7 ETHICS STATEMENTS

In terms of broader societal impact of this work, we do not see any foreseeable strongly negative impacts. However, this paper could positively impact the carbon footprint and accessibility of learning algorithms. The computations required for machine learning research have been doubling every few months, resulting in a large carbon footprint (Schwartz et al., 2020). Moreover, the financial cost of the computations can make it difficult for academics, students, and researchers to apply these methods. The decreased computational time shown by TabPFN translates to reductions in CO2 emissions and cost, making it available to an audience that does not have access to larger scale computing.

As the TabPFN provides a highly portable and convenient way of building new classifiers that work in real-time, it is likely to increase the pervasiveness of machine learning even further. While this can have many positive effects on society, such as better personalized healthcare, increased customer satisfaction, efficiency of processes, etc, it will also be crucial to study and improve the TabPFN under the lens of the many dimensions of trustworthy AI other than computational sustainability, such as algorithmic fairness, robustness to adversarial examples, explainability, and auditability. We hope that its foundation in causal models and simplicity will allow possible avenues for work along these lines.

8 REPRODUCIBILITY

Code release In an effort to ensure reproducibility, we release code alongside our pre-trained TabPFN and notebooks to reproduce our experiments at `https://github.com/automl/TabPFN`.

Application to public benchmarks In our work, we evaluate TabPFN to publicly available benchmarks: the OpenML-CC18 Benchmark and the OpenML-AutoML Benchmark. This ensures, that dataset choices are not cherry-picked to our method. Also, for the OpenML-AutoML Benchmark, we use official baseline results and evaluate our method using the evaluation scripts published for this benchmark.[6]

Availability of datasets All datasets used in our experiments are freely available at `OpenML.org` (Vanschoren et al., 2014), with downloading procedures included in the submission. Further details on the datasets used can be found in Section F.3.

Online resources We created a Colab notebook, that lets you interact with our scikit-learn interface at `https://colab.research.google.com/drive/1J0l1AtMV_H1KQ7IRbgJje5hMhKHczH7-?usp=sharing`.

We created another Colab, where our evaluation and plots on 179 test and validation datasets can be reproduced easily. `https://colab.research.google.com/drive/1yUGaAf3D7RSyO5Jc4PYXSVbtaUAozh6S`

We also created two demos. One to experiment with the TabPFNs predictions (`https://huggingface.co/spaces/TabPFN/TabPFNPrediction`) and one to check cross-validation ROC AUC scores on new datasets (`https://huggingface.co/spaces/TabPFN/TabPFNEvaluation`). Both of them run on a weak CPU, thus it can require a little bit of time.

Details of training procedures for TabPFN and baselines Details shared in the training procedure of all our Transformer models can be found in Appendix F. An overview of the hyperparameters used for running the TabPFN and our baselines can be found in Tables 5 and 6, respectively.

ACKNOWLEDGMENTS

Robert Bosch GmbH is acknowledged for financial support. This research was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828, the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG, and TAILOR, a project funded by EU Horizon 2020 research, and innovation programme under GA No 952215. We acknowledge funding through the

[6] Available at https://github.com/openml/automlbenchmark

European Research Council (ERC) Consolidator Grant "Deep Learning 2.0" (grant no. 101045765). Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the ERC. Neither the European Union nor the ERC can be held responsible for them.

REFERENCES

S. Arik and T. Pfister. TabNet: Attentive interpretable tabular learning. In Q. Yang, K. Leyton-Brown, and Mausam, editors, *Proceedings of the Thirty-Fifth Conference on Artificial Intelligence (AAAI'21)*, pages 6679–6687. AAAI Press, 2021.

I. Beltagy, M. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv:2004.05150 [cs.CL]*, 2020.

B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. Mantovani, J. van Rijn, and J. Vanschoren. OpenML benchmarking suites. In J. Vanschoren, S. Yeung, and M. Xenochristou, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *arXiv:2110.01889 [cs.LG]*, 2021.

L. Breimann. Random forests. *Machine Learning Journal*, 45:5–32, 2001.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pages 785–794. ACM Press, 2016.

T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv:2006.10029v2 [cs.LG]*, 2020.

M. Chui, J. Manyika, M. Miremadi, N. Henke, R. Chung, P. Nel, and S. Malhotra. Notes from the AI frontier: insights from hundreds of use cases, 2018.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv:2003.06505 [stat.ML]*, 2020.

M. Feurer and F. Hutter. Hyperparameter Optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pages 3–38. Springer, 2019. Available for free at http://automl.org/book.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 2962–2970. Curran Associates, 2015.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-sklearn 2.0: Hands-free AutoML via meta-learning. *arXiv:2007.04074 [cs.LG]*, 2021.

J. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv:2207.08815 [cs.LG]*, 2022.

D. Janzing. Causal regularization. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*. Curran Associates, 2020.

A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. Well-tuned simple nets excel on tabular datasets. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 23928–23941. Curran Associates, 2021.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, 2017.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015. Published online: iclr.cc

J. Kossen, N. Band, C. Lyle, A. Gomez, T. Rainforth, and Y. Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 28742–28756. Curran Associates, 2021.

T. Kyono, Y. Zhang, and M. van der Schaar. CASTLE: Regularization via auxiliary causal graph discovery. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 1501–1512. Curran Associates, 2020.

T. Kyono, Y. Zhang, A. Bellot, and M. van der Schaar. Miracle: Causally-aware imputation via learning missing data mechanisms. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Advances in Neural Information Processing Systems*, pages 23806–23817. Curran Associates, 2021.

L. Lin, M. Sperrin, D. A Jenkins, G. Martin, and N. Peek. A scoping review of causal methods enabling predictions under hypothetical interventions. *Diagnostic and prognostic research*, 5(1): 1–16, 2021.

I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: iclr.cc.

S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, 2022. URL https://openreview.net/forum?id=KSugKcbNf9. Published online: iclr.cc.

V. Nair and G. Hinton. Rectified linear units improve restricted Boltzmann machines. In J. Fürnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*. Omnipress, 2010.

R. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Computer Science. Springer, 1996.

A. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In R. Greiner, editor, *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Omnipress, 2004.

J. Pearl. *Causality*. Cambridge University Press, 2 edition, 2009.

J. Pearl. Causal inference. *Causality: objectives and assessment*, pages 39–58, 2010.

J. Pearl and D. Mackenzie. *The Book of Why*. Basic Books, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin. CatBoost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*. Curran Associates, 2018.

D. Rothenhäusler, N. Meinshausen, P. Bühlmann, and J. Peters. Anchor regression: heterogeneous data meets causality. *arXiv:1801.06229 [stat.ME]*, 2018.

J. Schmidhuber. The speed prior: A new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. Sloan, editors, *Computational Learning Theory*, pages 216–228. Springer, 2002.

R. Schwartz, J. Dodge, N. Smith, and Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.

B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij. On causal and anticausal learning. In J. Langford and J. Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML'12)*, pages 459–466. Omnipress, 2012.

R. Shwartz-Ziv and A. Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

G. Somepalli, M. Goldblum, A. Schwarzschild, C. Bruss, and T. Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv:2106.01342 [cs.LG]*, 2021.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–1958, 2014.

Bojan Tunguz. Xgboost search space tweet. https://twitter.com/tunguz/status/1572642449302106112, 2022. Accessed: 2022-09-28.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2014.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, Inc., 2017.

M. Waldmann and Y. Hagmayer. Causal reasoning. *The Oxford handbook of cognitive psychology*, pages 733—-752, 2013.

F. Wenzel, J. Snoek, D., Tran, and R. Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 6514–6527. Curran Associates, 2020.

Z. Wojtowicz and S. DeDeo. From probability to consilience: How explanatory values implement bayesian reasoning. *Trends in Cognitive Sciences*, 24(12):981–993, 2020.

Yanzhao Wu, Ling Liu, Zhongwei Xie, Ka-Ho Chow, and Wenqi Wei. Boosting ensemble accuracy by revisiting ensemble diversity metrics. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16464–16472, 2021. doi: 10.1109/CVPR46437.2021.01620.

I. Yeo and R. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

M. Zaheer, G. Guruganesh, K. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang Qifan, L. Yang, and A. Amr. Big bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 17283–17297. Curran Associates, 2020.

S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. Teh. Neural ensemble search for uncertainty estimation and dataset shift. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, pages 7898–7911. Curran Associates, 2021.

B. Zoph and Q. V. Le. Neural Architecture Search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: iclr.cc.

A LIMITATIONS

The runtime and memory usage of the Transformer-based PFN architecture used in this work scales quadratically with the number of inputs, i.e. training samples passed. Thus, inference on larger sequences ($> 100\,000$) is hard on current consumer GPUs. A growing number of methods seek to tackle this issue and report similar performances while scaling linearly with the number of inputs (Zaheer et al., 2020; Beltagy et al., 2020). These methods can be integrated into the PFN architecture and thus into the TabPFN. Furthermore, in our experiments we limit the number of features to 100 and the number of classes to 10 as described in Section 5. While this choice is flexible, the precise TabPFN that we fitted cannot work with datasets that go beyond these limits. We also focused the development of TabPFN to purely numerical datasets without missing values, and while they *can* be applied to datasets with categorical features and/or missing values, their performance is generally worse. We hope to tackle this problem in future versions of TabPFN by a modified architecture and prior. Finally, we did not consider the existence of many uninformative features in our prior, leading to performance degradation when such features are added; we hope to address this issue in future versions of TabPFN. While baseline models (except Gaussian Processes) fit and inference in separate steps, TabPFN performs both at the same time. Thus when pretrained baseline models are used, inference is fast (see Table 2).

B ADDITIONAL RESULTS

B.1 DETAILED TABULAR RESULTS

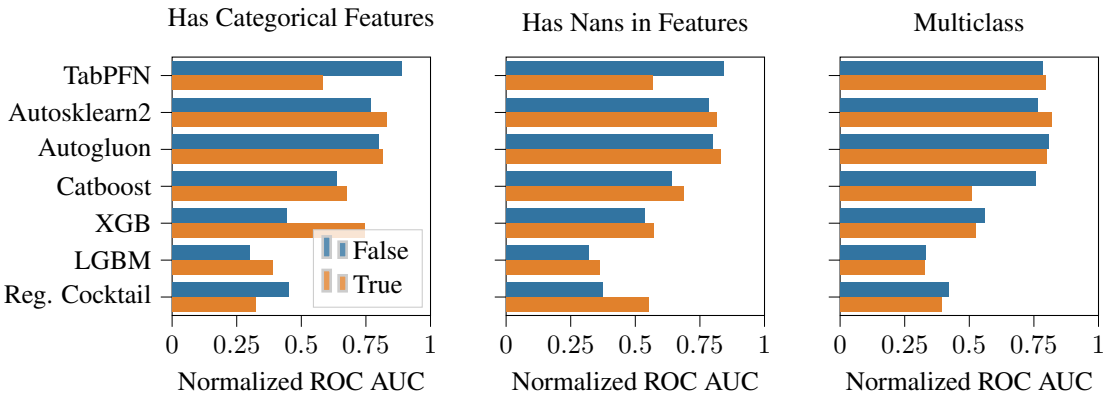

Figure 6: Normalized ROC AUC performance on datasets from the OpenML-CC18 Benchmark, divided by dataset characteristics. For each plot, we split the datasets into two groups. Left: Orange bars indicate the performance on datasets that have categorical features. Middle: Orange bars indicate datasets that contain missing values. Right: Orange bars indicate multiclass datasets, while others are binary.

In Figure 6, we explore how the kind of dataset evaluated affects the performance of TabPFN, compared to our baselines. We find that TabPFN performs much better when no categorical features are present. We also find that TabPFN performs better, when no missing values are present in the data. This warrants an extension of our prior in future work, to make it more customized towards categorical and missing data. Our method seems to work comparably well for binary and multi-class problems.

In addition to the results in the main paper in Section 5.2, we report detailed results on all 30 datasets in OpenML-CC18, which are small enough, but might include categorical features and missing values. We show performance over time in Figure 7 and a wide range of performance values and per dataset results with a 1 hour time limit in Table 2.

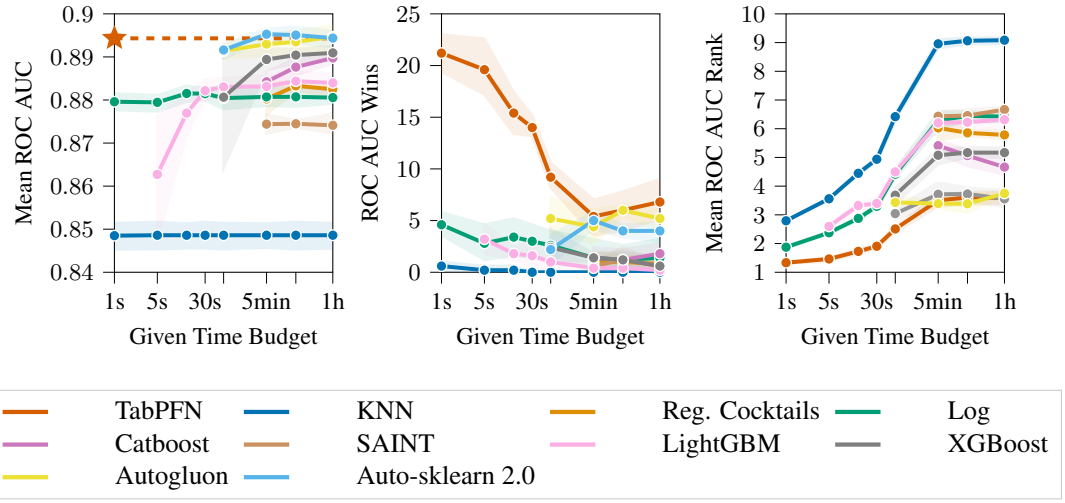

Figure 7: ROC AUC performance over time on the 30 small OpenML-CC18 including datasets with categorical features and missing values. We report the mean, mean wins and rank along with the 95% confidence interval across 5 splits for increasing training and tuning time budgets (Unlabelled ticks: 1min, 15min). The red star indicates performance of TabPFN with 32 data permutations (which requires 0.42s on GPU). We report detailed results with a 60 min budget in Table 1.

B.2 Results on the OpenML-AutoML Benchmark

We evaluate TabPFN using the official benchmarking scripts[7] datasets, splits and baselines results of the OpenML-AutoML Benchmark. The full list of datasets can be found in Table 10. We note that these datasets are not disjoint from our evaluation datasets, in fact 4 evaluated datasets from the OpenML-AutoML Benchmark were included in our evaluation datasets from the OpenML-CC18 Benchmark as well, while one dataset ("Australian") was included in our list of 150 meta-validation datasets. The evaluation on another set of benchmarking scripts with previously published train-test splits and baseline results, however, help to confirm that: (1) Our baselines are well tuned and not outperformed by another method in the extensive OpenML-AutoML Benchmark; (2) the runtimes of TabPFN are reproducible in controlled environment provided by the OpenML-AutoML Benchmark; and (3) TabPFN is not overfit to datasplits or our evaluation metric. While we used a 50-50 train-test split with 5 iterations for our experiments in Table 3, the OpenML-AutoML Benchmark uses a 10-fold cross-validation, which results in a 90-10 splits with 10 iterations. Thus in the OpenML-AutoML Benchmark, all methods use more training samples as in our experiments on the OpenML-CC18 Benchmark, which leads to slightly stronger results.

B.3 In-depth analysis of model biases

Previous work by Grinsztajn et al. (2022) empirically investigates the inductive biases of tree-based and deep-learning models. They identify three challenges in developing tabular-specific models. Models must be (1) able to fit irregular functions; (2) robust to uninformative features; and (3) preserve the orientation of the data. We perform and extend this analysis in the following section. We consider three model types in our analyses: TabPFN, GBDTs (LightGBM) and NNs (Standard Sklearn Multi Layer Perceptron with a hidden deimensionality of 100). We do not seek to make absolute comparisons between these model types, which would warrant tuning of our baselines, but only seek to investigate their qualitative behavior in the following experiments.

B.3.1 Fitting irregular patterns

While GBDT methods learn non-smooth and irregular patterns in the targets, MLPs learn smooth, low-frequency functions, as can be seen in Figure 4. The work by (Grinsztajn et al., 2022) suggests that GBDT methods perform well since they are able to learn non-smooth patterns. Prior works

[7] Available at https://github.com/openml/automlbenchmark

Table 2: ROC AUC OVO results on the 30 small OpenML-CC18 (including datasets with categorical features and missing values) for 60 minutes requested time per dataset and per split. If available, all baselines are given ROC AUC optimization as an objective, others optimize CE. Overall each method got a time budget of 150 hours, but not all methods used the full budget. Times for TabPFN refer to times on GPU. TabPFN runs training, tuning and prediction in a joint step, so only the aggregate time is shown.

	LightGBM	CatBoost	XGBoost	ASKL2.0	AutoGluon	TabPFN$_{n.e.}$	TabPFN	TabPFN + AutoGluon
balance-scale	0.9938	0.9245	0.9939	0.997	0.9919	0.9965	**0.9973**	0.9958
mfeat-fourier	0.9786	0.9816	0.9803	0.9826	**0.9843**	0.9767	0.9811	0.9838
breast-w	0.991	0.9931	0.9896	0.9939	0.9933	0.9931	0.9934	**0.994**
mfeat-karhunen	0.9979	0.9986	0.9983	0.9975	**0.9987**	0.9939	0.9978	0.9985
mfeat-morphologica..	0.9601	0.9629	0.9612	0.9671	0.9698	0.9657	0.9669	**0.9722**
mfeat-zernike	0.9716	0.9759	0.9735	0.9812	**0.9908**	0.9812	0.9823	0.9901
cmc	0.7288	0.7256	0.7299	**0.7378**	0.7331	0.7233	0.7276	0.7336
credit-approval	0.9415	0.9389	**0.9422**	0.9406	0.9415	0.9253	0.9322	0.9394
credit-g	0.7684	0.7852	0.7853	0.793	0.7941	0.7894	0.7894	**0.7948**
diabetes	0.8247	0.8383	0.8378	0.8343	0.8391	0.8412	0.841	**0.8427**
tic-tac-toe	0.9988	0.9992	**1**	0.9943	**1**	0.9547	0.9759	0.9992
vehicle	0.9232	0.9302	0.9282	0.9504	0.9416	0.9568	**0.9589**	0.9538
eucalyptus	0.8931	0.8979	0.9004	0.9132	0.9204	0.9218	0.9245	**0.9278**
analcatdata_author..	0.9999	0.9999	0.9997	**1**	0.9993	**1**	**1**	**1**
analcatdata_dmft	0.5461	0.5589	0.5743	0.5752	0.5657	0.5643	**0.579**	0.5756
pc4	0.9301	0.9413	0.9291	0.9331	0.9428	0.9298	0.9383	**0.944**
pc3	0.8178	0.8247	0.8288	0.8265	0.8282	0.8308	**0.8373**	0.836
kc2	0.8141	0.8323	0.8227	0.8311	0.8242	0.8322	**0.8346**	0.8321
pc1	0.8321	0.86	0.8489	0.8527	0.8578	**0.877**	0.8761	0.8739
banknote-authentic..	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
blood-transfusion-..	0.7144	0.7403	0.7312	0.7504	0.7364	0.753	**0.7549**	0.7469
ilpd	0.6917	0.7279	0.7171	0.7212	0.723	**0.7412**	0.7379	0.7326
qsar-biodeg	0.9126	0.9217	0.9191	0.9247	0.9276	**0.9345**	0.9336	0.9336
wdbc	0.9904	0.9931	0.9904	0.9947	0.9956	0.996	**0.9964**	0.996
cylinder-bands	0.8556	0.8757	0.8782	0.8718	**0.8878**	0.8314	0.8336	0.8751
dresses-sales	0.5593	0.5696	**0.5823**	0.5705	0.5507	0.5333	0.5376	0.5509
MiceProtein	0.9997	0.9999	0.9998	0.9999	**1**	0.9997	0.9999	**1**
car	0.9925	0.9955	0.9948	**0.998**	0.997	0.9926	0.995	0.9972
steel-plates-fault..	0.9626	0.9655	0.9656	**0.9694**	0.9666	0.9619	0.9655	0.9687
climate-model-simu..	0.9286	0.9344	0.9255	0.9291	0.9391	**0.9426**	0.9415	0.9421
Wins AUC OVO	0	0	2	2	2	4	**5**	**5**
Wins Acc.	0	2	2	3	3	0	6	**8**
Wins CE	0	1	3	1	7	1	6	**9**
M. rank AUC OVO	6.6167	4.9667	5.4167	4.05	3.7833	4.65	3.7	**2.8167**
Mean rank Acc.	6.5333	4.9833	5.1833	4.8667	3.8167	4.5333	3.6167	**2.4667**
Mean rank CE	5.7333	5.6	5.4667	5.8	2.8667	4.6167	3.5333	**2.3833**
Win/T/L AUC vs Tab..	5/4/21	9/4/17	6/5/19	10/6/14	13/4/13	4/8/18	–/–/–	15/7/8
Win/T/L Acc vs Tab..	6/0/24	9/1/20	11/0/19	11/2/17	12/0/18	6/3/21	–/–/–	19/3/8
Win/T/L CE vs TabP..	6/0/24	8/0/22	8/0/22	8/0/22	20/0/10	1/4/25	–/–/–	23/0/7
Mean AUC OVO	0.884±.012	0.89±.011	0.891±.011	0.894±.01	0.895±.01	0.891±.01	0.894±.01	**0.898**±.0097
Mean Acc.	0.815±.014	0.818±.011	0.821±.013	0.821±.016	0.83±.012	0.82±.013	0.825±.012	**0.834**±.011
Mean CE	0.782±.074	0.767±.061	0.758±.047	0.815±.06	**0.72**±.015	0.742±.021	0.732±.018	0.721±.015
Time Tune + Train (s)	3241	3718	3304	3601	3127	**0.0187**	0.4197	3127
Predict (s)	0.0815	0.0168	0.0685	1.224	21.18			

on regularization and an intuitive exploration of the target decision boundary of GBDT methods in Figure 4, suggests , however, that smooth target functions are desirable. TabPFN learns target functions that are rather smooth, as suggested in Figure 4. This is due to our models prior, which prefers simple SCMs as explanations and thus less irregular decision planes. We note, however, that when many training samples are provided, TabPFN fits more complex functions.

The tradeoff between complexity and number of training samples is explored in Figure 8. Here, we evaluate the training set uncertainty (cross-entropy loss) on synthetic data generated from random SCMs. The number of training samples and the complexity of the generated data (number of hidden units in the data generating graph) is varied.

B.3.2 Robustness to uninformative features

Tabular datasets contain a large fraction of uninformative features (Grinsztajn et al., 2022). To evaluate robustness to uninformative features, we add an increasingly large fraction of uninformative features to our data and show results in Figure 9. Uninformative features are generated by copying existing features and shuffling their values randomly between samples. We find that TabPFN and MLPs are less robust to uninformative features than LightGBM. TabPFN could be adapted by including more uninformative features in the used prior. In a second experiment we drop an increasingly large

Table 3: ROC AUC OVO results on the 5 small datasets ($\leq$ 1 111 examples, 100 features and 10 classes) for 60 minutes requested time per dataset and per split. If available, all baselines are given ROC AUC optimization as an objective, others optimize CE. Evaluation on this benchmark was performed with our previously released TabPFN in order to mitigate test-set overfitting. Mean OpenML-Metric is not shown in the table as averaging over a mixture of Cross Entropy and ROC AUC is problematic (however, TabPFN had the strongest average as well).

	AutoGluon	ASKL	ASKL2.0	TunedRandomForest	FLAML	TPOT	TabPFN
vehicle	-0.3084	-0.3816	-0.3412	-0.4849	-0.4286	-0.3433	**-0.2955**
eucalyptus	-0.6905	-0.7255	-0.6967	-0.7209	-0.7433	-0.7123	**-0.665**
blood-transfusion-service-center	0.7532	0.749	0.7557	0.6879	0.7332	0.7359	**0.7593**
Australian	0.941	0.9315	**0.9411**	0.9394	0.9356	0.9382	0.9395
credit-g	0.7977	0.7891	0.7984	**0.8017**	0.7838	0.7821	0.7989
Wins OpenML Metric	0	0	0	1	0	0	**3**
Wins Acc.	1	0	0	1	0	1	**2**
Wins CE	**3**	0	0	0	0	0	2
M. rank OpenML Metric	2.4	5.4	2.6	4.8	6.2	5	**1.6**
Mean rank Acc.	2.4	5.7	4.7	4.4	5.7	3	**2.1**
Mean rank CE	**1.4**	5.8	4.4	5.4	4.4	4.7	1.9
Mean Acc.	0.793±.031	0.763±.052	0.775±.052	0.763±.039	0.761±.035	0.784±.031	**0.794**±.033
Mean CE	0.454±.039	0.537±.061	0.502±.056	0.545±.079	0.499±.048	0.73±.7	**0.449**±.05
Mean time (s)	3182	3611	3609	2877	3600	3400	**4.374** (CPU)

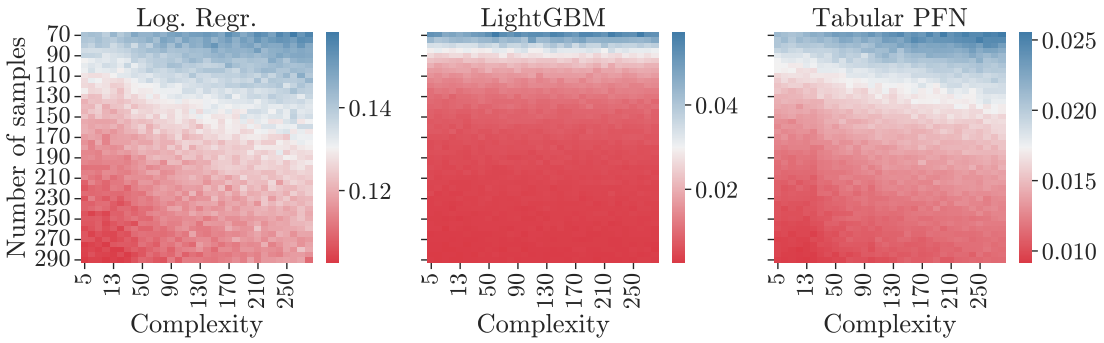

Figure 8: Mean training set uncertainty (cross-entropy loss) on synthetic data generated from random SCMs. The number of training samples is varied on the y-axis and the complexity of the generated data (number of hidden units in the data generating graph) on the x-axis. The cross-entropy mean is averaged across the 100 samples and 1 000 SCMs for each point.

fraction of features. We drop these features according to feature importance (ranked by a Random Forest), first removing least informative features. We show results in Figure 9 and observe that the classification accuracy of a TabPFN and GBDT is not much affected by removing up to 30% of the features, but constantly diminishes. The MLP, which is less not robust to uninformative features, initially performs even better when features are removed.

We use our testing tasks from the OpenML CC-18 Benchmark for all evaluations. To simplify analyses, we drop multiclass datasets and datasets that contain more than 50 features (As adding 100% more features to a dataset with more than 50 features yields more than 100 features, which TabPFN cannot handle).

B.3.3 Invariance to feature rotation

Each feature of a tabular dataset typically carries meaning individually, as expressed by column names, such as age or sex. A learning algorithm is rotationally invariant in the sense of Ng (2004), if it is left unchanged when a rotation (unitary) matrix is applied to the features of both the training and testing set, i.e. when features are mixed. To remove uninformative features under a feature rotation, an algorithm first has to restore the original orientation of the features, and then select

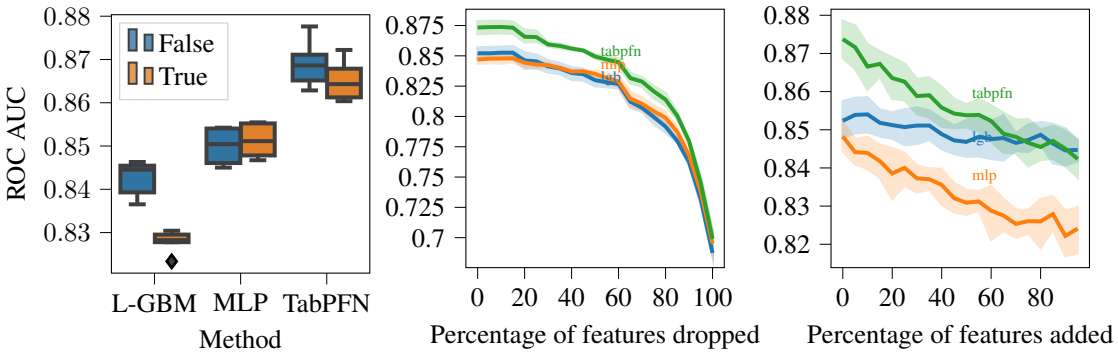

Figure 9: Left: Performance of LightGBM, MLP and TabPFN when random rotations are applied to the feature space. LightGBM loses most predictive accuracy when rotations are applied, MLP is unaffected and TabPFN performs only slightly worse. Center: Removing uninformative features leads to MLPs initially performing better and overtaking LightGBM performance. TabPFN looses predictive accuracy and performs similar to baselines, when most features are removed. Right: Adding uninformative features leads to performance degradation of MLPs and TabPFN, while LightGBM remains relatively constant.

informative ones. A rotationally invariant algorithm discards the data orientation and thus has to restore the original orientation internally. Thus, Ng (2004) shows that any rotationally invariant learning algorithm has a worst-case sample complexity that grows at least linearly in the number of irrelevant features.

Figure 9 shows the change in test ROC AUC when randomly rotating our datasets, and confirms that only MLPs are rotationally invariant. GBDT methods are highly sensitive to rotations, while TabPFN is less sensitive, but still performs better when no rotations are applied.

The theoretical results by Ng (2004) and empirical results by Grinsztajn et al. (2022) imply, that TabPFN's diminishing performance when uninformative features are added is linked to it's relative rotation invariance. Adjusting the prior to include more uninformative features could address these results.

We use our testing tasks from the OpenML CC-18 Benchmark for all evaluations. To simplify analyses, we drop multiclass datasets and datasets that contain categorical data. Rotating categorical datasets is problematic, as some GBDT classifiers treat categorical variables distinctly (e.g. generating embeddings per category).

B.4 Ablation on the selection of prior models

We perform ablation experiments for a prior based solely on BNNs, SCMs and a mix of both using a hyperparameter to control the sampling likelihood for either. The PFNs for each prior are fitted using less compute than in our final experiments, thus their scores generally are slightly worse. The BNN prior, similar to the one used in Müller et al. (2022), provides diminished performance compared to the priors based on causal models. Additionally, we can see that mixing BNN and SCM prior does not seem to make a big difference on our test set compared to a pure SCM prior.

	BNN	SCM	SCM + BNN
Mean CE	0.811±0.009	**0.771**±0.006	0.776±0.009
Mean ROC AUC	0.865±0.007	0.881±0.002	**0.883**±0.003

Table 4: An evaluation of the impact of the prior mixing on the final performance. Our final model was trained in the *SCM + BNN* setting (see Section 4 for details on the priors).

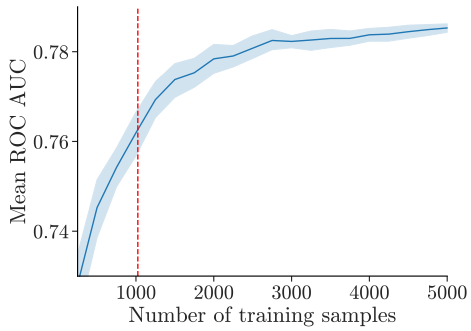

Figure 10: Extrapolation performance of our TabPFN to dataset sizes never seen during training. Maximum number of samples during training was 1024 (dashed red line). The shading indicates the 95% confidence interval over random data splits. A method that does not generalize, would be expected to flatten at 1024.

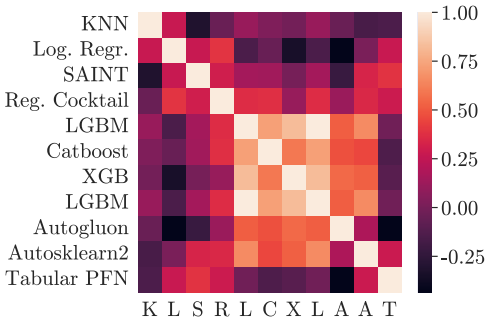

Figure 11: Spearman correlation of per-dataset normalized ROC AUC performance (i.e. the ranking correlation of per-dataset normalized ROC AUC scores) between the considered methods. Ordering on the x-axis is the same as on the y-axis. The TabPFN performs well on a different set of datasets than the baselines, i.e. the Spearman correlation with the GBDT methods is low, while GBDT and AutoML methods are highly correlated, and thus perform well on the same datasets. TabPFN correlates stronger with DL based methods (SAINT and Reg. Cocktail), which, however, do not perform as well as GBDT methods in terms of absolute ROC AUC performance (see 1)

B.5 Extended analysis on a larger benchmark of datasets

We provide an extended benchmark where we:

i) Include an analysis on the additional 149 validation datasets [8] (see 8).

ii) Include default configurations of our baselines.

iii) Run one step of the default configuration of our baselines initially and then tuning them.

iv) Added SVM, Gradient Boosting and Random Forests as a Baseline. and report results in Figure 14 and Figure 15. The benchmark was done across 5 splits and followed the same experimental setup described in F

[8] The dataset $flags$ was removed as not enough splits could be generated by our code.

C Details of the TabPFN Prior

C.1 SCM Prior

The Sampling Algorithm We instantiate a subfamily of DAGs that can be efficiently sampled from by starting with a MLP architecture and dropping weights from it. That is, to sample a dataset with k features and n samples from our prior we perform the following steps for each dataset:

(1) We sample the number of MLP layers $l \sim p(l)$ and nodes $h \sim p(h)$ and sample a graph $\mathcal{G}(Z, E)$ structured like an l-layered MLP with hidden size h.

(2) We sample weights for each Edge E_{ij} as $W_{i,j} \sim p_w(\cdot)$.

(3) We drop a random set of edges $e \in E$ to yield a random DAG.

(4) We sample a set of k feature nodes N_x and a label node N_y from the nodes Z.

(5) We sample the noise distributions $p(\epsilon) \sim p(p(\epsilon))$ from a meta-distribution. This yields an SCM, with all f_i's instantiated as random affine mappings followed by an activation. Each z_i corresponds to a sparsely connected neuron in the MLP.

With the above parameters fixed, we perform the following steps for each member of the dataset:

(1) We sample noise variables ϵ_i from their specific distributions.

(2) We compute the value of all $z \in Z$ with $z_i = a((\sum_{j \in \text{PA}_{\mathcal{G}(i)}} E_{ij} z_j) + \epsilon_i)$.

(3) We retrieve the values at the feature nodes N_x and the output node N_y and return them.

We sample one activation function a per dataset from $\{Tanh, LeakyReLU, ELU, Identity\}$ (Nair and Hinton, 2010). The sampling scheme for the number of layers $p(l)$ and nodes $p(h)$ is designed to follow a discretized noisy log-normal distribution, $p(\epsilon)$ is a noisy log-normal distribution and the dropout rate follows a beta distribution. The full information can be found in Table 5.

C.2 Tabular Data Refinements

Tabular datasets comprise a range of peculiarities, e.g. feature types can be numerical, ordinal, or categorical and feature values can be missing, leading to sparse features. We seek to reflect these peculiarities in the design of our prior as described in the following sections.

C.2.1 Preprocessing

During meta-training, input data is normalized to zero mean and unit variance, and we apply the same step when evaluating on real data. Since tabular data frequently contains exponentially scaled data, which might not be present during meta-training, we apply power scaling during inference (Yeo and Johnson, 2000). Thus, during inference on real tabular datasets the features more closely match those seen during meta-training. We use only training samples for calculating z-statistics, power transforms and all other preprocessing. We take this preprocessing time into account when reporting the inference time of our method.

C.2.2 Correlated Features

Feature correlation in tabular data varies between datasets and ranges from independent to highly correlated. This poses problems to classical deep learning methods (Borisov et al., 2021). When considering a large space of SCMs, correlated features of varying degrees naturally arise in our priors. Furthermore, in real-world tabular data, the ordering of features is often unstructured, however adjacent features are often more highly correlated than others. We use "Blockwise feature sampling" to reflect the correlation structure between ordered features. Our generation method of SCMs naturally provides a way to do this. The first step in generating our SCMs is generating a unidirectional layered network structure in which nodes in one layer can only receive inputs from the preceding layer. Thus, features in the same layer tend to be more highly correlated. We use this by sampling adjacent nodes in the layered network structure in blocks and using these ordered blocks in our set of features. In Figure 12, we visualize the correlations of such a generated dataset (right) and compare them to a real-world dataset (left), demonstrating that our prior yields correlation structures similar to those of real datasets.

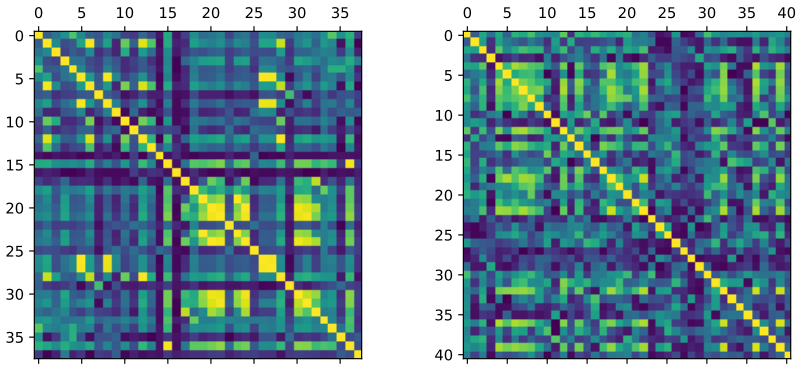

Figure 12: Feature correlation matrices for a real-world ("PC4 Software defect prediction", left) and a synthetic (right) dataset, where brighter colors indicate higher correlation.

C.2.3 Generating irregular functions

In real-world data, some features are consistently more important than others. While a random network weight initialization leads to slightly different feature importances, the average effect of input features regresses to the mean when the hidden dimensionality increases. We amplify differences by sampling a weight parameter for each input feature and multiplying all outgoing weights by this factor. In the prior, we randomly sparsify connections of the graph. Thus hidden variables and the output node are influenced by fewer parameters, yielding more irregular patterns, as a larger number of parameters once again regresses to the mean. We also extend sparsification to blocks of variables, leading to some groups of variables interacting more strongly. We also extend the way noise variables are sampled. Instead of sampling Gaussian noise at each node from the same distribution, we first sample separate noise means and standard deviations for each node and then sample from this distribution. Also, we generate non-uniformly distributed input data x, as observed in real-world data: We sample the input variables x (which are propagated through our network), from a mix of distributions, namely the Gaussian, Zipfian and Multivariate Distribution.

C.2.4 NaN Handling

We do not have special nan handling built into our model. We replace nan values with zero at test time.

C.2.5 Categorical Features

Tabular data often includes not only numeric features but also discrete categorical ones. While categorical features should technically not be ordered, in practice, they sometimes are, i.e., the categories represent binned degrees of some underlying variable. We introduce categorical features by picking a random fraction p_{cat} (a hyperparameter) of categorical features per dataset. Analogous to transforming numeric class labels to discrete multiclass labels, we convert dense features to discrete ones. Also analogous to multiclass labels, we pick a shuffling fraction of categorical features p_{scat} where we reshuffle categories. For details, see Section 4.5. During prior-fitting, we use a probability for categorical features of 20%.

C.2.6 Prior work on PFNs for Tabular Data

Prior work by Müller et al. (2022) has demonstrated tabular data classification using PFNs, but was limited to 30 training samples, balanced binary-classification and 60 features. Here we summarize the most important changes, to this prior work:

i) The PFNs for tabular data described in Müller et al. (2022) can only handle balanced binary datasets, in Section 3.5 we show how to extend the prior s.t. we can handle unbalanced data.

ii) We looked into pre-processing techniques for the TabPFN, which was not done at all before. Finally, we ensemble over different pre-processing pipelines, which include (power transforms and outlier removal) and feature/class rotations.
iii) We changed the transformer architecture s.t. it is faster. We shrank attention matrix sizes from $(n+m)^2$ to $n^2 + n * m$, for n training points and m inference points.
iv) We introduce a novel SCM prior. We compare the SCM, the improved BNN prior (whcih is more heavily based on the BNN setup of C.2.6) and SCM + BNN in Table 4. It pushes performance by 2% in this smaller scale setup, which is a bigger difference than between the final TabPFN and all baselines besides KNN and SAINT.

D Details of the Prior-Data Fitted Network Algorithm

Algorithm 1 describes the training method proposed by Müller et al. (2022) for PFNs.

Algorithm 1: Meta-Training of a PFN (Müller et al., 2022)

Input : A prior distribution over datasets $p(D)$, from which samples can be drawn and the number of samples K to draw
Output : A model q_θ that will approximate the PPD
Initialize the neural network q_θ;
for $j \leftarrow 1$ **to** K **do**
  Sample $D \cup \{(x_i, y_i)\}_{i=1}^{m} \sim p(D)$;
  Compute stochastic loss approximation $\bar{\ell}_\theta = \sum_{i=1}^{m}(-\log q_\theta(y_i|x_i, D))$;
  Update parameters θ with stochastic gradient descent on $\nabla_\theta \bar{\ell}_\theta$;
end

E Setup of our method

E.1 Transformer Hyperparameters

We considered only PFN Transformers with 12 layers, embeddings size 512, hidden size 1024 in feed-forward layers, and 4-head attention. We used the Adam optimizer (Kingma and Ba, 2015) with linear-warmup and cosine annealing (Loshchilov and Hutter, 2017). For each training we tested a set of 3 learning rates, $\{.001, .0003, .0001\}$, and used the one with the lowest final training loss. The resulting model contains 25.82 M parameters.

E.2 PFN Architecture Adaptations

Attention Adaption The original PFN architecture (Müller et al., 2022) uses a single multi-head self-attention module (Vaswani et al., 2017) to compute the attention between all the training examples, as well as, the attention from validation examples to training examples. We replaced this, with two modules that share weights, one which computes self-attention among the training examples and the other that only compute cross-attention from validation examples to training examples. Conceptually, this is equivalent to the original architeture, except that we're using a slightly different self-attention mask than the original architecture, which allowed all examples to attend to itself (the diagonal is 1), as in this example:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{3}$$

For validation examples, we remove the attention to themselves. In terms of the example above:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}. \tag{4}$$

Information about the state of the current position does still flow through the residual branch, though.

Flexible Encoder Datasets have unequal numbers of input dimensions (features), while PFNs use an encoder layer that accepts fixed dimensional inputs. Here we explain how datasets with different numbers of dimensions can be modelled with a single PFN: We draw the number of dimensions of a dataset during training uniformly at random up to 100. Our encoder changes to accomodate this training and inference with different numbers of features by zero-padding datasets where the number of features k is smaller than the maximum number of features K and scaling these features by $\frac{K}{k}$, s.t. the magnitude stays the same.

E.3 TabPFN Training

We trained our final model for 18 000 steps with a batch size of 512 datasets. That is our TabPFN is trained on 9 216 000 synthetically generated datasets. This training takes 20 hours on 8 GPUs (Nvidia RTX 2080 Ti). Each dataset had a fixed size of 1024 and we split it into training and validation uniformly at random. We generally saw that learning curves tended to flatten after around 10 million datasets and were generally very noisy. Likely, this is because our prior generates a wide variety of different datasets.

E.4 Prior Hyperparameters

The hyperparameters of our prior were chosen based on simplicity and our observations on the validation datasets (such as their class distributions or feature correlation strengths). Also, during algorithm development, we evaluated our models on this set of datasets to decide if our developed methods were correct and working. Since our prior hyperparameters specify distributions and not definite values, they can be chosen over a wide range and resemble the intervals chosen for a random hyperparameter search. The prior distributions we used are given in Table 5.

Table 5: Overview of our prior hyperparameter distribution. For many features we use a Log Uniform distribution with truncated normal noise, which we refer to as TNLU$(h|\check{\mu}, \hat{\mu}, min, round)$. We sample from it by first sampling mean μ and standard deviation σ from $\mu, \sigma \sim$ LogUniform$(\check{\mu}, \hat{\mu})$ and then sampling from the resulting truncated normal distribution $v \sim$ TruncNormal(μ, σ^2). v is rounded to the closest integer, if $round$ is set. The final sampled value then is $h = v + min$.

	Sampling distribution $p(\psi)$				
MLP weight dropout	$0.9 \cdot$ Beta(a, b), where $a, b \sim$ Uniform$(0.1, 5.0)$				
		Choices			
Sample SCM vs BNN	Uniform Choice	{True, False}			
Share Noise mean for nodes	Uniform Choice	{True, False}			
Input feature scaling enabled	Uniform Choice	{True, False}			
Sample y from last MLP layer	Uniform Choice	{True, False}			
MLP Activation Functions	Uniform Choice	{Tanh, Leaky ReLU, ELU, Identity}			
Blockwise Dropout	Uniform Choice	{True, False}			
Keep SCM feature order	Uniform Choice	{True, False}			
Sample feature nodes blockwise	Uniform Choice	{True, False}			
		Max Mean $\hat{\mu}$	Min Mean $\check{\mu}$	$round$	min
MLP #layers	TNLU	6	1	True	2
MLP #hidden nodes per layer	TNLU	130	5	True	4
Gaussian Noise Std.	TNLU	0.3	0.0001	False	0.0
MLP Weights Std.	TNLU	10.0	0.01	False	0.0
SCM #nodes at layer 1	TNLU	12	1	True	1

F Details for Tabular Experiments

Here we provide additional details for the experiments conducted in Section 5 in the main paper.

F.1 HARDWARE SETUP

All evaluations, including the baselines, ran on a compute cluster equipped with Intel(R) Xeon(R) Gold 6242 CPU @ 2.80GHz using 1 CPU with up to 6GB RAM. For evaluation using our TabPFN, we additionally use an RTX 2080 Ti.

F.2 BASELINES

We provide the search space used to tune our baselines in Table 6. For *CatBoost* and *XGBoost*, we used the same ranges as Shwartz-Ziv and Armon (2022) with the following exception: For *CatBoost* we removed the hyperparameter max_size since we could not find it in the official documentation. To be maximally fair to XGBoost, we also tried the search space of quadruple Kaggle grandmaster Bojan Tunguz (Tunguz, 2022), which we adapted slightly by using softmax instead of logistic, as we are in the multi-class setting. XGBoost with this search space performed worse for all considered time budgets than the search space by Shwartz-Ziv and Armon (2022). The search spaces for the KNN, GP and Logistic Regression baselines were designed from scratch and we used the respective implementation from *scikit-learn* (Pedregosa et al., 2011). For *CatBoost* and *AutoSklearn*, we pass the position of categorical features to the classifier (*AutoGluon* automatically detects categorical feature columns). We normalize inputs for Logistic Regression, GP and KNN to the range [0, 1] using MinMax Scaling.

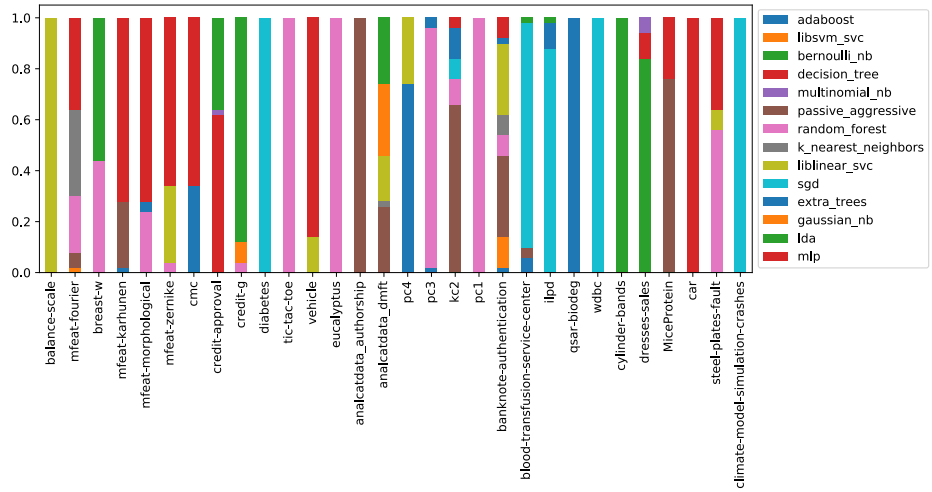

Figure 13: Ensemble weights of classifiers used in AutoSklearn baseline for each dataset. Ensemble weights are averaged across 5 splits for each dataset in the OpenML-CC18 Benchmark after one hour of training.

F.3 USED DATASETS

To construct and evaluate our method, we used the following four sets of datasets.

First, our meta-test set (see Table 7) comprises all datasets in the OpenML-CC18 benchmark suite (Bischl et al., 2021)(available at `OpenML.org`) with at most 2 000 samples, 100 features and 10 classes, which leaves us with 30 datasets that represent small, tabular datasets.

Second, our meta-validation set (see Tables 8 and 9) comprises 150 datasets from `OpenML.org` (Vanschoren et al., 2014). For this, we considered all datasets on `OpenML.org` and applied the following filtering procedure: We dropped all datasets that are in the meta-test set and all datasets with more than 2 000 samples, 100 features or 10 classes. We also manually checked for overlaps and removed datasets where the number of features, classes and samples was identical to a dataset in the meta-test

Table 6: Hyperparameter spaces for baselines. All, except LightGBM, adapted from Shwartz-Ziv and Armon (2022).

baseline	name	type	log	range
LogReg	penalty	cat	(l1, l2, none)	-
	max_iter	int	[50, 500]	-
	fit_intercept	cat	(True, False)	-
	C	float	[e^{-5}, 5]	-
KNN	n_neighbors	int	[1, 16]	-
GP	params_y_scale	float	[0.05, 5.0]	yes
	params_length_scale	float	[0.1, 1.0]	yes
CatBoost	learning_rate	float	[e^{-5}, 1]	yes
	random_strength	int	[1, 20]	-
	l2_leaf_reg	float	[1, 10]	yes
	bagging_temperature	float	[0, 1.0]	yes
	leaf_estimation_iterations	int	[1, 20]	-
	iterations	int	[100, 4000]	-
XGBoost	learning_rate	float	[e^{-7}, 1]	yes
	max_depth	int	[1, 10]	-
	subsample	float	[0.2, 1]	-
	colsample_bytree	float	[0.2, 1]	-
	colsample_bylevel	float	[0.2, 1]	-
	min_child_weight	float	[e^{-16}, e^{5}]	yes
	alpha	float	[e^{-16}, e^{2}]	yes
	lambda	float	[e^{-16}, e^{2}]	yes
	gamma	float	[e^{-16}, e^{2}]	yes
	n_estimators	int	[100, 4000]	-
LightGBM	num_leaves	int	[5, 50]	yes
	max_depth	int	[3, 20]	yes
	learning_rate	float	[e^{-3}, 1]	-
	n_estimators	int	50, 2000	-
	min_child_weight	float	[e^{-5}, e^{4}]	yes
	reg_alpha	float	[0, 1e-1, 1, 2, 5, 7, 10, 50, 100]	yes
	reg_lambda	float	[0, 1e-1, 1, 5, 10, 20, 50, 100]	yes
	subsample	float	[0.2, 0.8]	-

Table 7: Datasets used for the evaluation. These include all 30 datasets in the OpenML-CC18 benchmark suite with at most 2 000 samples, 100 features and 10 classes.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
balance-scale	5	1	625	3	0	49	11
mfeat-fourier	77	1	2000	10	0	200	14
breast-w	10	1	699	2	16	241	15
mfeat-karhunen	65	1	2000	10	0	200	16
mfeat-morphological	7	1	2000	10	0	200	18
mfeat-zernike	48	1	2000	10	0	200	22
cmc	10	8	1473	3	0	333	23
credit-approval	16	10	690	2	67	307	29
credit-g	21	14	1000	2	0	300	31
diabetes	9	1	768	2	0	268	37
tic-tac-toe	10	10	958	2	0	332	50
vehicle	19	1	846	4	0	199	54
eucalyptus	20	6	736	5	448	105	188
analcatdata_auth...	71	1	841	4	0	55	458
analcatdata_dmft	5	5	797	6	0	123	469
pc4	38	1	1458	2	0	178	1049
pc3	38	1	1563	2	0	160	1050
kc2	22	1	522	2	0	107	1063
pc1	22	1	1109	2	0	77	1068
banknote-authenti...	5	1	1372	2	0	610	1462
blood-transfusion-...	5	1	748	2	0	178	1464
ilpd	11	2	583	2	0	167	1480
qsar-biodeg	42	1	1055	2	0	356	1494
wdbc	31	1	569	2	0	212	1510
cylinder-bands	40	22	540	2	999	228	6332
dresses-sales	13	12	500	2	835	210	23381
MiceProtein	82	5	1080	8	1396	105	40966
car	7	7	1728	4	0	65	40975
steel-plates-fault	28	1	1941	7	0	55	40982
climate-model-simu...	21	1	540	2	0	46	40994

set. Furthermore, we manually dropped FOREX (since it is a time series dataset) and artificially-created datasets, such as the Univ and Friedman datasets. The remaining meta-validation set then contains 150 datasets. This meta-validation set was used to guide the development of our prior hyperparameters as described in Appendix E.4.

Third, a subset of 5 datasets from the OpenML-AutoML Benchmark, comprises all datasets from the OpenML-AutoML Benchmark with at most 1 111 samples, 100 features and 10 classes. This is given in Table 10. Due to the 10-fold cross-validation in the OpenML-AutoML Benchmark this is identical to the setup for our meta-test and meta-validation datasets above, where we used up to 2 000 samples split 50-50 into training and test.

Fourth, our meta-generalization set, described in Appendix F.4, comprises 18 larger datasets from the OpenML AutoML Benchmark.

F.4 Model Generalization

For testing the TabPFN performance on longer sequences, as described in Section 10, we used a set of 18 datasets from the OpenML AutoML Benchmark that contain at least 10 000 samples. The list of datasets used can be found in Table 11. For this evaluation, datasets with more than 100 features are limited to the first 100 features. When more than 10 classes are contained in the datasets, samples with any but the first 10 classes are discarded.

F.5 Details on Time Comparisons

Time comparisons refer to combined fitting, tuning and prediction; see Table 2 for the times split into tuning/fitting and prediction. The time taken for each baseline and TabPFN does not include the one-time cost of development of each method. Thus, for AutoML baselines meta-learning cost was not included (e.g. Auto-Sklearn pipelines meta-learning, which involved running hyper-parameter search for 24 hours on 140 datasets (= 3360 CPU hours) (Feurer et al., 2021)). For GBDT methods the manual time that went into defining suitable hyperparameter spaces and the manual crafting of algorithms that perform well on tabular datasets is hard to measure and was not included. For TabPFN the prior-fitting phase (which is part of our algorithm development: i.e. developing ideas, writing code, trying ideas) is not included. This is fair to all methods, as these costs are not on the user side and are amortized over time.

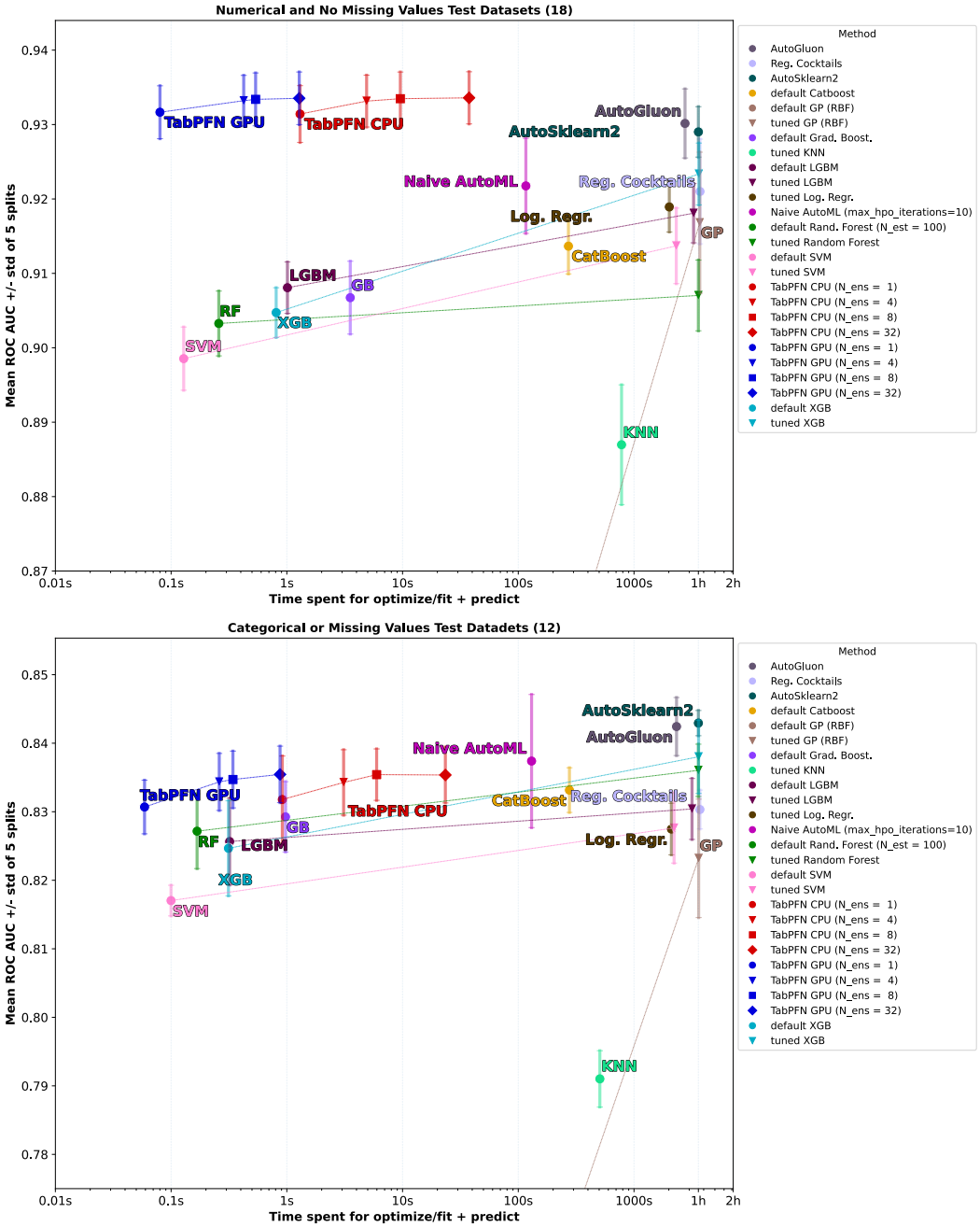

Figure 14: ROC AUC comparison on the OpenML-CC18 Benchmark. Baselines were tuned for one hour or until 10000 configurations were exhausted (Log. Reg and KNN).

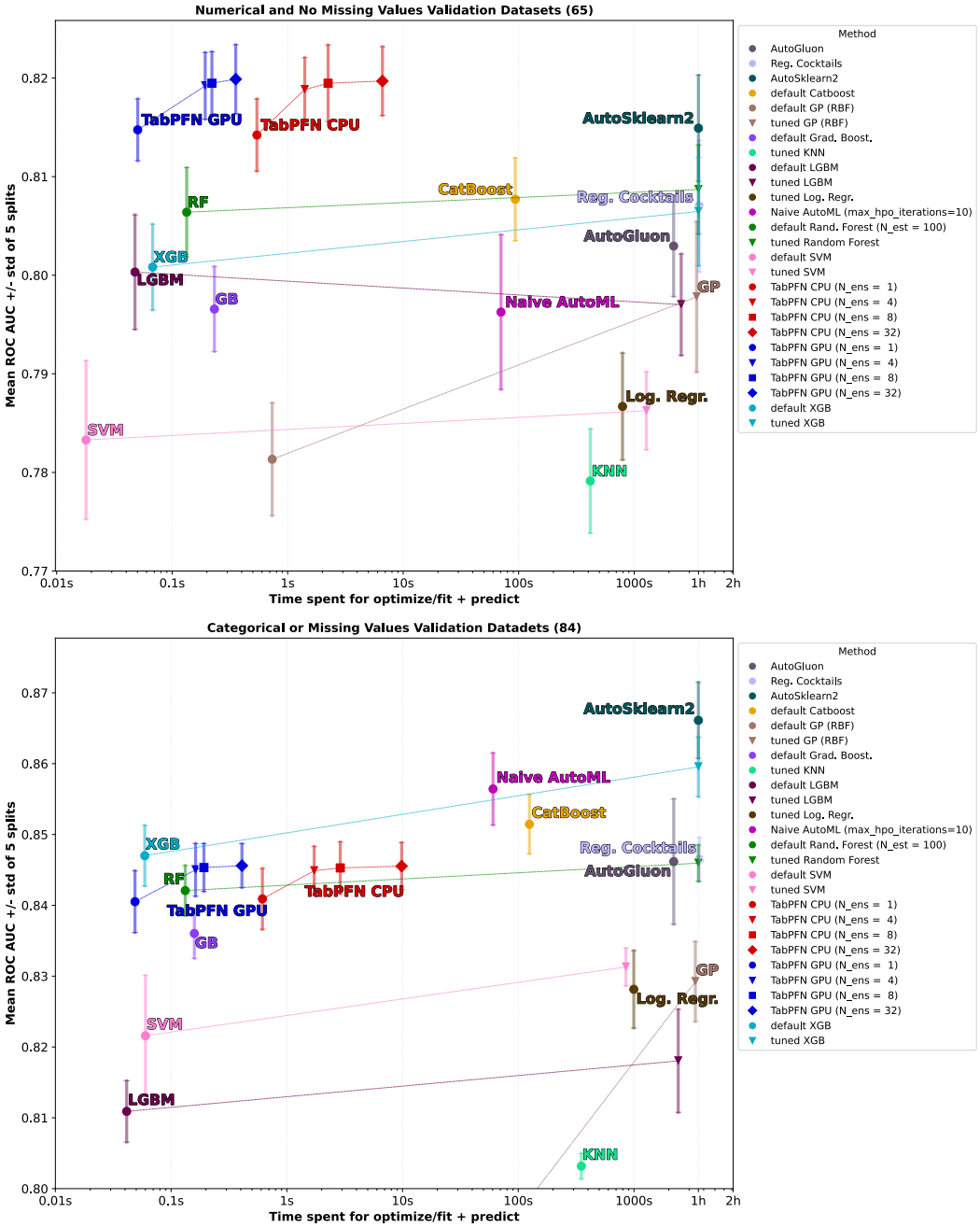

Figure 15: ROC AUC comparison on 149 validation datasets (see Table 8). Baselines were tuned for one hour or until 10 000 configurations were exhausted (Log. Reg and KNN).

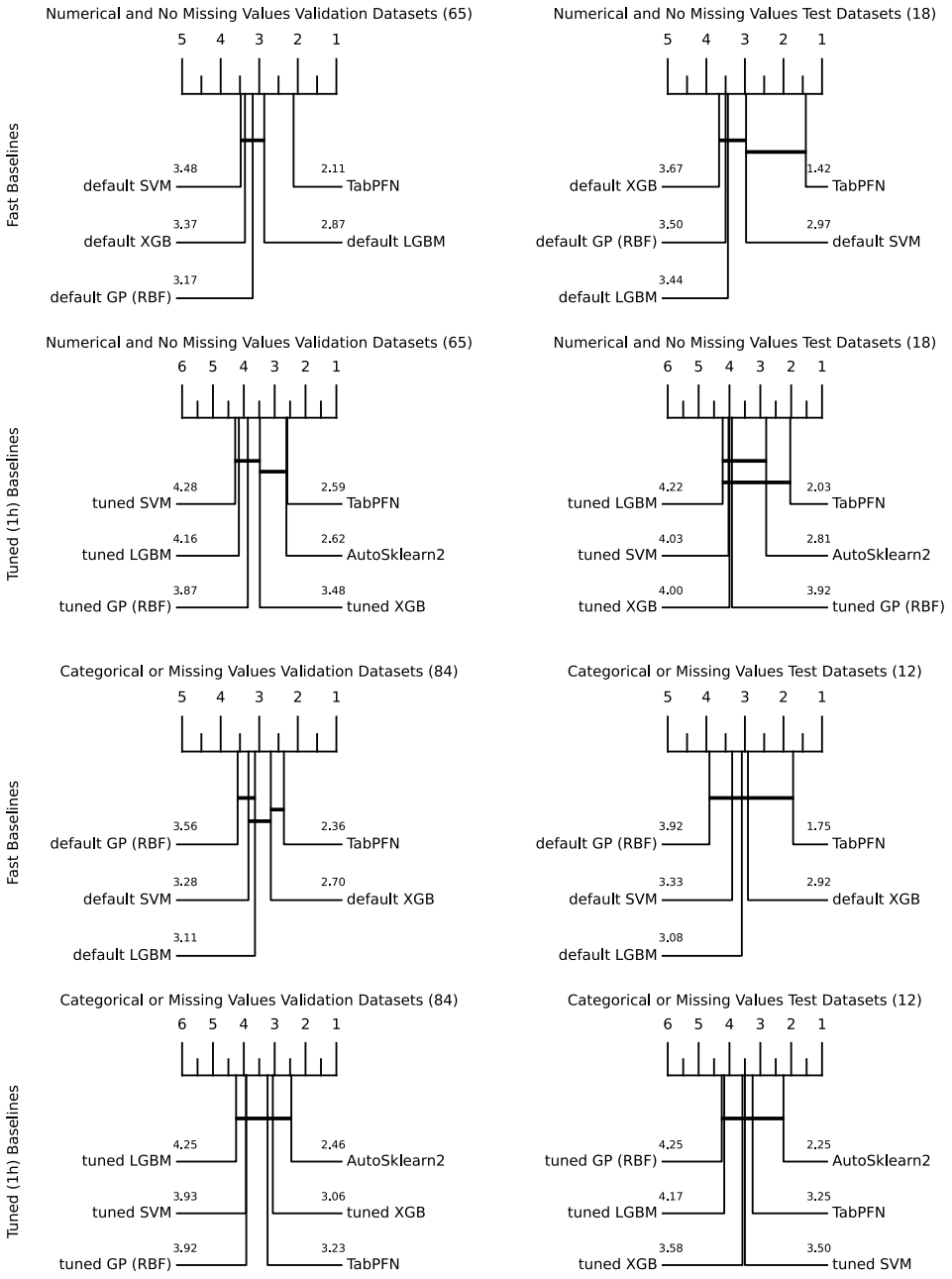

Figure 16: Critical difference plots on average ranks with a Wilcoxon significance analysis. We show plots for both our test set (OpenML-CC18) and our validation set. We split each into a subset of purely numerical datasets without missing values and the rest. We compare to two sets of baselines. i) *Fast Baselines* Baselines that finish tuning, training and prediction in less than 30 seconds on average (the TabPFN on CPU is within this bound). ii) *Tuned (1h) Baselines* Stronger baselines with one hour of budget for tuning, training and prediction (TabPFN still requires less than 30 seconds on average with the same hardware). We can see that TabPFN performs similar on the large validation and the test set. Additionally, we see that TabPFN is much stronger on purely numerical datasets.
A critical difference analysis comparing tuned XGB and TabPFN (ensemble of 32) on numerical datasets without missing values shows a statistically significant improvement of TabPFN on test ($p = 1.6\%$) and validation datasets ($p = 2.2\%$). We observe statistical significance only when comparing tuned XGB to TabPFN without any other methods, but not when comparing multiple methods at once. This is, because in the above plots statistical corrections are made to avoid multiple testing.

Table 8: Meta-Datasets used for developing the prior.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
breast-cancer	10	10	286	2	9	85	13
colic	27	20	368	2	1927	136	25
dermatology	35	34	366	6	8	20	35
sonar	61	1	208	2	0	97	40
glass	10	1	214	6	0	9	41
haberman	4	2	306	2	0	81	43
tae	6	3	151	3	0	49	48
heart-c	14	8	303	2	7	138	49
heart-h	14	8	294	2	782	106	51
heart-statlog	14	1	270	2	0	120	53
hepatitis	20	14	155	2	167	32	55
vote	17	17	435	2	392	168	56
ionosphere	35	1	351	2	0	126	59
iris	5	1	150	3	0	50	61
wine	14	1	178	3	0	48	187
flags	29	27	194	8	0	4	285
hayes-roth	5	1	160	3	0	31	329
monks-problems-1	7	7	556	2	0	278	333
monks-problems-2	7	7	601	2	0	206	334
monks-problems-3	7	7	554	2	0	266	335
SPECT	23	23	267	2	0	55	336
SPECTF	45	1	349	2	0	95	337
grub-damage	9	7	155	4	0	19	338
synthetic_control	61	1	600	6	0	100	377
prnn_crabs	8	2	200	2	0	100	446
analcatdata_lawsuit	5	2	264	2	0	19	450
irish	6	4	500	2	32	222	451
analcatdata_broadwaymult	8	5	285	7	27	21	452
analcatdata_reviewer	8	8	379	4	1418	54	460
backache	32	27	180	2	0	25	463
prnn_synth	3	1	250	2	0	125	464
schizo	15	3	340	2	834	163	466
profb	10	5	672	2	1200	224	470
analcatdata_germangss	6	5	400	4	0	100	475
biomed	9	2	209	2	15	75	481
rmftsa_sleepdata	3	1	1024	4	0	94	679
diggle_table_a2	9	1	310	9	0	18	694
rmftsa_ladata	11	1	508	2	0	222	717
pwLinear	11	1	200	2	0	97	721
analcatdata_vineyard	4	2	468	2	0	208	724
machine_cpu	7	1	209	2	0	56	733
pharynx	11	10	195	2	2	74	738
auto_price	16	2	159	2	0	54	745
servo	5	5	167	2	0	38	747
analcatdata_wildcat	6	3	163	2	0	47	748
pm10	8	1	500	2	0	246	750
wisconsin	33	1	194	2	0	90	753
autoPrice	16	1	159	2	0	54	756
meta	22	3	528	2	504	54	757
analcatdata_apnea3	4	3	450	2	0	55	764
analcatdata_apnea2	4	3	475	2	0	64	765
analcatdata_apnea1	4	3	475	2	0	61	767
disclosure_x_bias	4	1	662	2	0	317	774
bodyfat	15	1	252	2	0	124	778
cleveland	14	8	303	2	6	139	786
triazines	61	1	186	2	0	77	788
disclosure_x_tampered	4	1	662	2	0	327	795
cpu	8	2	209	2	0	53	796
cholesterol	14	8	303	2	6	137	798
chscase_funds	3	1	185	2	0	87	801
pbcseq	19	7	1945	2	1133	972	802
pbc	19	9	418	2	1239	188	810
rmftsa_ctoarrivals	3	2	264	2	0	101	811
chscase_vine2	3	1	468	2	0	212	814
chatfield_4	13	1	235	2	0	93	820
boston_corrected	21	4	506	2	0	223	825
sensory	12	12	576	2	0	239	826
disclosure_x_noise	4	1	662	2	0	329	827
autoMpg	8	4	398	2	6	189	831
kdd_el_nino-small	9	3	782	2	466	274	839
autoHorse	26	9	205	2	57	83	840
stock	10	1	950	2	0	462	841
breastTumor	10	9	286	2	9	120	844
analcatdata_gsssexsurvey	10	6	159	2	6	35	852
boston	14	2	506	2	0	209	853
fishcatch	8	3	158	2	87	63	854
vinnie	3	1	380	2	0	185	860
mu284	11	1	284	2	0	142	880
no2	8	1	500	2	0	249	886
chscase_geyser1	3	1	222	2	0	88	895
chscase_census6	7	1	400	2	0	165	900
chscase_census5	8	1	400	2	0	193	906
chscase_census4	8	1	400	2	0	194	907
chscase_census3	8	1	400	2	0	192	908
chscase_census2	8	1	400	2	0	197	909
plasma_retinol	14	4	315	2	0	133	915
visualizing_galaxy	5	1	323	2	0	148	925
colleges_usnews	34	2	1302	2	7830	614	930

Table 9: Meta-Datasets used for developing the prior (continued).

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
disclosure_z	4	1	662	2	0	314	931
socmob	6	5	1156	2	0	256	934
chscase_whale	9	1	228	2	20	111	939
water-treatment	37	16	527	2	542	80	940
lowbwt	10	8	189	2	0	90	941
arsenic-female-bladder	5	2	559	2	0	80	949
analcatdata_halloffame	17	2	1340	2	20	125	966
analcatdata_birthday	4	3	365	2	30	53	968
analcatdata_draft	5	3	366	2	1	32	984
collins	23	3	500	2	0	80	987
prnn_fglass	10	1	214	2	0	76	996
jEdit_4.2_4.3	9	1	369	2	0	165	1048
mc2	40	1	161	2	0	52	1054
mw1	38	1	403	2	0	31	1071
jEdit_4.0_4.2	9	1	274	2	0	134	1073
PopularKids	11	5	478	3	0	90	1100
teachingAssistant	7	5	151	3	0	49	1115
lungcancer_GSE31210	24	3	226	2	0	35	1412
MegaWatt1	38	1	253	2	0	27	1442
PizzaCutter1	38	1	661	2	0	52	1443
PizzaCutter3	38	1	1043	2	0	127	1444
CostaMadre1	38	1	296	2	0	38	1446
CastMetal1	38	1	327	2	0	42	1447
KnuggetChase3	40	1	194	2	0	36	1448
PieChart1	38	1	705	2	0	61	1451
PieChart3	38	1	1077	2	0	134	1453
parkinsons	23	1	195	2	0	48	1488
planning-relax	13	1	182	2	0	52	1490
qualitative-bankruptcy	7	7	250	2	0	107	1495
sa-heart	10	2	462	2	0	160	1498
seeds	8	1	210	3	0	70	1499
thoracic-surgery	17	14	470	2	0	70	1506
user-knowledge	6	1	403	5	0	24	1508
wholesale-customers	9	2	440	2	0	142	1511
heart-long-beach	14	1	200	5	0	10	1512
robot-failures-lp5	91	1	164	5	0	21	1520
vertebra-column	7	1	310	3	0	60	1523
Smartphone-Based...	68	2	180	6	0	30	4153
breast-cancer-...	10	10	277	2	0	81	23499
LED-display-...	8	1	500	10	0	37	40496
GAMETES_Epistasis...	21	21	1600	2	0	800	40646
calendarDOW	33	21	399	5	0	44	40663
corral	7	7	160	2	0	70	40669
mofn-3-7-10	11	11	1324	2	0	292	40680
thyroid-new	6	1	215	3	0	30	40682
solar-flare	13	13	315	5	0	21	40686
threeOf9	10	10	512	2	0	238	40690
xd6	10	10	973	2	0	322	40693
tokyo1	45	3	959	2	0	346	40705
parity5_plus_5	11	11	1124	2	0	557	40706
cleve	14	9	303	2	0	138	40710
cleveland-nominal	8	8	303	5	0	13	40711
Australian	15	9	690	2	0	307	40981
DiabeticMellitus	98	1	281	2	2	99	41430
conference_attendance	7	7	246	2	0	31	41538
CPMP-2015-...	23	1	527	4	0	78	41919
TuningSVMs	81	1	156	2	0	54	41976
regime_alimentaire	20	17	202	2	17	41	42172
iris-example	5	1	150	3	0	50	42261
Touch2	11	1	265	8	0	27	42544
penguins	7	3	344	3	18	68	42585
titanic	8	5	891	2	689	342	42638

Table 10: Datasets used for the evaluation in the OpenML-AutoML Benchmark. These include all datasets with at most 1 111 samples, 100 features and 10 classes.

Name	#Feat.	#Cat.	#Inst.	Class Size	#NaNs	Minor. Class Size	OpenML ID
credit-g	21	14	1000	2	0	300	31
vehicle	19	1	846	4	0	199	54
wine	14	1	178	2	0	71	973
blood-transfusion-service-center	5	1	748	2	0	178	1464
Australian	15	9	690	2	0	307	40981

Table 11: Evaluation datasets for model generalization experiments.

Name	#Feat.	#Cat.	#Inst.	#Class.	#NaNs	Minor. Class Size	OpenML ID
KDDCup09_appetency	231	39	50000	2	8024152	890	1111
airlines	8	5	539383	2	0	240264	1169
bank-marketing	17	10	45211	2	0	5289	1461
nomao	119	30	34465	2	0	9844	1486
adult	15	9	48842	2	6465	11687	1590
covertype	55	45	581012	7	0	2747	1596
numerai28.6	22	1	96320	2	0	47662	23517
connect-4	43	43	67557	3	0	6449	40668
jungle_chess_2pcs..	7	1	44819	3	0	4335	41027
APSFailure	171	1	76000	2	1078695	1375	41138
albert	79	53	425240	2	2734000	212620	41147
MiniBooNE	51	1	130064	2	0	36499	41150
guillermo	4297	1	20000	2	0	8003	41159
riccardo	4297	1	20000	2	0	5000	41161
volkert	181	1	58310	10	0	1361	41166
dionis	61	1	416188	355	0	878	41167
jannis	55	1	83733	4	0	1687	41168
helena	28	1	65196	100	0	111	41169