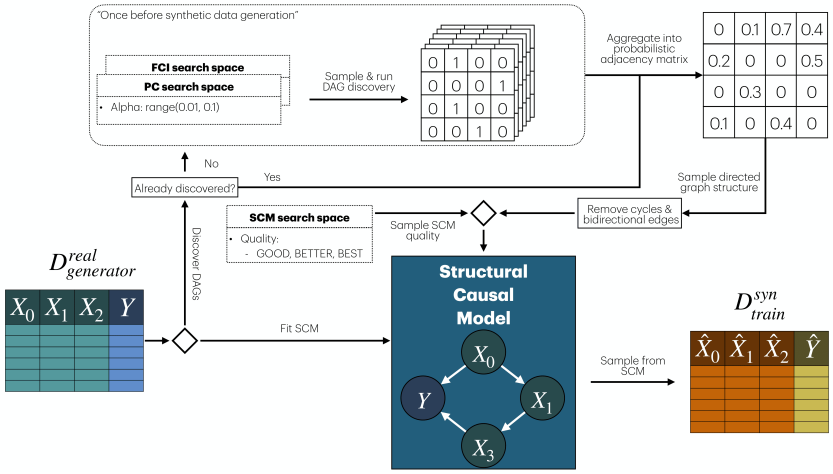


Figure 13: The SCM generator consists of two phases. First, discovering the structural relationships between the features, and secondly sampling DAGs and fitting SCMs on the generator dataset $D^{real}_{generator}$.

In this method, we estimate dependencies between features through structure discovery and use Structural Causal Models (SCMs) to generate synthetic datasets. The process consists of two phases: first we discover the structural relationships between the features, and secondly we sample and fit SCMs from which we sample synthetic data.

Structural Dependencies Discovery. In the first phase, which we only have to do once per fine-tuning run, we want to find structural dependencies between the features. Therefore, we apply the Peter-Clark (PC) and Fast Causal Inference (FCI) algorithms from the `causal-learn` library, each executed 50 times with differently sampled hyperparameters, resulting in a total of 100 discovery runs. The procedure is as follows:

- Each run is limited to a maximum runtime of 20 minutes to avoid infinite loops or long convergence times.
- The input data is subsampled to a maximum of 1,000 rows and 50 columns if these thresholds are exceeded.
- Each run returns an adjacency matrix that indicates detected edges between features.
- These 100 adjacency matrices are aggregated into a probabilistic adjacency matrix C, where each cell $c_{i,j}$ denotes the relative frequency with which an edge from feature i to feature j was discovered:

$$c_{i,j} = \frac{\text{Number of runs where edge } i \to j \text{ was found}}{\text{Total number of runs}}.$$

Although PC and FCI are typically used to recover causal graphs under strict assumptions, our use case only requires discovery of correlational structure. As such, potential violations of assumptions are acceptable, and the discovered graphs are treated as representations of meaningful (though not necessarily causal) dependencies. This step is performed once during initialization. Section 18 shows an exemplary probabilistic adjacency matrix.

SCM Fitting and Data Generation. Once the probabilistic adjacency matrix is computed, the following steps are performed each time a synthetic dataset is generated:

- **Graph Sampling:**
  - A directed graph is sampled from the probabilistic adjacency matrix C.
  - Bidirectional edges are resolved by randomly removing one direction.
  - Cycles are removed by randomly deleting edges until the graph becomes a Directed Acyclic Graph (DAG).
- **SCM Fitting:**
  - The resulting DAG is passed to DoWhy's SCM fitting API, which fits an additive noise model using the structure and the generator data $D^{real}_{\text{generator}}$.
  - The fitting quality can be configured via a `quality` parameter, with the following options:
    - **GOOD:** Fast and simple models.
      - Numerical: Linear regressors (with/without polynomial features), Histogram Gradient Boost Regressor.
      - Categorical: Logistic regression (with/without polynomial features), Histogram Gradient Boost Classifier.
    - **BETTER:** Wider model variety for better accuracy.
      - Numerical: Adds Ridge, Lasso, Random Forest, SVR, Extra Trees, KNN, AdaBoost.
      - Categorical: Adds Random Forest, Extra Trees, SVC, KNN, Gaussian Naive Bayes, AdaBoost.
    - **BEST:** Uses `AutoGluon` (AutoML). Offers highest accuracy but slower training and inference.

Synthetic samples are generated using DoWhy's API by drawing noise for exogenous variables and propagating it through the SCM. The default number of samples per synthetic dataset is set to 20,000.

J.5 TabEBM Generator

The TabEBM generator is a class conditional generating method. Using the TabEBM official implementation, this returns data, where each class is present with the same number of samples. To have a dataset, which is more representative of the real training data, we subsample from the synthetic dataset, such that the class distribution of the real training dataset is maintained. After this, the subsampled dataset is returned. As the sampling of TabEBM works purely through in-context learning, the TabEBM generator needs to be fitted every time we generate a new synthetic dataset.

J.6 CTGAN Generator

For the CTGAN generator, before generating any synthetic data, we fit the GAN on the training data using a set of sampled hyperparameters from a predefined search space. Once the GAN model is fitted, we can directly sample synthetic data from it (default 20,000 samples), which represent the synthetic dataset.

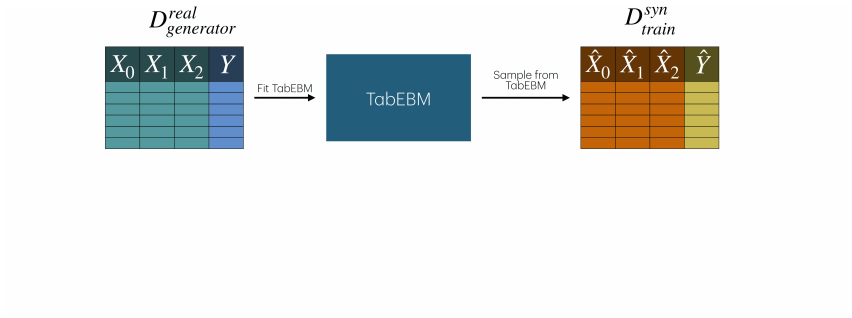


Figure 14: The TabEBM generator.

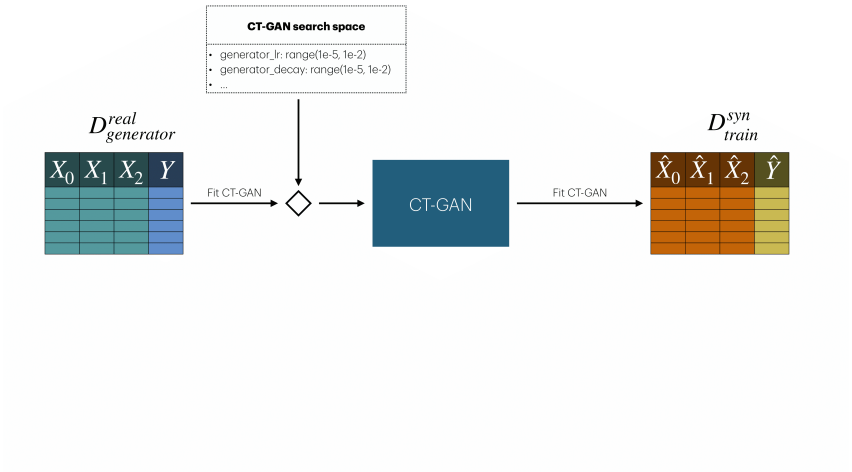


Figure 15: The CT-GAN generator.

K Generator Hyperparameters

K.1 MixedModel Generator Hyperparameters

Table 1: Overview of hyperparameter ranges for classifiers used in the *MixedModel generator*.

Classifier	Hyperparameter	Type / Choices	Range or Values
TabPFNClassifier	`n_estimators`	Integer (Uniform)	[1, 10]
	`n_jobs`	Categorical	{1}
	`device`	Categorical	{"cpu"}
RandomForestClassifier	`n_estimators`	Integer (Log-Uniform)	[10, 500]
	`criterion`	Categorical	{"gini", "log_loss", "entropy"}
	`max_depth`	Integer (Log-Uniform)	[10, 100]
	`min_samples_split`	Integer (Uniform)	[2, 20]
	`min_samples_leaf`	Integer (Uniform)	[1, 10]
	`max_leaf_nodes`	Integer (Uniform)	[10, 100]
	`bootstrap`	Categorical	{True, False}
DecisionTreeClassifier	`criterion`	Categorical	{"gini", "entropy", "log_loss"}
	`splitter`	Categorical	{"best", "random"}
	`max_depth`	Integer (Log-Uniform)	[5, 100]
	`min_samples_split`	Integer (Uniform)	[2, 20]
	`min_samples_leaf`	Integer (Uniform)	[1, 10]
	`max_features`	Categorical	{0.1, 0.25, 0.5, 0.75, 1.0, "sqrt", "log2", None}
MLPClassifier	`hidden_layer_sizes`	Integer (Uniform)	[1, 100]
	`activation`	Categorical	{"relu", "logistic", "tanh"}
	`solver`	Categorical	{"adam", "sgd", "lbfgs"}
	`alpha`	Float (Uniform)	[0.0001, 0.1]
	`batch_size`	Categorical	{"auto", 32, 64, 128}
	`learning_rate`	Categorical	{"constant", "invscaling", "adaptive"}
	`learning_rate_init`	Float (Uniform)	[0.0001, 0.01]
	`max_iter`	Integer (Uniform)	[100, 1000]
	`momentum`	Float (Uniform)	[0.5, 0.95]
	`nesterovs_momentum / early_stopping`	Categorical	{True, False}
SVC	`kernel`	Categorical	{"linear", "rbf", "poly", "sigmoid"}
	`C`	Float (Log-Uniform)	[1e-6, 1e6]
	`degree`	Integer (Uniform)	[1, 5]
	`gamma`	Categorical	{"scale", "auto"}
	`coef0`	Float (Uniform)	[-1, 1]
	`shrinking`	Categorical	{True, False}
	`probability`	Categorical	{True, False}
	`tol`	Float (Log-Uniform)	[1e-5, 1e-2]
	`cache_size`	Float (Uniform)	[200, 1000]
	`class_weight`	Categorical	{None, "balanced"}
	`max_iter / break_ties`	Integer / Bool	[100, 1000] / {True, False}
HistGradientBoostingClassifier	`loss`	Categorical	{"log_loss"}
	`learning_rate`	Float (Uniform)	[0.01, 1.0]
	`max_iter`	Integer (Uniform)	[50, 1000]
	`max_leaf_nodes`	Integer (Uniform)	[5, 100]
	`max_depth`	Integer (Uniform)	[3, 15]
	`min_samples_leaf`	Integer (Uniform)	[5, 100]
	`l2_regularization`	Float (Uniform)	[0.0, 1.0]
	`max_bins`	Integer (Uniform)	[10, 255]

K.2 SCM Generator Hyperparameters

Table 2: Overview of the `quality` hyperparameter for internal model assignment in the *SCM generator*.

Quality Setting	Included Models (Examples)	Description / Characteristics
`GOOD`	Numerical: Linear Regressor, Polynomial Regressor, Histogram Gradient Boost Regressor Categorical: Logistic Regressor, Polynomial Logistic Regressor, Histogram Gradient Boost Classifier	Small, efficient model set for fast training and inference; medium predictive accuracy.
`BETTER`	Numerical: Ridge, Lasso, Random Forest, SVR, Extra Trees, KNN, AdaBoost Categorical: Random Forest, Extra Trees, SVC, KNN, GaussianNB, AdaBoost	Expanded model pool for higher accuracy while maintaining reasonable training speed.
`BEST`	AutoML backend (AutoGluon)	Full AutoML configuration offering the best accuracy, but with increased computational cost.

K.3 TabEBM Generator Hyperparameters

Table 3: Overview of hyperparameter ranges for the *TabEBMGenerator*.

Generator	Hyperparameter	Type / Choices	Range or Values
TabEBMGenerator	`n_samples_per_class`	Integer (Fixed)	150
	`device`	Categorical	{"cpu"}
	`name`	Categorical	{"TabEBMGenerator"}

K.4 CTGAN Generator Hyperparameters

Table 4: Overview of hyperparameter ranges for the *CTGANGenerator*.

Generator	Hyperparameter	Type / Choices	Range or Values
CTGANGenerator	`refit_interval`	Integer (Fixed)	10
	`n_synthetic_samples`	Integer (Fixed)	20,000
	`n_sample_attempts`	Integer (Fixed)	10
	`model_cache_lower_bound`	Integer (Fixed)	2
	`model_cache_upper_bound`	Integer (Fixed)	5
	`cuda`	Categorical	{True}
	`embedding_dim`	Integer (Uniform)	[8, 256]
	`generator_lr`	Float (Log-Uniform)	[1e-5, 1e-2]
	`generator_decay`	Float (Log-Uniform)	[1e-5, 1e-2]
	`discriminator_lr`	Float (Log-Uniform)	[1e-5, 1e-2]
	`discriminator_decay`	Float (Log-Uniform)	[1e-5, 1e-2]
	`discriminator_steps`	Integer (Uniform)	[1, 10]
	`epochs`	Integer (Uniform)	[100, 200]

K.5 TableAugment Generator Hyperparameters

Table 5: Overview of hyperparameter ranges for the *TableAugmentGenerator*.

Generator	Hyperparameter	Type / Choices	Range or Values
TableAugmentGenerator	`name`	Categorical	{"TableAugmentGenerator"}
	`normalize`	Categorical	{False}
	`sub_sample_features.active`	Categorical	{True}
	`sub_sample_features.min_ratio`	Float (Uniform)	[0.5, 1.0]
	`sub_sample_features.max_ratio`	Float (Uniform)	[0.5, 1.0]
	`sub_sample_features.include_target`	Categorical	{"random", "always", "never"}
	`random_sample_target.active`	Categorical	{True}
	`random_sample_target.include_target`	Categorical	{"random", "always", "never"}
	`random_sample_target.allow_target_as_target`	Categorical	{True}
	`random_sample_target.use_dataset_num_classes`	Categorical	{True}
	`random_sample_target.min_discrete_values`	Integer (Fixed)	2
	`random_sample_target.max_discrete_values`	Integer (Fixed)	10

L Data-Splitting

Given a target dataset D^{real}, we partition it into three mutually exclusive subsets: training, validation, and test. Formally, this decomposition is expressed as:

$$D^{\text{real}} = D^{\text{real}}_{\text{train}} \cup D^{\text{real}}_{\text{val}} \cup D^{\text{real}}_{\text{test}}, \quad \text{with} \quad D^{\text{real}}_{\text{train}} \cap D^{\text{real}}_{\text{val}} = \emptyset, \quad D^{\text{real}}_{\text{train}} \cap D^{\text{real}}_{\text{test}} = \emptyset, \quad D^{\text{real}}_{\text{val}} \cap D^{\text{real}}_{\text{test}} = \emptyset.$$

To evaluate the performance of fine-tuning under limited data conditions, we constrain the dataset sizes used for training and validation. Including, if the total number of samples N^{real} exceeds 1000, we apply truncation as follows:

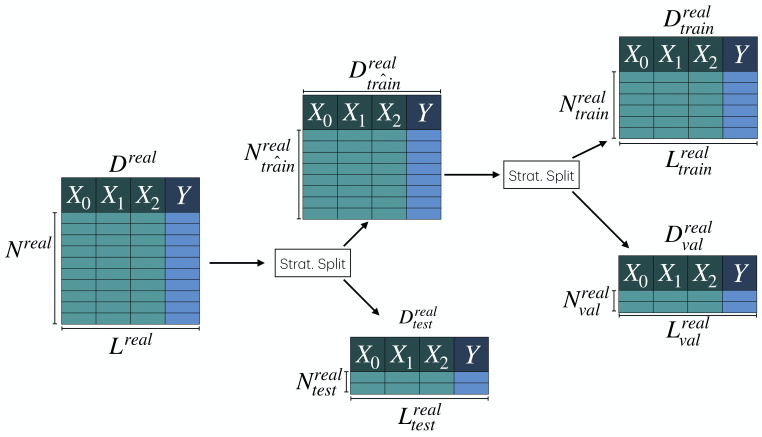


Figure 16: Splitting the full dataset into a train, validation and test set.

- The training set $D^{\text{real}}_{\text{train}} \in \mathbb{R}^{N^{\text{real}}_{\text{train}} \times L^{\text{real}}}$, with
$$N^{\text{real}}_{\text{train}} = \min(0.6 \cdot N^{\text{real}}, 600).$$
- The validation set $D^{\text{real}}_{\text{val}} \in \mathbb{R}^{N^{\text{real}}_{\text{val}} \times L^{\text{real}}}$, with
$$N^{\text{real}}_{\text{val}} = \min(0.2 \cdot N^{\text{real}}, 200).$$
- The test set $D^{\text{real}}_{\text{test}} \in \mathbb{R}^{N^{\text{real}}_{\text{test}} \times L^{\text{real}}}$, where
$$N^{\text{real}}_{\text{test}} = \max(0.2 \cdot N^{\text{real}}, N^{\text{real}} - N^{\text{real}}_{\text{train}} - N^{\text{real}}_{\text{val}}).$$

Regarding the validation split, we first apply a test, train split, and then get the train, val split only based on train. This procedure ensures that the training and validation subsets comprise at most 60% and 20% of the full dataset, capped at 600 and 200 samples respectively. The test set comprises the remaining data, ensuring at least 20% coverage.

To preserve the distribution of class labels, all splits are generated using stratified sampling with respect to the class labels. This partitioning process is repeated across K different folds to support robust evaluation. The overall splitting strategy is illustrated in Figure 16.

M Data Pre-processing

Having obtained the training, validation, and test splits, we proceed with data pre-processing to ensure compatibility with both the data-generating procedures and the foundation model.

For the foundation model, all non-numerical (categorical or textual) features are encoded into numerical representations to enable model compatibility. This step is essential for ensuring that the input conforms to the expected numerical format of the foundation model. For this, we use Autogluon's *AutoMLPipelineFeatureGenerator*, with the default setting.

Further, for the data-generating methods, we create a working copy of the training dataset $D^{\text{real}}_{\text{train}} \rightarrow D^{\text{real}}_{\text{generator}}$. On this cloned dataset, we perform the following pre-processing steps:

- **Mean imputation** is applied to continuous (real-valued) features to handle missing values.
- **Mode imputation** is used for categorical features, ensuring that missing entries are filled with the most frequent category.
- Optionally, **Z-score normalization** is applied to standardize the input distribution.

During the fine-tuning we use $D^{\text{real}}_{\text{generator}}$ to build our generators, and $D^{\text{real}}_{\text{train}}$, $D^{\text{real}}_{\text{val}}$ to assess validation performance. This pre-processing workflow is illustrated in Figure 17.

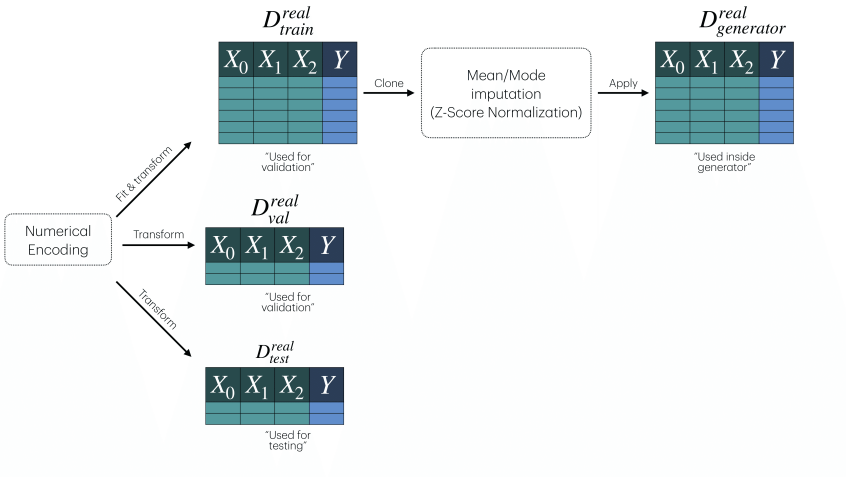


Figure 17: Pre-processing the data. Non-numerical features and numerically encoded. Further, the training data is cloned for the data generating methods with mean/mode imputation and optional z-score normalization applied.

N Fine-tuning Hyperparameters

Table 6: Default parameter values of the training hyperparameter configuration.

Parameter	**Default Value**
`initial_learning_rate`	1×10^{-4}
`finetune_steps`	50
`shuffle_classes`	False
`shuffle_features`	False
`use_random_transforms`	False
`random_mirror_x`	True
`patience`	40

O Probabilistic Adjacency Matrix

In the first step of our method, we apply a set of causal discovery algorithms with varying hyperparameters and aggregate their outputs into a probabilistic adjacency matrix. Each matrix entry represents the relative frequency of a directed edge across all discovered graphs.

Figure 18 illustrates an example probabilistic adjacency matrix for the Is-this-a-good-customer dataset (fold 5). Qualitatively, we observe that most matrices are relatively sparse across datasets, consistent with prior findings on sparse real-world data structures [13]. This sparsity facilitates efficient causal structure learning even in high-dimensional settings (up to 200 features in our setup).

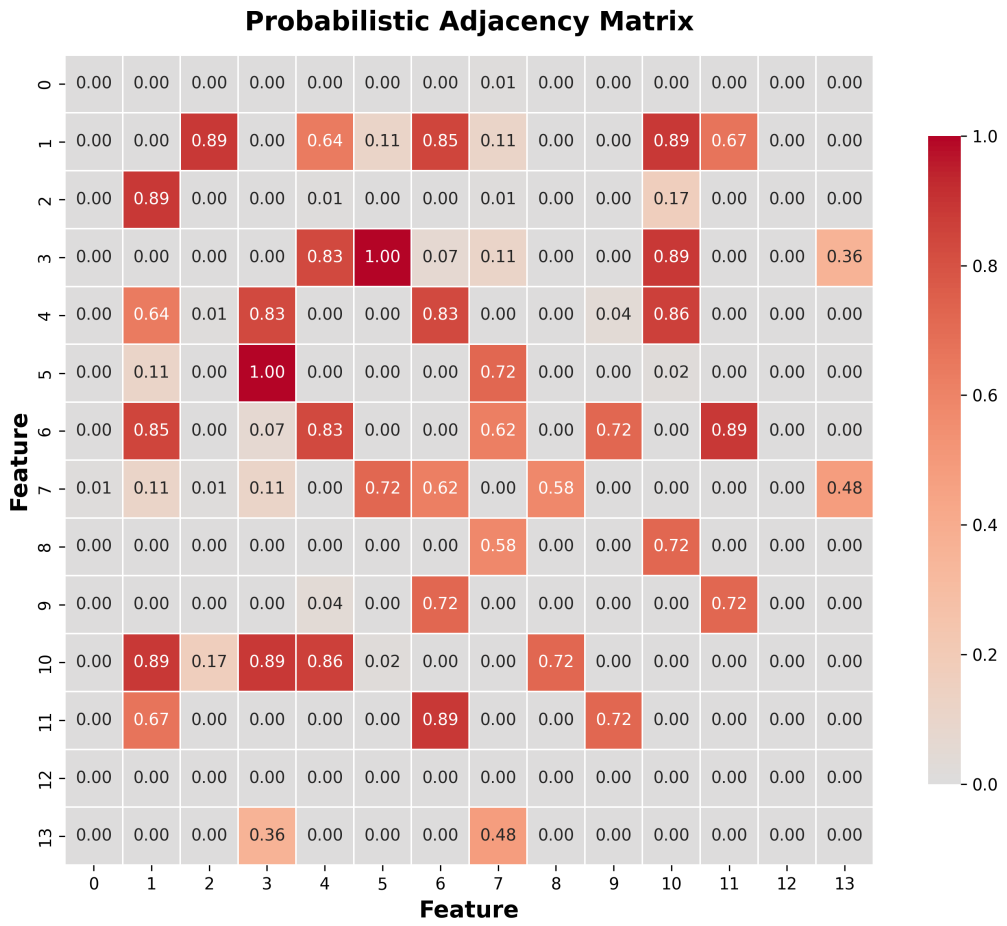


Figure 18: Example probabilistic adjacency matrix for the *Is-this-a-good-customer* dataset (fold 5). Each entry encodes the empirical frequency of a directed edge across multiple causal discovery runs with varying hyperparameters. Brighter values indicate higher consensus about the presence and direction of an edge. The overall sparsity reflects the typically sparse causal structure of real-world tabular data.