

# Robust Tabular Foundation Models

Matthew Peroni<sup>1,2</sup>, Franck Le<sup>2</sup>, Vadim Sheinin<sup>2</sup>

<sup>1</sup> MIT <sup>2</sup> IBM Research

mperoni1@mit.edu, fle@us.ibm.com, vadims@us.ibm.com

## Abstract

The development of tabular foundation models (TFMs) has accelerated in recent years, showing strong potential to outperform traditional ML methods for structured data. A key finding is that TFMs can be pretrained entirely on synthetic datasets, opening opportunities to design data generators that encourage desirable model properties. Prior work has mainly focused on crafting high-quality priors over generators to improve overall pretraining performance. Our insight is that parameterizing the generator distribution enables an adversarial robustness perspective: during training, we can adapt the generator to emphasize datasets that are particularly challenging for the model. We formalize this by introducing an optimality gap measure, given by the difference between TFM performance and the best achievable performance as estimated by strong baselines such as XGBoost, CatBoost, and Random Forests. Building on this idea, we propose ROBUST TABULAR FOUNDATION MODELS (RTFM), a model-agnostic adversarial training framework. Applied to the TabPFN V2 classifier, RTFM improves benchmark performance, with up to a 6% increase in mean normalized AUC over the original TabPFN and other baseline algorithms, while requiring less than 100k additional synthetic datasets. These results highlight a promising new direction for targeted adversarial training and fine-tuning of TFMs using synthetic data alone.

## Introduction

Recent survey studies have challenged the deep learning community to improve upon boosted tree methods for learning from structured data, having found a significant gap in performance remained between the paradigms [14]. In response, substantial progress has been made to close this gap, resulting in a variety of strong deep learning approaches being proposed over the past few years [24, 13, 15, 16, 5, 27, 30]. Among these new approaches, tabular foundation models (TFMs) which rely on in-context learning (ICL) [16, 21, 30] have emerged as a promising direction for classification and regression tasks with structured datasets. The benefits of this approach are two-fold. In many cases, the performance of the TFMs improves upon strong traditional baselines such as boosted trees [12, 10]. Additionally, this performance is achieved in a zero-shot framework, enabling high-quality predictions on new datasets in milliseconds

---

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

when GPU-accelerated. Further, as we will explore in this work, since these TFMs are pretrained using synthetic data, there is significant opportunity for improvement as the data generation process is expanded and improved.

Training TFMs relies on generating a large amount of diverse synthetic datasets [16]. As we will formally define in the following section, this generation process relies on constructing structural causal models (SCMs) from which datasets can be sampled [20, 23]. The structure of these SCMs is implicitly parameterized, giving significant control over the data generation process. All current publicly available, competitive TFMs [16, 21, 30] have been pretrained on datasets generated from a fixed prior distribution over these SCM parameters. However, fixed priors underrepresent certain regions of the parameter space, potentially degrading performance on real-world datasets with similar structure. This contributes to state-of-the-art TFMs still lagging behind tree-based methods on some benchmarks [19, 29]. In this work, we leverage the significant control provided by the data generation process to frame TFM training from an adversarial robustness perspective [18]. Recent work by [26] also explores a narrower version of this idea, focusing on adjusting the weights of a specific class of SCMs in GAN-style training of TFMs. In this work, we focus on a broad and highly flexible framework for adversarially robust training of TFMs. Our contributions can be summarized as: **1)** We formalize adversarial training over the SCM parameter space, allowing the model to adapt to challenging regions (e.g., varying numbers of features, categorical feature ratios, nonlinearities). We introduce an *optimality gap* concept and use it to target regions where the TFM underperforms relative to the best achievable performance. **2)** We propose an efficient, model-agnostic two-stage adversarial training algorithm for TFMs, which we call ROBUST TABULAR FOUNDATION MODELS (RTFM). **3)** We apply RTFM to TabPFN V2 [17], showing that with only 90k additional training datasets, we can significantly improve the ranking of TabPFN on several real-world tabular benchmarks.

## Problem Definition

We begin by defining some notation, and then introduce our theoretical formulation for RTFM. Let  $\Phi$  represent the space of hypothesis functions which define the relationship between features  $x$  and outcome  $y$ . The outcome  $y$  can be

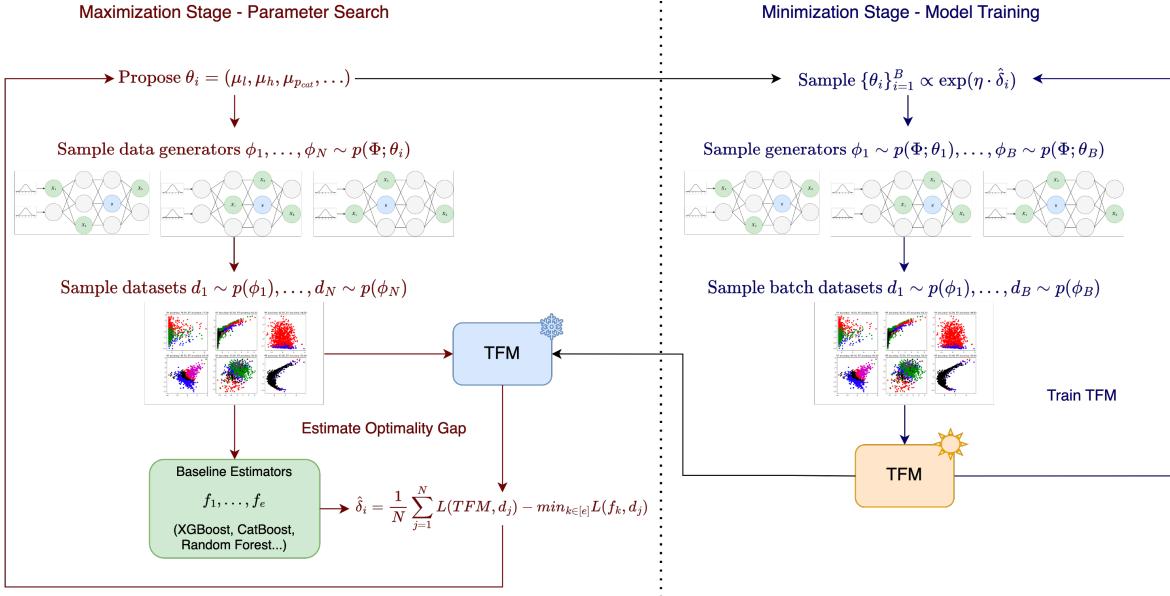


Figure 1: Overview of ROBUST TABULAR FOUNDATION MODELS (RTFM).

discrete or continuous, but for this work, we focus on classification tasks. Then a hypothesis  $\phi \in \Phi$  is a specific function which we can treat as a data generating mechanism. In the case of SCMs,  $\Phi$  is the entire space of SCMs, and  $\phi \in \Phi$  is a specific instance of an SCM, from which a dataset  $D \sim p(\phi)$  can be generated. In this work, we implement SCMs as randomly initialized multi-layer perceptrons (MLPs), as introduced in [16]. Full details of this process are provided in Appendix B. For example, we can sample the number of layers  $l \sim p(l)$ , the hidden size  $h \sim p(h)$ , the activation function  $a \sim p(a)$ , and the ratio of categorical features  $r_{cat} \sim p(r_{cat})$ , among other dimensions. An SCM is generated by first sampling from the joint distribution over these hyperparameters, each of which is parameterized. For example, we might sample the ratio of categorical features  $r_{cat} \sim TruncNorm(\mu_{r_{cat}}, a = 0, b = 1)$  from the truncated normal distribution with some mean  $\mu_{r_{cat}}$  which parametrizes the distribution. Then, assuming we have similar parameterizations for the distribution over each hyperparameter, we define  $\theta = (\mu_l, \mu_h, \mu_{r_{cat}}, \dots)$  as the parameterization of the joint distribution over all the hyperparameters. We define  $\mathcal{P}$  as the space of the parameters, such that  $\theta \in \mathcal{P}$ . The generators are then sampled from  $\phi \sim p(\Phi; \theta)$ .

We now define the original objective for the Prior-Fitted Network (PFN) learning task and provide our adversarially robust formulation. Let  $g_{\mathbf{W}}$  be our predictive model parameterized by weight matrix  $\mathbf{W}$ . As a PFN,  $g_{\mathbf{W}}$  takes as input a sequence of labeled training samples and unlabeled test samples, and predicts labels for the test samples. For a given data generator  $\phi \in \Phi$ , the PFN loss is defined as the cross entropy loss over the test samples,

$$\mathcal{L}_{PFN}(\mathbf{W}; \phi) = \quad (1)$$

$$\mathbb{E}_{\{(x_t, y_t)\} \cup D \sim p(\phi)}[-\log(g_{\mathbf{W}}(y_t | x_t, D))].$$

Since the model will be trained by sampling many data generators from the distribution  $\phi \sim p(\Phi; \theta)$ , we can write the original optimization problem as follows,

$$\min_{\mathbf{W}} \mathbb{E}_{\phi \sim p(\Phi; \theta)} [\mathcal{L}_{PFN}(\mathbf{W}; \phi)]. \quad (2)$$

Since our goal is to learn a foundation model which performs well across all data generation schemes, we want to introduce an adversary who can shift the distribution over data generators to identify the regions over which  $g_{\mathbf{W}}$  yields the worst performance. However, there is a key subtlety in this formulation. Maximizing the loss alone could lead the maximizer to select data generators which produce datasets that are hard to learn for any model. Instead, we want to maximize the *optimality gap* between the model's current performance and the best achievable performance. Since we are using the cross-entropy loss, we can frame this optimality gap in terms of the conditional entropy of the data. For any data generator  $\phi$ , let us denote the features as a random variable  $Z_x$  and the target as  $Z_y$  such that  $(Z_x, Z_y) \sim p(\phi)$ . Then the Bayes' optimal predictor is exactly  $f^* = p(Z_y | Z_x)$  which achieves the cross-entropy loss  $L_{CE}(f^*) = H_{\phi}(Z_y | Z_x)$ , the conditional entropy of the target  $Z_y$  given the features  $Z_x$ . It follows that for any fixed data generator,  $\mathcal{L}_{PFN} \geq H(Z_y | Z_x)$ , and  $\mathcal{L}_{PFN} - H(Z_y | Z_x) \geq 0$  is the optimality gap. We can then state the adversarially robust optimization problem in terms of maximizing the optimality gap,

$$\min_{\mathbf{W}} \max_{\theta \in \mathcal{P}} \mathbb{E}_{\phi \sim p(\Phi; \theta)} [\mathcal{L}_{PFN}(\mathbf{W}; \phi) - H_{\phi}(Z_y | Z_x)]. \quad (3)$$

In practice, maximizing over a single parameterization could lead to the model overfitting to a particular region of the parameter space. We therefore frame this problem from a distributionally robust optimization (DRO) perspective by constraining the actions of the maximizer [22]. Rather than a

specific parameterization  $\theta$ , we allow the maximizer to select a distribution  $Q \in \Delta_{\mathcal{P}}$  over parameters, where  $\Delta_{\mathcal{P}}$  is the simplex over  $\mathcal{P}$ . We assume  $\mathcal{P}$  is discrete, which is reasonable in practice since the parameters are either categorical or can be easily discretized (e.g.  $\mu_{r_{cat}} \in \{0, 0.1, \dots, 1\}$ ). For this work, we do not assume access to a reference distribution over  $\mathcal{P}$  from which we can constrain the divergence of  $Q$ . Instead, we will constrain the minimum entropy,  $H_{min}$ , of  $Q$  directly. This is equivalent to constraining the KL-divergence of  $Q$  when the reference distribution is uniform. For brevity, we denote the objective function as  $\delta_\theta(\mathbf{W}) = \mathbb{E}_{\phi \sim p(\Phi; \theta)} [\mathcal{L}_{PFN}(\mathbf{W}; \phi) - H_\phi(Z_y|Z_x)]$ . We can then define the DRO problem as

$$\min_{\mathbf{W}} \max_{\substack{Q \in \Delta_{\mathcal{P}} \\ H(Q) \geq H_{min}}} \mathbb{E}_{\theta \sim Q} [\delta_\theta(\mathbf{W})]. \quad (4)$$

As we prove in Appendix C, the optimal distribution for the maximizer in Equation 4 is a softmax distribution. Specifically, denoting  $Q^* = (q_1^*, \dots, q_{|\mathcal{P}|}^*)$  as the optimal distribution, for any  $i \in |\mathcal{P}|$ ,  $q_i^* \propto \exp(\eta \cdot \delta_{\theta_i}(\mathbf{W}))$ . The temperature  $\eta = \eta(H_{min}, \delta_\theta(\mathbf{W}))$  is uniquely determined by the minimum entropy and the optimality gaps. The DRO problem can then be rewritten as

$$\min_{\mathbf{W}} \sum_{i=1}^{|\mathcal{P}|} \frac{\exp(\eta \cdot \delta_{\theta_i}(\mathbf{W}))}{\sum_{i \in |\mathcal{P}|} \exp(\eta \cdot \delta_{\theta_i}(\mathbf{W}))} \cdot \delta_{\theta_i}(\mathbf{W}). \quad (5)$$

While Equation 5 is the precise theoretical optimization problem we wish to solve, in reality, estimating the conditional entropy  $H_\phi(Z_y|Z_x)$  or evaluating it analytically is often infeasible. Therefore, rather than estimate  $H_\phi(Z_y|Z_x)$  directly, in this work, we replace this term with the minimum cross-entropy loss over a collection of baseline estimators  $f_1, \dots, f_e$ , such as XGBoost, Catboost, and Random Forests [6, 9, 4], which are fit to each generated dataset. Since  $\min_{i \in [e]} \mathcal{L}_{PFN}(f_i; \phi) \geq H(Z_y|Z_x)$ , it follows that  $\mathbb{E}_{\phi \sim p(\Phi; \theta)} [\mathcal{L}_{PFN}(\mathbf{W}; \phi) - \min_{i \in [e]} \mathcal{L}_{PFN}(f_i; \phi)] \leq \delta_\theta(\mathbf{W})$ , providing us with a lower bound on the optimality gap. We then modify our original objective function, defining

$$\widehat{\delta}_\theta(\mathbf{W}) = \mathbb{E}_{\phi \sim p(\Phi; \theta)} [\mathcal{L}_{PFN}(\mathbf{W}; \phi) - \min_{i \in [e]} \mathcal{L}_{PFN}(f_i; \phi)], \quad (6)$$

and arriving at the following optimization problem,

$$\min_{\mathbf{W}} \sum_{i=1}^{|\mathcal{P}|} \frac{\exp(\eta \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}))}{\sum_{i \in |\mathcal{P}|} \exp(\eta \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}))} \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}). \quad (7)$$

Since  $\widehat{\delta}_\theta(\mathbf{W}) \leq \delta_\theta(\mathbf{W})$ , it follows that  $\widehat{\delta}_\theta(\mathbf{W}) > 0$  implies  $\delta_\theta(\mathbf{W}) > 0$ , such that for any parameter estimated to have a positive optimality gap, the model  $g_W$  does indeed have room for improvement. Further, this lower bound can always be improved by adding more baseline estimators with varying strengths and inductive biases. We aim to solve Equation 7 to recover a robust tabular foundation model. As a whole, this problem is both non-convex and non-differentiable, including a composition of multiple black-box functions. In the following section, we introduce a two-stage optimization approach to find high-quality solutions to this problem.

## ROBUST TABULAR FOUNDATION MODELS

In this section, we give an overview of our two-stage optimization algorithm, ROBUST TABULAR FOUNDATION MODELS (RTFM), which is visualized in Figure 1. The pseudocode and additional details are provided in Appendix A. We first decompose the problem naturally into a maximization stage (parameter search) and a minimization stage (model training). We assume access to a pretrained TFM,  $g_W$ , with weights  $\mathbf{W}$  which will be used as the input model for our RTFM algorithm. While not absolutely necessary, starting with a pretrained model gives us a warm-start for the model weights. In addition, we specify a fixed set of baseline models  $f_1, \dots, f_e$  which will be used to estimate the optimality gap. We begin with the maximization stage, during which we freeze  $g_W$  to maximize the optimality gap in its current state. Since the cardinality of the parameter space is large in practice, we use a black-box optimization algorithm [1, 25, 3] to efficiently search the space for parameters with large optimality gaps.

Using the black-box optimization algorithm, parameters  $\theta_i$  are proposed in sequence. For each  $\theta_i$  proposed, we estimate the optimality gap by sampling a fixed number,  $n_{ds}$ , of generators  $\phi_1, \dots, \phi_{n_{ds}} \sim p(\Phi; \theta_i)$  and datasets  $d_j \sim p(\phi_j)$  and compute  $\widehat{\delta}_{\theta_i} = \frac{1}{n_{ds}} \sum_{j=1}^{n_{ds}} \mathcal{L}_{PFN}(\mathbf{W}; d_j) - \min_{k \in [e]} \mathcal{L}_{PFN}(f_k, d_j)$ . Then for each proposed  $\theta_i$ , we must fit  $n_{ds} \cdot e$  baseline estimators. However, each pair of dataset and baseline estimator can be fit independently, such that this step can be trivially parallelized, where each baseline estimator is fit to each dataset in parallel. In practice, we found that for  $n_{ds} = 20$  and  $e = 7$ , the estimated optimality gap  $\widehat{\delta}_{\theta_i}$  could be computed in a matter of seconds when parallelized, given sufficient CPU cores (we used  $n_{cores} = n_{ds} \cdot e$ ). We repeat this process for a fixed number of trials,  $n_{trials}$ , with the underlying optimization algorithm proposing new parameters  $\theta_i$  based on the observed estimated optimality gaps. At the end of this stage, we have a collection  $\{(\theta_i, \widehat{\delta}_{\theta_i})\}_{i=1}^{n_{trials}}$  of parameters and associated estimated optimality gaps.

In the minimization stage, our goal is to train  $g_W$  on the regions of the parameter space with large optimality gaps. Following our derivation in Equation 7, we define a distribution  $Q$  over  $\mathcal{P}$  such that  $q_i \propto \exp(\eta \cdot \delta_{\theta_i}(\mathbf{W}))$ . The function  $\eta \mapsto H(Q(\eta))$  is strictly decreasing on  $(0, \infty)$ , so we can quickly find  $\eta$  given a set  $H_{min}$  via one-dimensional root finding (e.g., bisection). Using the observations from the maximization stage, our optimization problem becomes

$$\min_{\mathbf{W}} \sum_{i=1}^{n_{trials}} \frac{\exp(\eta \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}))}{\sum_{i \in [n_{trials}]} \exp(\eta \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}))} \cdot \widehat{\delta}_{\theta_i}(\mathbf{W}). \quad (8)$$

Then, to train the TFM, we generate each dataset within a batch by sampling  $\theta_i \sim Q$  and generator  $\phi \sim p(\Phi; \theta_i)$ . This sampling and dataset generation process is repeated for each batch during model training such that, in expectation, the loss is weighted according to Equation 8. After training  $g_W$  for  $n_{iter}$  iterations, we freeze the model return to the maximization stage, using the updated model weights to generate the next set of estimated optimality gaps. This max-min iteration is repeated until the objective in Equation 8 converges.

Table 1: Experiment results across two tabular dataset benchmarks. For consistency, we report normalized AUC, with values scaled to [0,1] for each dataset [19, 17]. For rank-1 wins, we do not count ties, only outright wins.

Metric	Log. Reg.	MLP	Random Forest	CatBoost	XGBoost	TabPFN <sub>n.e.</sub>	TabPFN <sub>n.e.</sub> (RTFM)
TabPertNet[28]	Mean rank AUC	5.1	4.6	4.0	3.8	4.6	<b>2.7</b>
	Mean Norm. AUC	0.4253	0.5005	0.6481	0.6663	0.5222	<b>0.8167</b>
	Rank-1 Wins	1	8	5	7	5	<b>17</b>
TabArena[10]	Mean rank AUC OVO	4.9	6.3	4.8	3.4	4.5	<b>1.9</b>
	Mean Norm. AUC OVO	0.4277	0.1801	0.5761	0.7749	0.5918	<b>0.9298</b>
	Rank-1 Wins	2	0	0	2	0	<b>12</b>

In addition to the baseline estimators  $f_1, \dots, f_e$ , after a fixed number of max-min iterations, we incorporate the TFM with its original weights as an additional baseline model. This can help mitigate any unlearning that has occurred, encouraging the model to both improve upon strong baselines as well as its own performance at the start of training.

## Experiments

We evaluate our RTFM algorithm on benchmark classification datasets from TabArena [10] and TabPertNet [28], using TabPFN V2 [17] as the starting point for our TFM. We provide a full description of the experiment setup and results in Appendix D. For the RTFM algorithm, we use  $n_{trials} = 100$  and  $n_{ds} = 20$  datasets to estimate the optimality gap for each trial. During the minimization stage, we use a learning rate of  $lr = 1e-5$ , a batch size of  $b = 64$ , and train the model for  $n_{iter} = 3000$  training steps per iteration. We run the full max-min loop for  $e = 30$  epochs. For baseline models, we use Random Forest, CatBoost, XGBoost, Logistic Regression, and MLPs. For this work, we evaluate the performance of each algorithm in Table 1 using their respective default settings, and report TFM results without ensembling. All experiments were run on a single node with one A100 GPU and 256 CPU cores over which we distributed the training and evaluation of baseline estimators during the maximization stage. Details of the parameter space are provided in Appendix B.

The results of our experiments are summarized in Table 1, and full results are provided in Appendix D. Overall, we observe a significant improvement in both AUC and mean rank when applying the RTFM algorithm compared to both the baseline TabPFN model, as well as a the collection of baseline estimators. To measure the significance of our results, we first perform the Friedman test for repeated samples on the median normalized AUC scores [8, 2]. For the TabPertNet datasets, the Friedman test yielded  $p = 2.2 \times 10^{-14}$ , and for TabArena, the test yielded  $p = 1.6 \times 10^{-12}$ . Having established significance for the benchmark performance varying by model class, we performed the Wilcoxon signed-rank test pair-wise, comparing the RTFM version of TabPFN to the other models [7]. For TabPertNet, the Wilcoxon test comparing TabPFN (RTFM) and the original TabPFN model yielded  $p = 0.0023$ , and for TabArena, the test yielded  $p = 0.0103$ . The other Wilcoxon tests all yielded smaller p-values, supporting the significance of applying RTFM to improving TabPFN against all the baseline estimators. These

results support the statistical significance of the performance improvements observed from applying the RTFM algorithm.

Across our three metrics, we observe that TabPFN (RTFM) strictly dominates the other models. Further, among datasets where the original TabPFN model is not the top ranked model ( $\text{rank} > 1$ ), we observe consistent improvement from applying the RTFM algorithm. For example, for TabPertNet, among datasets where TabPFN (original) has  $\text{rank} > 2$ , the mean rank of TabPFN (RTFM) is 3.26, which is the highest mean rank among all models and a 1.24 point improvement over the original. We also find that among 21% of these datasets where TabPFN (original) is strictly worse than the baseline models, TabPFN (RTFM) “leaps” the competition and is the top ranked model. We observe similar gains in mean rank for the TabArena datasets, with a 1.8 point improvement in mean rank on datasets where TabPFN (original) has  $\text{rank} > 1$  and TabPFN (RTFM) “leaps” the competition for 20% of these datasets. These results demonstrate the ability of the RTFM algorithm to not only improve the performance of TFMs overall, but to address specific types of datasets where the TFM lags behind traditional methods, in some cases becoming the dominant model. We also note that these results were achieved while training the TabPFN model on an additional 90k synthetic datasets, about 1% of the  $> 9$  million datasets generated to pretrain the original TabPFN model.

## Conclusion

In this work, we introduce an exciting new direction for training TFMs which is both model-agnostic and massively extensible. We demonstrate its ability to improve upon an existing state-of-the-art TFM with limited additional training across multiple tabular benchmarks. While we use TabPFN as our starting point, RTFM can be applied to any TFM, such as Mitra or TabICL. Our framework can also be easily extended to regression tasks. Further, we achieve these results while only implementing MLP-based SCMs. In future work, plan to expand the parameter space and include tree-based SCMs within our framework. The RTFM algorithm is highly-parallelizable, enabling large-scale training and efficient parameter search. We hope these promising results motivate further research in the direction of optimized, adaptive synthetic dataset generation for pretraining and fine-tuning of TFMs.

## References

- [1] Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. ArXiv:1907.10902 [cs].
- [2] Benavoli, A.; Corani, G.; and Mangili, F. 2016. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1): 152–161.
- [3] Bergstra, J.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.
- [4] Breiman, L. 2001. Random Forests. *Machine Learning*, 45(1): 5–32.
- [5] Carballo, K. V.; Na, L.; Ma, Y.; Boussioux, L.; Zeng, C.; Soenksen, L. R.; and Bertsimas, D. 2023. TabText: A Flexible and Contextual Approach to Tabular Data Representation. ArXiv:2206.10381 [cs].
- [6] Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, 785–794. New York, NY, USA: Association for Computing Machinery. ISBN 978-1-4503-4232-2.
- [7] Conover, W. J. 1999. *Practical nonparametric statistics*. john wiley & sons.
- [8] Demšar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan): 1–30.
- [9] Dorogush, A. V.; Ershov, V.; and Gulin, A. 2018. CatBoost: gradient boosting with categorical features support. ArXiv:1810.11363 [cs].
- [10] Erickson, N.; Purucker, L.; Tschalzev, A.; Holzmüller, D.; Desai, P. M.; Salinas, D.; and Hutter, F. 2025. TabArena: A Living Benchmark for Machine Learning on Tabular Data. ArXiv:2506.16791 [cs].
- [11] Frazier, P. I. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.
- [12] Friedman, J. H. 2001. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5): 1189–1232. Publisher: Institute of Mathematical Statistics.
- [13] Gorishniy, Y.; Rubachev, I.; Khrulkov, V.; and Babenko, A. 2023. Revisiting Deep Learning Models for Tabular Data. ArXiv:2106.11959 [cs].
- [14] Grinsztajn, L.; Oyallon, E.; and Varoquaux, G. 2022. Why do tree-based models still outperform deep learning on tabular data? ArXiv:2207.08815 [cs].
- [15] Hegselmann, S.; Buendia, A.; Lang, H.; Agrawal, M.; Jiang, X.; and Sontag, D. 2023. TabLLM: Few-shot Classification of Tabular Data with Large Language Models. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, 5549–5581. PMLR. ISSN: 2640-3498.
- [16] Hollmann, N.; Müller, S.; Eggensperger, K.; and Hutter, F. 2023. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. ArXiv:2207.01848 [cs].
- [17] Hollmann, N.; Müller, S.; Purucker, L.; Krishnakumar, A.; Körfer, M.; Hoo, S. B.; Schirrmeister, R. T.; and Hutter, F. 2025. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045): 319–326. Publisher: Nature Publishing Group.
- [18] Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. ArXiv:1706.06083 [stat].
- [19] McElfresh, D.; Khandagale, S.; Valverde, J.; C, V. P.; Feuer, B.; Hegde, C.; Ramakrishnan, G.; Goldblum, M.; and White, C. 2024. When Do Neural Nets Outperform Boosted Trees on Tabular Data? ArXiv:2305.02997 [cs].
- [20] Pearl, J. 2009. *Causality*. Cambridge: Cambridge University Press, 2 edition. ISBN 978-0-521-89560-6.
- [21] Qu, J.; Holzmüller, D.; Varoquaux, G.; and Morvan, M. L. 2025. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. ArXiv:2502.05564 [cs].
- [22] Rahimian, H.; and Mehrotra, S. 2019. Distributionally robust optimization: A review. *arXiv preprint arXiv:1908.05659*.
- [23] Shanmugam, R. 2018. Elements of causal inference: foundations and learning algorithms. *Journal of Statistical Computation and Simulation*, 88(16): 3248–3248.
- [24] Somepalli, G.; Goldblum, M.; Schwarzschild, A.; Bruss, C. B.; and Goldstein, T. 2021. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training. ArXiv:2106.01342 [cs].
- [25] Watanabe, S. 2023. Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. ArXiv:2304.11127 [cs].
- [26] Wu, Y.; and Bergman, D. L. 2025. Zero-shot Meta-learning for Tabular Prediction Tasks with Adversarially Pre-trained Transformer. ArXiv:2502.04573 [cs].
- [27] Yan, J.; Zheng, B.; Xu, H.; Zhu, Y.; Chen, D. Z.; Sun, J.; Wu, J.; and Chen, J. 2024. Making Pre-trained Language Models Great on Tabular Prediction. ArXiv:2403.01841 [cs].
- [28] Ye, C.; Lu, G.; Wang, H.; Li, L.; Wu, S.; Chen, G.; and Zhao, J. 2024. Towards Cross-Table Masked Pretraining for Web Data Mining. ArXiv:2307.04308 [cs].
- [29] Ye, H.-J.; Liu, S.-Y.; and Chao, W.-L. 2025. A Closer Look at TabPFN v2: Understanding Its Strengths and Extending Its Capabilities. ArXiv:2502.17361 [cs].
- [30] Zhang, X.; and Robinson, D. M. 2025. Mitra: Mixed synthetic priors for enhancing tabular foundation models.

## A RTFM Algorithm

We summarize ROBUST TABULAR FOUNDATION MODELS in Algorithms 1 and 2. A description of the algorithm is given in Section . In Algorithm 2, we reference an abstract function `SuggestParameters` which represents any black-box optimization algorithm. In this case, we use the Optuna python package [1] and the Tree-structured Parzen Estimator [25] algorithm, which belongs to the class of Bayesian Optimization algorithms [11]. At each iteration, this algorithm uses the information gained from the previous trials (the estimated optimality gaps) to propose a new parameter, balancing exploration and exploitation to efficiently find parameters corresponding to large optimality gaps. Since we will sample the parameters following a softmax distribution, identifying the parameters corresponding to the largest optimality gaps is paramount, as the probability of sampling from the rest of the parameter space will be negligible. To begin the minimization stage, we first construct the distribution  $Q \in \Delta_{\mathcal{P}}$ . Rather than constructing  $H_{min}$  as a parameter itself, we introduce a scalar parameter  $c \in (0, 1)$  and define  $H_{min} = c \log(n_{trials})$ , such that  $H_{min}$  is a given as a fraction of the maximum possible entropy of  $Q$ . With  $H_{min}$  specified, we can find  $\eta$  such that  $H(Q(\eta)) = H_{min}$  using one-dimensional root finding (e.g. bisection). Then we generate each dataset in a training batch by sampling a parameter  $\theta_i \sim Q$  and generator  $\phi \sim p(\Phi; \theta_i)$ . We train the model for a fixed number of iterations and then return to the maximization stage. In our implementation, after five epochs we incorporate the original TabPFN model as another baseline model. This can potentially address regions of the parameter space where the model has reduced its performance by focusing on improving in other regions. An illustration of the maximum estimated optimality gap found during each maximization stage is shown in Figure 2. We observe that after the original TFM is introduced as an additional baseline model, the parameter search uncovers larger optimality gaps for a few epochs, suggesting that the model did have some performance degradation during the initial training epochs over some regions of the parameter space.

The RTFM algorithm is highly parallelizable, both in the dataset generation steps and the parameter search. For a fixed parameterization  $\theta$ , each dataset generator  $\phi \sim p(\Phi; \theta)$  is sampled independently from the same distribution, such that all datasets generated for a given batch can be created concurrently. The `ParameterSearch` algorithm can be completely parallelized within each trial. In our implementation, for each trial, we generate a batch of datasets and then fit and evaluate each model and dataset pair concurrently. For example, in our experiments, we set  $n_d = 20$  and used  $e = 7$  baseline models, resulting in  $n_{ds} * e = 140$  separate processes distributed across all available cores.

## B SCM Parameters

We summarize the parameter space used in our experiments in Table 2. We show the parameters corresponding to the maximum optimality gap found over the  $n_{trials}$  per max-min iteration in Table 5. For this work, we implemented SCMs as randomized MLPs. In addition to the structure

---

Algorithm 1: ROBUST TABULAR FOUNDATION MODELS (RTFM)

---

**Require:** Initial TFM weights  $\mathbf{W}$ , baseline estimators  $\{f_1, \dots, f_e\}, n_e, n_{iter}, n_t, n_d, c \in (0, 1)$   
 $\{(\theta_i, \delta_i)\}_{i=1}^{n_{trials}} \leftarrow \text{ParameterSearch}(\mathbf{W}, \{f_1, \dots, f_e\}, n_t, n_d)$   
 $H_{min} \leftarrow c \log(n_t)$   
 $\eta \leftarrow \text{RootFinding}(H_{min}, \delta)$   
 $Q \leftarrow \text{Softmax}(\eta \cdot \delta)$   
**for**  $e \in [n_e]$  **do**  
    **for**  $t \in [n_{iter}]$  **do**  
         $\theta_1, \dots, \theta_{n_b} \sim Q$   
         $\phi_1, \dots, \phi_{n_b} \sim p(\Phi; \theta_1), \dots, p(\Phi; \theta_{n_b})$   
         $\{(x^i, y^i)\}_{i=1}^{n_b} \sim p(\phi_1), \dots, p(\phi_{n_b})$   
         $\{x_{train}^i, x_{test}^i, y_{train}^i, y_{test}^i\}_{i=1}^{n_b} \leftarrow \text{TrainTestSplit}(\{(x^i, y^i)\}_{i=1}^{n_b})$   
         $l(gw) \leftarrow \mathcal{L}_{PFN}(gw, \{x_{train}^i, x_{test}^i, y_{train}^i, y_{test}^i\}_{i=1}^{n_b})$   
         $\mathbf{W} \leftarrow \text{update}(\mathbf{W}, l(gw))$   $\triangleright$  Gradient step  
    **end for**  
     $\{(\theta_i, \delta_i)\}_{i=1}^{n_{trials}} \leftarrow \text{ParameterSearch}(\mathbf{W}, \{f_1, \dots, f_e\}, n_t, n_d)$   
     $\eta \leftarrow \text{RootFinding}(H_{min}, \delta)$   
     $Q \leftarrow \text{Softmax}(\eta \cdot \delta)$   
**end for**  
**Return** Robust TFM model  $gw$

---

Algorithm 2: ParameterSearch

---

**Require:** TFM weights  $\mathbf{W}$ , baseline estimators  $\{f_1, \dots, f_e\}$ , num. trials  $n_t$ , num. datasets  $n_d$   
 $D_\theta = \{\}$   
**for**  $i \in [n_t]$  **do**  
     $\theta_i \leftarrow \text{SuggestParameters}(D_\theta)$   $\triangleright$  Black-box optimizer proposes new parameter  
    **for**  $j \in [n_d]$  **do**  
         $\phi_j \sim p(\Phi; \theta_i)$   
         $x_{train}^j, x_{test}^j, y_{train}^j, y_{test}^j \sim p(\phi_j)$   
         $l_j(gw) \leftarrow \mathcal{L}_{PFN}(gw, x_{train}^j, y_{train}^j, x_{test}^j, y_{test}^j)$   
         $l_b \leftarrow \infty$   
        **for**  $f_k \in \{f_1, \dots, f_e\}$  **do**  
             $l_j(f_k) \leftarrow \mathcal{L}_{PFN}(f_k, x_{train}^j, y_{train}^j, x_{test}^j, y_{test}^j)$   
             $l_b \leftarrow \min(l_b, l_j(f_k))$   
        **end for**  
         $\delta_i^j \leftarrow l_j(gw) - l_b$   $\triangleright$  Estimate optimality gap  
    **end for**  
     $\delta_i \leftarrow \frac{1}{n_d} \sum_{j=1}^{n_d} \delta_i^j$   
     $D_\theta \leftarrow D_\theta \cup \{(\theta_i, \delta_i)\}$   
**end for**  
**Return** Parameters and estimated optimality gaps  $D_\theta$

---

of the MLP itself, we can control more general properties such as the number of features (node activations) that are included in the dataset, and the ratio of features which are categorical. An illustration of this implementation is shown in Figure 3 . For integer-valued parameters, we sample from the truncated normal distribution over the range specified in the table, and then round to the nearest integer. For

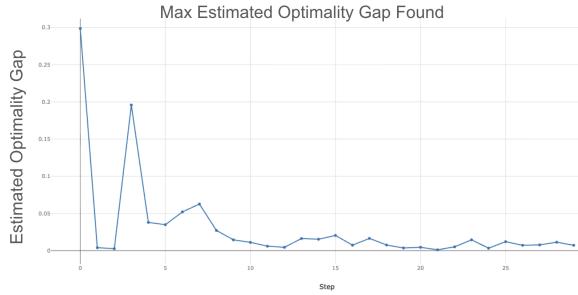


Figure 2: Maximum estimated optimality gap after each parameter search during model training. After epoch 5, the original TFM is introduced as an additional baseline model.

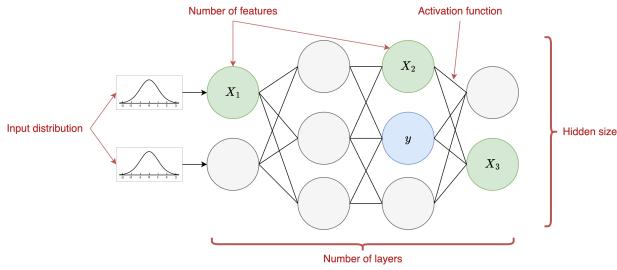


Figure 3: Anatomy of an MLP-based SCM.

categorical variables, we use an  $\epsilon$ -greedy sampling scheme. For example, we select the activation function  $\mu_a$  with probability  $1 - \epsilon$  and with probability  $\epsilon$  select another activation function at random. For missingness, we mask the training data completely at random. During the maximization stage, when we perform the parameter search, we make a few simplifications to reduce the size of the search space. We discretize the values for all numeric parameters. Rather than varying  $\mu_h, \mu_l, \mu_{x_{in}}$  independently, we couple them and define five different MLP sizes  $(\mu_h, \mu_l, \mu_{x_{in}}) = [(5, 3, 2), (10, 5, 3), (32, 8, 3), (64, 10, 8), (128, 12, 13)]$  to search over. The parameters for our experiments is then  $\theta = (\mu_h, \mu_l, \mu_{x_{in}}, \mu_a, \mu_{z_x}, \mu_c, \mu_{r_{cat}}, \mu_{r_s}, \mu_m, \mu_d)$ , and each parameter is sampled independently. Within our framework, we can easily add additional parameters, including those that control the variance or completely separate dimensions such as the type of SCM (MLP or tree-based).

## C Proof of Optimal Maximizer Distribution

In this section, we provide the proof that the optimal distribution for the maximizer in equation 4 is a softmax distribution such that  $q_i^* \propto \exp(\eta \cdot \delta_{\theta_i}(\mathbf{W}))$ .

**Proposition 1.** Let  $\delta = (\delta_{\theta_1}, \dots, \delta_{\theta_n}) \in \mathbb{R}^n$  and consider

$$\max_{Q \in \Delta_n} \sum_{i=1}^n q_i \delta_{\theta_i} \quad \text{s.t.} \quad H(Q) \geq H_{\min},$$

where  $Q = (q_1, \dots, q_n)$ ,  $\Delta_n = \{Q \in \mathbb{R}^n : q_i \geq 0, \sum_i q_i = 1\}$  and  $H(Q) = -\sum_i q_i \log q_i$ . If  $0 < H_{\min} < \log n$  and  $\exists i, j$  such that  $\delta_{\theta_i} \neq \delta_{\theta_j}$ , the optimizer  $q^*$  is

strictly positive and there exists a unique  $\lambda > 0$  such that

$$q_i^* = \frac{\exp(\delta_{\theta_i}/\lambda)}{\sum_{j=1}^n \exp(\delta_{\theta_j}/\lambda)}, \quad i = 1, \dots, n,$$

where  $\lambda$  is determined implicitly by the entropy constraint  $H(Q^*) = H_{\min}$ .

*Proof.* Defining  $F(Q) = -\sum_i q_i \delta_{\theta_i}$ , we rewrite the problem as

$$\min_{Q \in \Delta_n} F(Q) \quad \text{s.t.} \quad g(Q) := -H(Q) + H_{\min} \leq 0.$$

Since  $F$  and  $g$  are convex and the feasible region has interior points (e.g., the uniform distribution), Slater's condition holds, so KKT conditions are necessary and sufficient.

The Lagrangian is given by

$$\begin{aligned} \mathcal{L}(Q, \lambda, \mu, \gamma) = & -\sum_i q_i \delta_{\theta_i} + \lambda \left( \sum_i q_i \log q_i + H_{\min} \right) \\ & + \mu \left( \sum_i q_i - 1 \right) - \sum_i \gamma_i q_i, \end{aligned}$$

where  $\lambda \geq 0$ ,  $\mu \in \mathbb{R}$ , and  $\gamma_i \geq 0$ . Stationarity gives

$$-\delta_{\theta_i} + \lambda(1 + \log q_i) + \mu - \gamma_i = 0.$$

When the entropy constraint is active ( $\lambda > 0$ ) and  $q_i > 0$ , complementary slackness implies  $\gamma_i = 0$ , yielding

$$\lambda(1 + \log q_i) = \delta_{\theta_i} - \mu.$$

Exponentiating and enforcing  $\sum_i q_i = 1$  gives

$$q_i(\lambda) = \frac{\exp(\delta_{\theta_i}/\lambda)}{\sum_j \exp(\delta_{\theta_j}/\lambda)}.$$

Because  $\lambda > 0$ , each  $q_i > 0$  and all  $\gamma_i = 0$ , so the solution is interior. We now prove the uniqueness of  $\lambda$  solving  $H(Q(\lambda)) = H_{\min}$ . Let  $q_i(\lambda) = \frac{\exp(\delta_{\theta_i}/\lambda)}{Z(\lambda)}$ ,  $Z(\lambda) = \sum_j \exp(\delta_{\theta_j}/\lambda)$ , and write  $E(\lambda) = \mathbb{E}_{Q(\lambda)}[\delta] = \sum_i q_i(\lambda) \delta_{\theta_i}$ ,  $E_2(\lambda) = \mathbb{E}_{Q(\lambda)}[\delta^2] = \sum_i q_i(\lambda) \delta_{\theta_i}^2$ . The entropy of  $q(\lambda)$  can be written

$$H(q(\lambda)) = \log Z(\lambda) - \frac{E(\lambda)}{\lambda}.$$

Differentiate  $\log Z$  with respect to  $\lambda$ , we have

$$\frac{d}{d\lambda} \log Z(\lambda) = \frac{1}{Z(\lambda)} \sum_i \exp(\delta_{\theta_i}/\lambda) \cdot (-\delta_{\theta_i}/\lambda^2) = -\frac{E(\lambda)}{\lambda^2}.$$

Next compute  $dE/d\lambda$ . From  $\log q_i(\lambda) = \delta_{\theta_i}/\lambda - \log Z(\lambda)$  we get

$$\frac{dq_i}{d\lambda} = q_i \frac{d}{d\lambda} \log q_i = q_i \left( -\frac{\delta_{\theta_i}}{\lambda^2} - \frac{d}{d\lambda} \log Z(\lambda) \right).$$

Table 2: SCM Parameter space and sampling distributions.

Parameter	Type	Range	Distribution
Mean Hidden Size $\mu_h$	integer	[3, 256]	TruncNorm( $\mu_h$ , 10)
Mean Num. Layers $\mu_l$	integer	[1, 12]	TruncNorm( $\mu_l$ , 1)
Mean Num. Inputs $\mu_{x_{in}}$	integer	[1, 15]	TruncNorm( $\mu_{x_{in}}$ , 1)
Activation fn $\mu_a$	categorical	[“relu”, “elu”, “identity”, “tanh”]	$\epsilon$ -greedy ( $\epsilon = 0.3$ )
Mean Num. Features $\mu_{z_x}$	integer	[2, 200]	TruncNorm( $\mu_{z_x}$ , $\mu_{z_x}/4$ )
Mean Num. Classes $\mu_c$	integer	[2, 10]	TruncNorm( $\mu_m$ , 1)
Mean categorical ratio $\mu_{r_{cat}}$	continuous	[0, 1]	TruncNorm( $\mu_{r_{cat}}$ , 0.1)
Mean ordered cat. ratio $\mu_{r_s}$	continuous	[0, 1]	TruncNorm( $\mu_{r_s}$ , 0.1)
Mean ratio missing $\mu_m$	continuous	[0, 1]	TruncNorm( $\mu_m$ , 0.1)
Input distribution $\mu_d$	categorical	[“exponential”, “uniform”, “normal”]	$\epsilon$ -greedy ( $\epsilon = 0.3$ )

Using this, we find

$$\begin{aligned} \frac{dE}{d\lambda} &= \sum_i \delta_{\theta_i} \frac{dq_i}{d\lambda} \\ &= -\frac{1}{\lambda^2} \sum_i q_i \delta_{\theta_i}^2 - \frac{d}{d\lambda} \log Z(\lambda) \sum_i q_i \delta_{\theta_i} \\ &= -\frac{E_2(\lambda)}{\lambda^2} - \left( -\frac{E(\lambda)}{\lambda^2} \right) E(\lambda) \\ &= -\frac{E_2(\lambda) - E(\lambda)^2}{\lambda^2}. \end{aligned}$$

Therefore

$$\frac{dE}{d\lambda} = -\frac{\text{Var}_{Q(\lambda)}(\delta)}{\lambda^2}.$$

Finally, differentiate  $H(Q(\lambda)) = \log Z(\lambda) - E(\lambda)/\lambda$ :

$$\frac{d}{d\lambda} H(Q(\lambda)) = \frac{d}{d\lambda} \log Z(\lambda) - \frac{1}{\lambda} \frac{dE}{d\lambda} + \frac{E(\lambda)}{\lambda^2}.$$

Substituting  $\frac{d}{d\lambda} \log Z(\lambda) = -\frac{E(\lambda)}{\lambda^2}$  cancels the last term, leaving

$$\frac{d}{d\lambda} H(Q(\lambda)) = -\frac{1}{\lambda} \frac{dE}{d\lambda} = \frac{\text{Var}_{Q(\lambda)}(\delta)}{\lambda^3}.$$

Since  $\text{Var}_{Q(\lambda)}(\delta) \geq 0$  and  $\lambda > 0$ , the derivative is nonnegative and is strictly positive unless all  $\delta_{\theta_i}$  are equal. Hence  $\lambda \mapsto H(Q(\lambda))$  is strictly increasing on  $(0, \infty)$ , which implies the uniqueness of  $\lambda$  solving  $H(q(\lambda)) = H_{\min}$ . By KKT sufficiency,  $Q^* = Q(\lambda)$  is the global optimum.  $\square$

## D Full Experiment Results

In Tables 3 and 4 we provide the full experiment results from the TabPertNet [28] and TabArena [10] tabular benchmarks. For both benchmarks, we evaluate on classification datasets only, and limit our evaluation to datasets with at most  $n = 10k$  samples and  $p = 500$  features. We hold out some datasets from both benchmarks as validation datasets during model training. We report our test results on  $N = 74$  datasets from TabPertNet and  $N = 21$  datasets from TabArena. For TabArena, we use the same folds and repeats as defined in the original work and corresponding leaderboard. For TabPertNet, we use the same train/test split used in the original paper. We report AUC for TabPertNet and AUC one-vs-one (OVO) for TabArena.

Table 3: TabPertNet [28] Test Results (AUC ↑).

<b>Dataset</b>	<b>TabPFN<sub>n.e.</sub> (RTFM)</b>	<b>TabPFN<sub>n.e.</sub> (Base)</b>	<b>CatBoost</b>	<b>Random Forest</b>	<b>XGBoost</b>	<b>Log. Reg.</b>	<b>MLP</b>
Breast_Cancer	0.886	0.885	0.830	0.820	0.773	0.852	0.816
1736_combined-wine-data	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0446_newton_hema	0.976	0.952	0.976	0.976	1.000	1.000	0.952
1564_Mammographic-Mass-Data-Set	0.938	0.934	0.843	0.919	0.857	0.916	0.892
0408_pharynx	0.911	0.800	0.778	0.844	0.867	0.867	0.889
1600_VulNoneVul	0.594	0.554	0.693	0.546	0.640	0.637	0.549
Customer_Behaviour	0.991	0.994	0.734	0.818	0.837	0.855	0.852
1333_ricci_vs_destefano	1.000	1.000	0.708	0.917	1.000	0.875	0.875
1635_Is-this-a-good-customer	0.789	0.796	0.750	0.767	0.788	0.693	0.639
1408_national-longitudinal-survey-binary	1.000	1.000	0.991	0.988	0.962	0.995	0.998
b_depressed	0.551	0.541	0.513	0.567	0.606	0.446	0.614
0356_delta_elevators	0.951	0.953	0.950	0.955	0.947	0.945	0.945
0437_quake	0.518	0.504	0.496	0.507	0.480	0.603	0.606
1461_heart-failure	0.856	0.867	0.856	0.833	0.839	0.944	0.867
0124_analcatdata_impeach	1.000	1.000	0.550	0.750	0.850	0.900	1.000
2304_electricity	0.891	0.890	0.870	0.856	0.869	0.699	0.790
0446_arsenic-female-bladder	0.811	0.819	0.776	0.825	0.774	0.825	0.887
0948_Ishwar	0.997	0.999	0.961	0.966	0.968	0.961	0.973
2393_airlines	0.650	0.661	0.572	0.645	0.592	0.589	0.567
0400_analcatdata_supreme	1.000	1.000	1.000	1.000	1.000	0.996	0.999
2619_sf-police-incidents	0.584	0.515	0.572	0.449	0.407	0.454	0.477
2308_electricity	0.920	0.913	0.893	0.885	0.895	0.732	0.842
0885_compas-two-years	0.744	0.746	0.692	0.735	0.659	0.729	0.733
1201_Gender-Recognition-by-Voice	1.000	0.999	0.998	0.999	0.999	0.995	0.999
0546_analcatdata_seropositive	1.000	1.000	0.833	0.861	1.000	0.972	0.944
0345_delta_ailerons	0.977	0.976	0.972	0.973	0.976	0.967	0.971
0445_arсенic-male-bladder	0.983	0.991	0.784	0.991	0.957	0.983	0.595
2389_airlines	0.678	0.699	0.663	0.646	0.637	0.656	0.588
0424_autoPrice	1.000	1.000	0.958	1.000	1.000	0.938	0.938
0419_pm10	0.628	0.591	0.565	0.637	0.675	0.404	0.537
2703_compas-two-years	0.763	0.761	0.731	0.756	0.704	0.752	0.749
1774_Early-Stage-Diabetes	1.000	1.000	1.000	1.000	1.000	0.997	0.999
1006_Titanic	0.742	0.742	0.742	0.742	0.742	0.662	0.742
1752_Wisconsin-breast-cancer	1.000	1.000	0.997	0.999	0.999	1.000	1.000
0541_plasma_retinol	0.688	0.693	0.315	0.562	0.594	0.506	0.574
1011_cleve	0.913	0.909	0.856	0.928	0.923	0.918	0.952
0472_analcatdata_marketing	0.714	0.663	0.566	0.480	0.437	0.383	0.714
0447_arсенic-female-lung	0.948	0.888	0.784	0.888	0.970	0.276	0.293
2391_airlines	0.616	0.628	0.665	0.661	0.629	0.608	0.558
2392_airlines	0.612	0.601	0.665	0.627	0.647	0.691	0.696
0555_socmob	0.996	0.998	0.951	0.957	0.960	0.932	0.959
2622_sf-police-incidents	0.502	0.456	0.398	0.474	0.561	0.504	0.469
0406_visualizing_environmental	1.000	1.000	1.000	1.000	1.000	1.000	1.000
diabetes_data_upload	1.000	0.998	1.000	1.000	1.000	0.989	0.992
0312_cpu_act	0.981	0.981	0.976	0.977	0.971	0.956	0.979
1458_kdd_ipums_la_97-small	0.942	0.942	0.932	0.943	0.930	0.881	0.540
piracydataset	0.678	0.604	0.684	0.737	0.656	0.770	0.778
2620_sf-police-incidents	0.531	0.492	0.367	0.541	0.425	0.491	0.517
2306_electricity	0.879	0.882	0.870	0.862	0.863	0.771	0.838
2621_sf-police-incidents	0.611	0.579	0.634	0.350	0.628	0.630	0.516
2390_airlines	0.600	0.597	0.590	0.574	0.560	0.584	0.516
Bank_Personal_Loan_Modelling	0.589	0.582	0.605	0.594	0.606	0.604	0.562
1512_eye_movements	0.652	0.650	0.701	0.686	0.700	0.594	0.660
NFL	0.632	0.595	1.000	1.000	1.000	1.000	0.977
2305_electricity	0.928	0.931	0.920	0.903	0.911	0.788	0.858
1592_Diabetes-Data-Set	0.882	0.882	0.871	0.884	0.874	0.839	0.844
0509_pollen	0.472	0.467	0.451	0.481	0.485	0.478	0.502
1413_shill-bidding	1.000	1.000	1.000	1.000	0.999	0.999	0.996
BankNoteAuthentication	1.000	1.000	1.000	1.000	1.000	0.999	1.000
0284_bank8FM	0.992	0.991	0.989	0.991	0.988	0.991	0.991
0292_cpu_small	0.980	0.979	0.978	0.979	0.974	0.949	0.972
1742_Loan-Prediction	0.729	0.698	0.669	0.671	0.703	0.692	0.657
Employee Satisfaction Index	0.574	0.538	0.463	0.484	0.522	0.468	0.429
1759_Red-White-wine-Dataset	1.000	1.000	1.000	1.000	1.000	1.000	1.000
0526_colleges_aaup	1.000	1.000	0.918	0.913	0.939	0.958	0.960
bt_dataset_t3	1.000	1.000	1.000	1.000	1.000	0.970	0.997
0435_strikes	1.000	1.000	1.000	0.995	0.991	0.616	0.903
UniversalBank	0.589	0.582	0.605	0.594	0.609	0.604	0.579
1692_Gender-Classification-Dataset	0.996	0.997	0.994	0.996	0.996	0.996	0.996
1142_Sick_numeric	0.999	0.998	0.999	0.999	0.997	0.949	0.943
1451_early-stage-diabetes	1.000	1.000	1.000	1.000	1.000	0.997	1.000
TravelInsurancePrediction	0.823	0.811	0.812	0.802	0.812	0.786	0.815
1898_Personal-Loan-Modeling	0.590	0.611	0.633	0.630	0.633	0.627	0.601
new_model	1.000	1.000	1.000	1.000	1.000	1.000	1.000
1578_kdd_ipums_la_97-small	0.957	0.952	0.947	0.958	0.952	0.906	0.534
loan_train	0.731	0.672	0.657	0.682	0.695	0.671	0.645

Table 4: TabArena [10] Test Results (AUC OVO  $\uparrow$ ).

Dataset	TabPFN <sub>n.e.</sub> (RTFM)	TabPFN <sub>n.e.</sub> (Base)	CatBoost	Random Forest	XGBoost	Log. Reg.	MLP
Bank_Customer_Churn	0.861 $\pm$ 0.007	0.870 $\pm$ 0.007	0.843 $\pm$ 0.008	0.848 $\pm$ 0.007	0.831 $\pm$ 0.007	0.838 $\pm$ 0.008	0.798 $\pm$ 0.009
Fitness_Club	0.821 $\pm$ 0.012	0.821 $\pm$ 0.012	0.748 $\pm$ 0.020	0.745 $\pm$ 0.020	0.718 $\pm$ 0.021	0.780 $\pm$ 0.018	0.701 $\pm$ 0.022
Is-this-a-good-customer	0.743 $\pm$ 0.018	0.738 $\pm$ 0.023	0.677 $\pm$ 0.025	0.725 $\pm$ 0.019	0.679 $\pm$ 0.021	0.677 $\pm$ 0.024	0.618 $\pm$ 0.029
MIC	0.670 $\pm$ 0.011	0.660 $\pm$ 0.012	0.674 $\pm$ 0.019	0.671 $\pm$ 0.020	0.667 $\pm$ 0.022	0.700 $\pm$ 0.025	0.659 $\pm$ 0.020
Marketing_Campaign	0.902 $\pm$ 0.016	0.900 $\pm$ 0.017	0.834 $\pm$ 0.017	0.837 $\pm$ 0.021	0.819 $\pm$ 0.020	0.848 $\pm$ 0.019	0.810 $\pm$ 0.018
NATICUSdroid	0.984 $\pm$ 0.002	0.983 $\pm$ 0.002	0.984 $\pm$ 0.002	0.985 $\pm$ 0.002	0.976 $\pm$ 0.003	0.981 $\pm$ 0.002	0.978 $\pm$ 0.003
anneal	0.994 $\pm$ 0.010	0.999 $\pm$ 0.003	0.983 $\pm$ 0.023	0.993 $\pm$ 0.011	0.995 $\pm$ 0.011	0.995 $\pm$ 0.007	0.998 $\pm$ 0.003
blood-transfusion-service-center	0.749 $\pm$ 0.029	0.748 $\pm$ 0.027	0.673 $\pm$ 0.033	0.738 $\pm$ 0.029	0.681 $\pm$ 0.026	0.752 $\pm$ 0.027	0.706 $\pm$ 0.070
churn	0.926 $\pm$ 0.012	0.926 $\pm$ 0.010	0.920 $\pm$ 0.009	0.923 $\pm$ 0.012	0.916 $\pm$ 0.008	0.820 $\pm$ 0.013	0.828 $\pm$ 0.012
coil2000_insurance_policies	0.766 $\pm$ 0.012	0.757 $\pm$ 0.013	0.711 $\pm$ 0.013	0.748 $\pm$ 0.012	0.691 $\pm$ 0.014	0.714 $\pm$ 0.011	0.682 $\pm$ 0.012
credit-g	0.779 $\pm$ 0.020	0.766 $\pm$ 0.017	0.755 $\pm$ 0.023	0.770 $\pm$ 0.020	0.771 $\pm$ 0.020	0.761 $\pm$ 0.023	0.723 $\pm$ 0.027
diabetes	0.838 $\pm$ 0.023	0.840 $\pm$ 0.024	0.796 $\pm$ 0.020	0.833 $\pm$ 0.024	0.820 $\pm$ 0.023	0.829 $\pm$ 0.024	0.721 $\pm$ 0.035
hazelnut-spread-contaminant-detection	0.986 $\pm$ 0.003	0.986 $\pm$ 0.004	0.974 $\pm$ 0.005	0.967 $\pm$ 0.005	0.957 $\pm$ 0.006	0.608 $\pm$ 0.047	0.849 $\pm$ 0.014
maternal_health_risk	0.945 $\pm$ 0.013	0.941 $\pm$ 0.012	0.939 $\pm$ 0.014	0.935 $\pm$ 0.014	0.941 $\pm$ 0.014	0.800 $\pm$ 0.018	0.777 $\pm$ 0.024
polish_companies_bankruptcy	0.956 $\pm$ 0.006	0.955 $\pm$ 0.007	0.952 $\pm$ 0.007	0.958 $\pm$ 0.010	0.933 $\pm$ 0.006	0.552 $\pm$ 0.041	0.650 $\pm$ 0.087
qsar-biodeg	0.938 $\pm$ 0.012	0.934 $\pm$ 0.012	0.923 $\pm$ 0.013	0.929 $\pm$ 0.012	0.928 $\pm$ 0.012	0.925 $\pm$ 0.013	0.917 $\pm$ 0.013
seismic-bumps	0.771 $\pm$ 0.023	0.768 $\pm$ 0.029	0.661 $\pm$ 0.024	0.746 $\pm$ 0.016	0.676 $\pm$ 0.016	0.713 $\pm$ 0.026	0.639 $\pm$ 0.029
splice	0.995 $\pm$ 0.002	0.995 $\pm$ 0.002	0.994 $\pm$ 0.002	0.994 $\pm$ 0.002	0.994 $\pm$ 0.002	0.989 $\pm$ 0.002	0.991 $\pm$ 0.002
students_dropout_and_academic_success	0.883 $\pm$ 0.007	0.880 $\pm$ 0.005	0.866 $\pm$ 0.006	0.859 $\pm$ 0.007	0.864 $\pm$ 0.009	0.844 $\pm$ 0.005	0.812 $\pm$ 0.007
taiwanese_bankruptcy_prediction	0.944 $\pm$ 0.007	0.943 $\pm$ 0.005	0.932 $\pm$ 0.010	0.937 $\pm$ 0.007	0.923 $\pm$ 0.012	0.576 $\pm$ 0.027	0.521 $\pm$ 0.015
website_phishing	0.965 $\pm$ 0.007	0.974 $\pm$ 0.005	0.972 $\pm$ 0.005	0.972 $\pm$ 0.005	0.969 $\pm$ 0.005	0.923 $\pm$ 0.009	0.969 $\pm$ 0.006

Table 5: Mean parameter values corresponding to the maximum estimated optimality gap for each max-min iteration of the RTFM algorithm. We observe significant variation in these parameters across iterations.

max-min iter.	$\mu_l$	$\mu_h$	$\mu_{x_{in}}$	$\mu_{z_x}$	$\mu_{r_s}$	$\mu_c$	$\mu_a$	$\mu_{r_{cat}}$	$\mu_m$	$\mu_d$
1	5	10	3	5	1.0	8	tanh	0.9	0.4	uniform
2	3	5	2	5	0.1	2	identity	1.0	0.4	uniform
3	12	128	13	200	0.0	2	identity	0.4	0.0	uniform
4	3	5	2	6	0.6	2	tanh	1.0	0.0	exponential
5	10	64	8	32	0.4	2	tanh	1.0	0.0	uniform
6	8	32	5	13	0.8	10	identity	0.0	0.2	normal
7	10	64	8	162	0.9	6	elu	0.6	0.0	normal
8	5	10	3	13	0.0	10	tanh	0.8	0.0	uniform
9	5	10	3	5	0.3	6	identity	0.6	0.0	exponential
10	5	10	3	5	0.1	10	tanh	0.5	0.0	normal
11	3	5	2	5	0.0	8	relu	1.0	0.0	normal
12	3	5	2	5	0.6	4	relu	0.6	0.0	exponential
13	3	5	2	5	0.4	6	tanh	0.6	0.0	normal
14	5	10	3	5	0.8	8	identity	0.8	0.0	exponential
15	3	5	2	5	1.0	8	elu	0.6	0.0	uniform
16	3	5	2	6	0.7	6	identity	0.3	0.0	uniform
17	8	32	5	91	0.0	2	tanh	1.0	0.0	exponential
18	3	5	2	5	0.0	8	tanh	1.0	0.0	exponential
19	3	5	2	6	0.2	6	tanh	0.2	0.0	exponential
20	12	128	13	200	0.2	2	identity	0.1	0.0	exponential
21	3	5	2	6	0.9	8	relu	0.6	0.0	exponential
22	3	5	2	5	0.3	8	identity	0.2	0.0	uniform
23	5	10	3	5	0.2	4	tanh	0.3	0.0	normal
24	3	5	2	5	0.8	8	identity	0.1	0.0	uniform
25	3	5	2	5	0.2	4	relu	1.0	0.0	exponential
26	3	5	2	5	0.5	8	tanh	0.8	0.0	normal
27	8	32	5	13	0.1	4	identity	0.0	0.0	exponential
28	3	5	2	6	0.4	4	tanh	0.1	0.0	exponential
29	3	5	2	5	0.1	8	identity	0.2	0.0	exponential