Towards Synthetic Data for Fine-tuning Tabular Foundation Models

Magnus Bühler [1] **Lennart Purucker** [1] **Frank Hutter** [1 2 3]

Abstract

Tabular foundation models pre-trained on synthetically generated datasets have exhibited strong in-context learning capabilities. While fine-tuning can further enhance predictive performance, overfitting to the training data of a downstream task poses a significant risk in tiny-to-small data regimes. We propose a fine-tuning method that employs synthetically generated fine-tuning data to avoid overfitting and improve generalization performance. We study three variants of data generation methods and empirically demonstrate that they mitigate overfitting and outperform standard fine-tuning approaches across five tiny-to-small real-world datasets. Our data generation methods leverage density estimators and structural causal models, akin to those employed during pre-training, to yield the best performance. Our findings indicate that synthetic data generation, a central element in pre-training, can be successfully adapted to enhance fine-tuning.

1. Introduction

Deep learning methods have recently begun to rival tree-based models in the domain of tabular classification, demonstrating competitive performance on a wide range of benchmarks. A particularly promising line of work leverages transfer learning across diverse datasets, often referred to as foundation models (Hollmann et al., 2025; Müller et al., 2023; Qu et al., 2025; Hollmann et al., 2022). One pivotal example is TabPFN, a pre-trained transformer architecture, using in-context learning (ICL). The model is given a set of correctly labled context samples and predicts class probabilities for query samples in a single forward pass. TabPFN bypasses the limited avaiability of publicly accessible, high-quality, tabular data sets by purely pre-training on synthetically generated data (Hollmann et al., 2025).

[1]University of Freiburg [2]Prior Labs, Freiburg, Germany [3]ELLIS Institute Tübingen, Tübingen, Germany. Correspondence to: Magnus Buehler <buehlema@informatik.uni-freiburg.de>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

Despite the strong zero-shot performance of such models, further performance can often be achieved by fine-tuning them for each target dataset (den Breejen & Yun, 2025; Breejen et al., 2024; Thomas et al., 2024a)[1]. However, fine-tuning tabular foundation models remains challenging due to the tiny-to-small size of most real-world tabular datasets which can lead to overfitting to the training data after only a few gradient steps and consequently poor predictive performance when generalizing to unseen data. A common mitigation strategy is to divide the training data into training and validation subsets, using validation performance as a proxy for generalization, and selecting the model checkpoint with the best validation score. However, this approach further reduces the limited number of available training samples, constraining the model's ability to learn the full complexity of the underlying data distribution. As a result, fine-tuning may cause the model to learn only from a simplified or biased representation of the data distribution, ultimately worsening its performance on unseen data.

To address this issue, we propose fine-tuning with synthetically generated data, a method for fine-tuning tabular foundation models on tiny-to-small datasets. Contrary to pre-training, the goal of this fine-tuning setting is to improve generalization performance for a single dataset. Inspired by the success of transfer learning across datasets during pre-training, we use data-generating models tailored to the training data distribution to enable efficient sampling of synthetic datasets. These synthetic datasets retain the structural characteristics of the target distribution while mitigating the problem of overfitting by allowing us to fine-tune on more data, improving performance on the target dataset.

Empirically, we demonstrate that conventional fine-tuning, using the training dataset directly, often does not improve performance on the test set, compared to the pre-trained models. Further, we show that fine-tuning with data generated through our methods often yields superior test performance compared to conventional fine-tuning and improves the downstream performance compared to the pre-trained model. Especially in data-scarce applications fine-tuning on synthetically generated data poses a promising solution to improve predictive performance without the drawbacks of

[1]AutoGluon provides an accessible fine-tuning API for their TabPFN variant TabPFN-Mix

too little training data.

2. Related Work

Recent studies have explored various methods for fine-tuning TabPFN: updating all weights (Breejen et al., 2024; den Breejen et al., 2023; Thomas et al., 2024b), training only a tokenizer layer (Liu et al., 2024), learning a compressed data representation (Feuer et al., 2024; Ma et al., 2024b), training separate encoders and a routing mechanism Xu et al. (2024), or training an ensemble encoder (Liu & Ye, 2025). All these studies have fine-tuned TabPFN on the training data of downstream tasks. In contrast, we keep the method static and change the data used for fine-tuning. Several works have explored tabular data synthesis, including class-specific energy-based models (Margeloiu et al., 2024), conditional GANs for mixed-type data (Xu et al., 2019), GANs with privacy-enhanced extensions (Zhao et al., 2024), and diffusion-based foundation models (Lin et al., 2024). In contrast to much of the existing work in the field of tabular data synthesis, our work does not focus on privacy-related aspects. Instead, we focus on improving downstream prediction performance, which could be enhanced using deep learning methods of tabular synthesis mentioned above.

To pre-train tabular foundation models (TFMs), several studies have used cost-efficient parametrized models to generate synthetic datasets. Hollmann et al. (2022) utilized multilayer perceptrons (MLPs), Gaussian processes (GPs), and structural causal models (SCMs), while Hollmann et al. (2025) and Qu et al. (2025) incorporated tree-based models and varied activation functions to introduce non-linearities. Unlike pre-training, which benefits from highly diverse datasets, our synthetic data generation for fine-tuning aims to generate data resembling the target dataset to enhance downstream performance.

3. Method

Below, we describe our fine-tuning pipeline and outline the data generation strategies employed by the different methods. Notably, the only varying component across the evaluated variants is the fine-tuning data generator.

Fine-tuning Pipeline. Fine-tuning is constrained to a maximum runtime of 4 hours. We employ the *AdamWScheduleFree* optimizer with a learning rate of 1×10^{-7}, batch size of 4, weight decay of 0.01, and gradient norm clipping capped at 1.0. During the iterative training loop, a batch of datasets is sampled from the data-generating process, split into context and query sets, and both the context samples and query features are forwarded to the foundation model. The loss criterion depends on the dataset's number of classes, binary cross-entropy for binary classification and cross-entropy for multiclass settings, following Hollmann et al. (2025). We perform full weight updates. To ensure consistency of the data during training and inference, we disable the heuristic preprocessing applied by the foundation models in both stages. Each iteration, we evaluate the updated model using the validation dataset and cache the model in case the model improves in performance. If fine-tuning does degrade the performance and never improves over the pre-trained model, this setup allows to then return the pre-trained model. This is important because fine-tuning with very small datasets is very dataset-dependent and performance improvements, compared to the pre-trained model, may not be achievable. The data loader utilizes 16 CPU workers, and fine-tuning is conducted on a single RTX2080 GPU. The fine-tuning process is illustrated in Figure 1, with further implementation details in Appendix A.

The data-generating methods differ in their underlying strategies for synthetic data generation. All methods are illustrated in Figure 2 and described below, with additional details on the individual variants provided in Appendix B.

Baseline: The baseline fine-tuning method uses standard minibatch updates on real data and serves as the reference point for all experimental comparisons.

TableAugmentation: Inspired by the dataset augmentation used by Ma et al. (2024a), the TableAugmentation method applies manually defined transformations on the dataset. First, we sample a random subset (50-100%) of training features (including the target). From this subset, one feature is randomly chosen as the new prediction target. This target is then discretized into classes, with the number of classes drawn uniformly between 2 and 10. Learning to predict the new target using a subset of features allows the model to view the dataset from different perspectives and to adapt to the relationships between features.

MixedModel: Our MixedModel method leverages machine learning predictors to augment the dataset in an automated way. To sample a synthetic dataset, a Bayesian Gaussian Mixture Model[2](BGM) is first fitted to the training features, and a classification model is randomly selected from a predefined pool of classifiers and trained on the real training features and labels. Synthetic features are sampled from the BGM and passed through the trained classifier to generate synthetic labels.

SCM: To generate synthetic datasets using our Structural Causal Model (SCM) approach, we first fit the SCM to the training data through a two-stage procedure. In the first stage, we infer the structure of a directed acyclic

[2]We use the `scikit-learn` implementation of the Bayesian Gaussian Mixture Model.

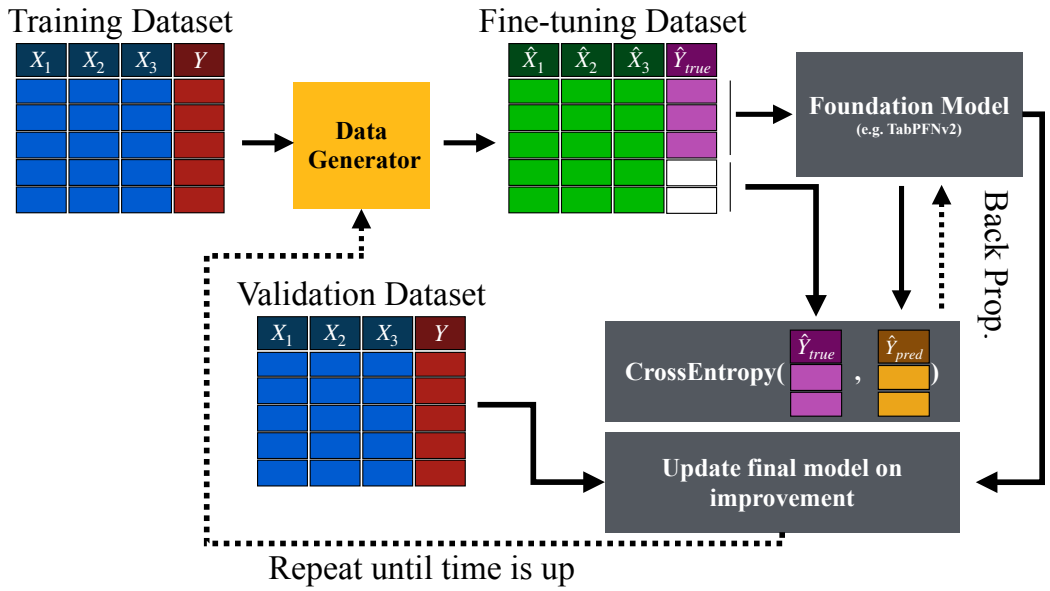


Figure 1. **Overview of the fine-tuning pipeline.** The data generator is trained on the training set. Each iteration samples a synthetic dataset, split into context and query samples. The model predicts query labels based on context and query features. Performance is evaluated on the validation set, and the final model is updated upon improvement.

Table 1. **Performance Across Different Data-Generating Methods for Fine-tuning.** Average test log-loss of TabPFNv2, with lower values indicate better performance. A dash ("–") denotes a crashed fine-tuning run. Raw represents fine-tuning with the training dataset (baseline). Green marks improvement over the pre-trained model (ICL), red highlights cases where fine-tuning degraded performance and bold indicates the best result per dataset. Values are rounded to five decimal places.

Dataset	ICL	Raw	MixedModel	SCM	TableAugmentation
blood.	0.49801	0.47887	-	**0.47134**	0.47735
churn	**0.15646**	0.27553	0.23701	0.23822	0.23407
credit-g	0.50738	0.55297	0.50228	**0.49834**	0.52894
diabetes	0.47791	0.53445	**0.47712**	0.49334	0.48838
splice	0.14111	0.54726	**0.11576**	0.11846	0.13174

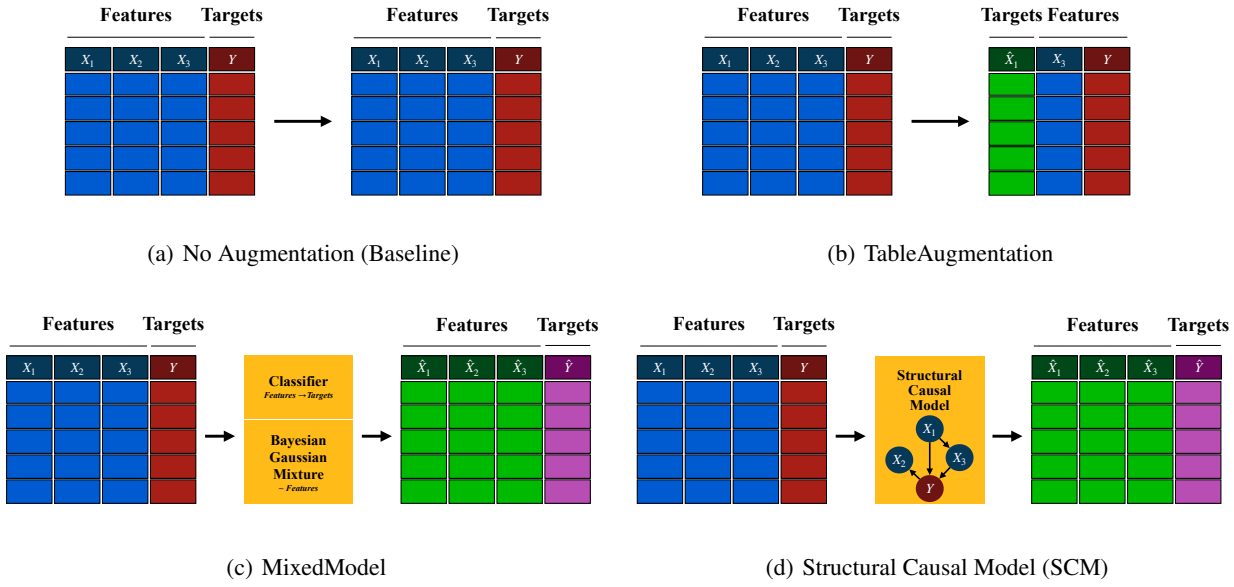
graph (DAG) that encodes the causal relationships among features. This is achieved using established causal discovery algorithms—specifically, the Peter-Clark (PC) (Spirtes et al., 2000) and Fast Causal Inference (FCI)(Spirtes et al., 2013) algorithms—applied with randomly sampled hyperparameters. The output adjacency matrices are aggregated to construct a probabilistic adjacency matrix, where edge frequencies indicate the strength of inferred relationships. A DAG is then sampled from this matrix, with cycles and bidirectional edges systematically removed to ensure acyclicity. Visualizations of an examplary probabilistic adjacency matrix are provided in Appendix 3(a).

In the second stage, we model the conditional dependencies between variables using learnable functions trained on the original data, consistent with the model class employed by Hollmann et al. (2025). The resulting SCM enables the generation of synthetic datasets that approximate the target distribution. Additional implementation details are provided in Appendix B.

4. Results

Data Setup. We evaluate our method using classification datasets from OpenML (IDs: 31, 37, 46, 1464, 40701), subsampling uniformly down to 1,000 instances if needed. Each dataset is split into five stratified folds with 60% training, 20% validation, and 20% test sets, using distinct random seeds to ensure robustness. Stratification is performed with respect to class labels to preserve the original distribution. Datasets are pre-processed using AutoGluon's automated preprocessing pipeline[3], which ordinally encodes non-numerical values, keeps missing values, and flags categorical features. The data-generating methods are constructed from a copy of the training set with additional mean

[3] AutoMLPipelineFeatureGenerator



(a) No Augmentation (Baseline)

(b) TableAugmentation

(c) MixedModel

(d) Structural Causal Model (SCM)

Figure 2. **Visual representation of the data-generating methods.** Blue and red indicate real training data; green and purple indicate synthetically generated data. MixedModel and SCM method first train internal models on real training data and then sample synthetic features from the fitted models.

imputation for numerical features and mode imputation for categorical features; z-score normalization is applied to all features except in the baseline and TableAugmentation method.

We report test set performance over five folds, using the corresponding training and validation sets as context for the test set prediction for fine-tuning of TabPFNv2, as it forms the basis for many subsequent works, while providing further analysis for TabDPT and TabICL in Appendix C. Our evaluation of different fine-tuning methods uses log-loss test performance, as it directly reflects the probabilistic calibration of the model's prediction, which aligns with the objective used during training.

Table 1 presents the average log-loss performance on the holdout test sets across multiple datasets. Values marked in green indicate an improvement compared to the pre-trained model. Notably, baseline fine-tuning using real data did not yield the best performance on any of the datasets and resulted in improved performance over the pre-trained model in only one case. This outcome is interpreted as a consequence of overfitting, despite the use of validation-based checkpointing, highlighting the inherent challenges associated with fine-tuning tabular foundation models under limited data conditions.

For the synthetic data genearting methods, the SCM and MixedModel variants achieve strong performance on most datasets, indicating that they effectively leverage classification models to generate synthetic data that captures key properties of the training distribution. In 3 out of 5 cases, these methods improved over the performance of the pre-trained model each, and **all synthetic data variants consistently outperform baseline fine-tuning across all scenarios**.

5. Conclusion

We demonstrate that competitive fine-tuning performance for tabular foundation models can be achieved using synthetic fine-tuning data. In particular, we find that conventional machine learning models can generate synthetic datasets that improve performance while preserving robustness on unseen test samples. Future work will explore a wider range of datasets and investigate how different data characteristics influence the finetunability.

Acknowledgements

L.P. acknowledges funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under SFB 1597 (SmallData), grant number 499552394; and funding by ELSA – European Lighthouse on Secure and Safe AI funded by the European Union under Grant Agreement No. 101070617. Finally, we thank the reviewers for their constructive feedback and contribution to improving the paper.

References

Breejen, F. d., Bae, S., Cha, S., and Yun, S.-Y. Fine-tuned in-context learning transformers are excellent tabular data classifiers. *arXiv preprint arXiv:2405.13396*, 2024.

den Breejen, F. and Yun, S.-Y. Attic: A new architecture for tabular in-context learning transformers, 2025. URL https://openreview.net/forum?id=DSl9sSuUhp.

den Breejen, F., Bae, S., Cha, S., Kim, T.-Y., Koh, S. H., and Yun, S.-Y. Fine-tuning the retrieval mechanism for tabular deep learning. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

Feuer, B., Schirrmeister, R., Cherepanova, V., Hegde, C., Hutter, F., Goldblum, M., Cohen, N., and White, C. Tunetables: Context optimization for scalable prior-data fitted networks. *Advances in Neural Information Processing Systems*, 37:83430–83464, 2024.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. 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.

Lin, X., Xu, C., Yang, M., and Cheng, G. Ctsyn: A foundational model for cross tabular data generation. *arXiv preprint arXiv:2406.04619*, 2024.

Liu, Q., Yang, W., Liang, C., Pang, L., and Zou, Z. Tokenize features, enhancing tables: the ft-tabpfn model for tabular classification. *arXiv preprint arXiv:2406.06891*, 2024.

Liu, S.-Y. and Ye, H.-J. Tabpfn unleashed: A scalable and effective solution to tabular classification problems. *arXiv preprint arXiv:2502.02527*, 2025.

Ma, J., Thomas, V., Hosseinzadeh, R., Kamkari, H., Labach, A., Cresswell, J. C., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. Tabdpt: Scaling tabular foundation models. *arXiv preprint arXiv:2410.18164*, 2024a.

Ma, J., Thomas, V., Yu, G., and Caterini, A. In-context data distillation with tabpfn. *arXiv preprint arXiv:2402.06971*, 2024b.

Margeloiu, A., Jiang, X., Simidjievski, N., and Jamnik, M. Tabebm: A tabular data augmentation method with distinct class-specific energy-based models. *arXiv preprint arXiv:2409.16118*, 2024.

Müller, A., Curino, C., and Ramakrishnan, R. Mothernet: A foundational hypernetwork for tabular classification. *arXiv preprint arXiv:2312.08598*, 2023.

Qu, J., Holzmüller, D., Varoquaux, G., and Morvan, M. L. Tabicl: A tabular foundation model for in-context learning on large data. *arXiv preprint arXiv:2502.05564*, 2025.

Spirtes, P., Glymour, C. N., and Scheines, R. *Causation, prediction, and search*. MIT press, 2000.

Spirtes, P. L., Meek, C., and Richardson, T. S. Causal inference in the presence of latent variables and selection bias. *arXiv preprint arXiv:1302.4983*, 2013.

Thomas, V., Ma, J., Hosseinzadeh, R., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. Retrieval & fine-tuning for in-context tabular models. *Advances in Neural Information Processing Systems*, 37:108439–108467, 2024a.

Thomas, V., Ma, J., Hosseinzadeh, R., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. Retrieval & fine-tuning for in-context tabular models. *Advances in Neural Information Processing Systems*, 37:108439–108467, 2024b.

Xu, D., Cirit, O., Asadi, R., Sun, Y., and Wang, W. Mixture of in-context prompters for tabular pfns. *arXiv preprint arXiv:2405.16156*, 2024.

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

Zhao, Z., Kunar, A., Birke, R., Van der Scheer, H., and Chen, L. Y. Ctab-gan+: Enhancing tabular data synthesis. *Frontiers in big Data*, 6:1296508, 2024.

A. fine-tuning

The context and prediction limits of all base models used—TabPFNv2, TabDPT, and TabICL—exceed 1,000 samples, enabling the full dataset to be passed as input during both fine-tuning and evaluation. Since the data-generating methods generate datasets with the same structure as the original, this compatibility is maintained throughout. During training, model performance is evaluated on the validation set using the training set as context. For evaluation and testing the sklearn compatible model wrappers, providing fit, predict and predict_proba functions are used with the fine-tuned weights, mimicing the later inference proceedure. At inference, both training and validation sets are used as context to predict on the test set.

B. Data-Generating Methods

This section provides detailed descriptions of the synthetic dataset construction processes. For efficiency, each synthetic dataset generation involves sampling four distinct datasets, each with different context and query subsets. This strategy reduces the frequency of refitting the internal models within the data-generating method.

B.1. TableAugmentation

For each sampling instance, we first select a subset of rows from the dataset. A subset of features is then uniformly sampled, with the number of retained features drawn from a uniform distribution between 50% and 100% of the total feature count. By default, the target feature is treated identically to all other features during selection. Alternative configurations allow for the target to be either always excluded or always included in the selected feature subset.

Subsequently, a new target feature is selected from among the available features, treating the original target as any other feature. Optional configurations include always reusing or never reusing the original target as the new target. The number of target classes is sampled uniformly from the range [2,10], with the effective class count constrained by the number of unique values in the selected feature. If a discrete feature is chosen as the target, the $k-1$ most frequent values are retained as individual class labels, while all remaining values are grouped as the k-th class label.

B.2. MixedModel

To generate synthetic data using the MixedModel method, we first sample a classification model, which is selected from those listed in Table 3, along with corresponding hyperparameters from a predefined distribution, including a BGM hyperparameter sample. The BGM is fitted to the training data features, while the classifier is trained on both features and labels. Once both estimators are fitted, synthetic features matching the shape of the training set are sampled from the BGM and passed through the classifier to obtain synthetic labels. As the classifier models require imputed inputs, we perform mean imputation for continuous features and mode imputation for categorical ones prior to model fitting. The hyperparamters of the BGM are shown in table 2 and the classifier models used in 3.

Table 2. Density Estimator Configuration Space of Hyperparameters for the Bayesian Gaussian Mixture Model

Model	Hyperparameter	Type	Range/Choices
BGM	`n_components`	UniformInteger	1–30
BGM	`covariance_type`	Categorical	`[full, tied, diag, spherical]`
BGM	`tol`	UniformFloat (log)	10^{-5}–10^{-1}
BGM	`reg_covar`	UniformFloat (log)	10^{-7}–10^{-4}
BGM	`max_iter`	UniformInteger	100–1000
BGM	`n_init`	UniformInteger	1–10
BGM	`init_params`	Categorical	`[kmeans, random, random_from_data, k-means++]`
BGM	`w. c. prior_type`	Categorical	`[dirichlet_process, dirichlet_distribution]`
BGM	`mean_precision_prior`	UniformFloat	0.1–10
BGM	`warm_start`	Categorical	`[True, False]`
BGM	`verbose`	Categorical	`[False]`

Table 3. Classifier Configuration Space of Hyperparameters

Model	Hyperparameter	Type	Range/Choices
RandomForestClassifier	rf_n_estimators	UniformInteger (log)	10–500
RandomForestClassifier	rf_criterion	Categorical	[gini, log_loss, entropy]
RandomForestClassifier	rf_max_depth	UniformInteger (log)	10-100
RandomForestClassifier	rf_min_samples_split	UniformInteger	2–20
RandomForestClassifier	rf_min_samples_leaf	UniformInteger	1–10
RandomForestClassifier	rf_max_leaf_nodes	UniformInteger	10–100
RandomForestClassifier	rf_bootstrap	Categorical	[True, False]
DecisionTreeClassifier	dt_criterion	Categorical	[gini, entropy, log_loss]
DecisionTreeClassifier	dt_splitter	Categorical	[best, random]
DecisionTreeClassifier	dt_max_depth	UniformInteger (log)	5–100
DecisionTreeClassifier	dt_min_samples_split	UniformInteger	2–20
DecisionTreeClassifier	dt_min_samples_leaf	UniformInteger	1–10
DecisionTreeClassifier	dt_max_features	Categorical	[0.1, 0.25, 0.5, 0.75, 1.0, sqrt, log2, None]
MLPClassifier	mlp_hidden_layer_sizes	UniformInteger	1–100
MLPClassifier	mlp_activation	Categorical	[relu, logistic, tanh]
MLPClassifier	mlp_solver	Categorical	[adam, sgd, lbfgs]
MLPClassifier	mlp_alpha	UniformFloat	0.0001–0.1
MLPClassifier	mlp_batch_size	Categorical	[auto, 32, 64, 128]
MLPClassifier	mlp_learning_rate	Categorical	[constant, invscaling, adaptive]
MLPClassifier	mlp_learning_rate_init	UniformFloat	0.0001–0.01
MLPClassifier	mlp_max_iter	UniformInteger	100–1000
MLPClassifier	mlp_momentum	UniformFloat	0.5–0.95
MLPClassifier	mlp_nesterovs_momentum	Categorical	[True, False]
MLPClassifier	mlp_early_stopping	Categorical	[True, False]
SVC	svc_kernel	Categorical	[linear, rbf, poly, sigmoid]
SVC	svc_C	UniformFloat (log)	1e-6–1e6
SVC	svc_degree	UniformInteger	1–5
SVC	svc_gamma	Categorical	[scale, auto]
SVC	svc_coef0	UniformFloat	-1–1
SVC	svc_shrinking	Categorical	[True, False]
SVC	svc_probability	Categorical	[True, False]
SVC	svc_tol	UniformFloat (log)	1e-5–1e-2
SVC	svc_cache_size	UniformFloat	200–1000
SVC	svc_class_weight	Categorical	[None, balanced]
SVC	svc_max_iter	UniformInteger	100–1000
SVC	svc_break_ties	Categorical	[True, False]
HistGradientBoostingClassifier	hgb_loss	Categorical	[log_loss]
HistGradientBoostingClassifier	hgb_learning_rate	UniformFloat	0.01–1.0
HistGradientBoostingClassifier	hgb_max_iter	UniformInteger	50–1000
HistGradientBoostingClassifier	hgb_max_leaf_nodes	UniformInteger	5–100
HistGradientBoostingClassifier	hgb_max_depth	UniformInteger	3–15
HistGradientBoostingClassifier	hgb_min_samples_leaf	UniformInteger	5–100
HistGradientBoostingClassifier	hgb_l2_regularization	UniformFloat	0.0–1.0
HistGradientBoostingClassifier	hgb_max_bins	UniformInteger	10–255

B.3. SCM

For directed acyclic graph (DAG) discovery, we employ the implementations provided by the causal-learn Python library. To introduce variability across runs, we sample the significance level α uniformly within the interval [0.01,0.1]. Each run is constrained to a maximum of 1000 samples and at most 50 features. When the number of available features exceeds this

threshold, we restrict the analysis to a randomly selected subset of 50 features, computing the adjacency matrix solely for this subset. No connections are inferred between selected and non-selected features.

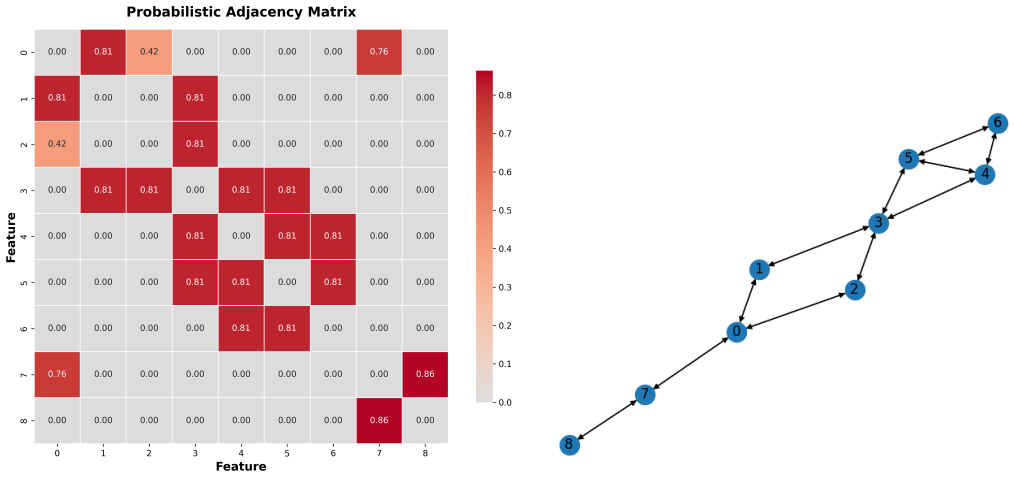
For conditional independence testing, one of the following methods is uniformly selected for each run: fisherz, chisq, or gsq. To ensure computational efficiency, each run is limited to a maximum runtime of 10 minutes, after which it is terminated and excluded from further analysis. All DAG discovery processes are parallelized across 32 CPUs to speed up computation.

While these methods are commonly referred to as "causal discovery" algorithms, their ability to uncover genuine causal relationships is contingent on strong assumptions, such as the causal markov condition and causal sufficiency. These assumptions are inherently untestable. However, in our context, their practical utility remains intact, as our objective is limited to identifying measurable associational structures that approximate the predictive distribution, rather than establishing definitive causal claims.

To construct the probabilistic adjacency matrix, we compute the empirical frequency of each directed edge across all DAG discovery runs. Specifically, the probability of an edge p_{edge} is defined as the ratio of the number of runs in which the edge appears to the total number of runs, i.e.,

$$p_{\text{edge}} = \frac{\#\text{ of runs in which edge appears}}{\#\text{ of total runs}}.$$

During each fine-tuning iteration, a single DAG is sampled from this probabilistic adjacency matrix by retaining directed edges probabilistically and subsequently eliminating bidirectional edges and cycles to ensure acyclicity. The resulting DAG is then provided to the `DoWhy`[4] framework's `StructuralCausalModel` class. Structural causal models (SCMs) are instantiated using the `assign_causal_mechanisms` function, where the `quality` parameter is randomly selected from the set {`"GOOD"`, `"Better"`, `"BEST"`}. This process fits an additive noise model to the inferred causal graph. In 3(a) we visualized the resulting probabilistic adjacency matrix from the *houses* dataset, as well as the resulting graph in 3(b). This is used to sample DAG structure and fit the SCM with.



(a) Probabilistic Adjacency (b) Resulting Graph

Figure 3. **Visual representation of the probabilistic adjacency matrix.**

C. Results TabDPT and TabICL

C.1. TabDPT

The table 4 shows that the TabDPT foundation model seems to be generally less finetunable, compared to TabPFNv2. notably the MixedModel method seams to perform best from all fine-tuning variants. This underlines the challenges of fine-tuning with limited data.

[4] DoWhy Documentation

Table 4. **Performance Across Different Data-Generating Methods for Fine-tuning.**
Average test log-loss of **TabDPT**, with **lower values indicate better performance**. A dash ("–") denotes a crashed fine-tuning run. Raw represents fine-tuning with the training dataset (baseline). Green marks improvement over the pre-trained model (ICL), red highlights cases where fine-tuning degraded performance and bold indicates the best result per dataset. Values are rounded to five decimal places.

Dataset	ICL	Raw	MixedModel	SCM	TableAugmentation
blood.	**0.47938**	0.66893	-	0.50810	0.49175
churn	**0.19633**	0.26275	0.23065	0.22606	0.23572
credit-g	0.50336	0.65663	**0.50218**	0.54437	0.50422
diabetes	**0.48536**	0.67438	0.52959	0.61478	0.49799
splice	0.28422	0.25887	**0.23713**	0.24625	0.24405

Table 5. **Performance Across Different Data-Generating Methods for Fine-tuning.**
Average test log-loss of **TabICL**, with **lower values indicate better performance**. A dash ("–") denotes a crashed fine-tuning run. Raw represents fine-tuning with the training dataset (baseline). Green marks improvement over the pre-trained model (ICL), red highlights cases where fine-tuning degraded performance and bold indicates the best result per dataset. Values are rounded to five decimal places.

Dataset	ICL	Raw	MixedModel	SCM	TableAugmentation
blood.	0.47673	0.47733	-	**0.45935**	0.47831
churn	**0.16524**	0.23588	0.23588	0.23420	0.22772
credit-g	0.49084	**0.45653**	0.47919	0.47919	**0.45653**
diabetes	**0.48057**	0.49693	0.49693	0.48454	0.48977
splice	0.18245	**0.13800**	**0.13800**	0.13966	0.14066

The fine-tuning results of TabICL, shown in 5, also shows the finetunability of the foundation model, as it seems to be very dataset and data method dependent for achieving good performance.