TABPFN: A TRANSFORMER THAT SOLVES SMALL TABULAR CLASSIFICATION PROBLEMS IN A SECOND

Noah Hollmann[*,1,2] **Samuel Müller**[*,1] **Katharina Eggensperger**[1] **Frank Hutter**[1,3]
[1] University of Freiburg, [2] Charité University Medicine Berlin
[3] Bosch Center for Artificial Intelligence [*] Equal contribution.
Correspondance to noah.hollmann@charite.de & muellesa@cs.uni-freiburg.de

ABSTRACT

We present TabPFN, a trained Transformer that can do supervised classification for small tabular datasets in *less than a second*, needs no hyperparameter tuning and is competitive with state-of-the-art classification methods. TabPFN is fully entailed in the weights of our network, which accepts training and test samples as a set-valued input and yields predictions for the entire test set in a single forward pass. TabPFN is a Prior-Data Fitted Network (PFN) and is trained offline once, to approximate Bayesian inference on synthetic datasets drawn from our prior. This prior incorporates ideas from causal reasoning: It entails a large space of structural causal models with a preference for simple structures. On the 18 datasets in the OpenML-CC18 suite that contain up to 1 000 training data points, up to 100 purely numerical features without missing values, and up to 10 classes, we show that our method clearly outperforms boosted trees and performs on par with complex state-of-the-art AutoML systems with up to 70× speedup. This increases to a 3 200× speedup when a GPU is available. We also validate these results on an additional 67 small numerical datasets from OpenML. We provide all our code, the trained TabPFN, an interactive browser demo and a Colab notebook at https://github.com/automl/TabPFN.

1 INTRODUCTION

Tabular data has long been overlooked by deep learning research, despite being the most common data type in real-world machine learning (ML) applications (Chui et al., 2018). While deep learning methods excel on many ML applications, tabular data classification problems are still dominated by Gradient-Boosted Decision Trees (GBDT; Friedman, 2001), largely due to their short training time and robustness (Shwartz-Ziv and Armon, 2022).

We propose a radical change to how tabular classification is done. We do *not* fit a new model from scratch to the training portion of a new dataset. Instead, we replace this step by performing a single forward pass with a large Transformer that has been pre-trained to solve artificially generated classification tasks from a tabular dataset prior.

Our method builds on Prior-Data Fitted Networks (PFNs; Müller et al., 2022; see Section 2), which learn the training and prediction algorithm itself. PFNs approximate Bayesian inference given any prior one can sample from and approximate the posterior predictive distribution (PPD) directly. While inductive biases in NNs and GBDTs depend on them being efficient to implement (e.g., through L_2 regularization, dropout (Srivastava et al., 2014) or limited tree-depth), in PFNs, one can simply design a dataset-generating algorithm that encodes the desired prior. This fundamentally changes the way we can design learning algorithms.

We design a prior (see Section 4) based on Bayesian Neural Networks (BNNs; Neal, 1996; Gal, 2016) and Structural Causal Models (SCMs; Pearl, 2009; Peters et al., 2017) to model complex feature dependencies and potential causal mechanisms underlying tabular data. Our prior also takes ideas from Occam's razor: explanations based on simpler SCMs and BNNs (with fewer parameters) have a higher likelihood. Our prior is defined via parametric distributions, e.g., a log-scaled uniform distribution for the average number of nodes in data-generating SCMs. The resulting PPD implicitly

models uncertainty over all possible data-generating mechanisms, weighting them by their likelihood given the data and their prior probability. Thus, the PPD corresponds to an infinitely large ensemble of data-generating mechanisms, i.e., instantiations of SCMs and BNNs. We learn to approximate this complex PPD in a single forward-pass, requiring no cross-validation or model selection.

Our **key contribution** is to introduce the *TabPFN* (see Section 3), a single Transformer that has been pre-trained to approximate probabilistic inference for the novel prior above in a single forward pass, and has thus learned to solve novel small numerical tabular classification tasks ($\leq 1\,000$ training examples, 100 purely numerical features without missing values and 10 classes) in *less than a second* yielding state-of-the-art performance.

To substantiate this claim, we qualitatively and quantitatively analyze the behavior and performance of our TabPFN on different tasks and compare it to previous approaches for tabular classification on 18 small, numerical datasets (see Section 5). Quantitatively, the TabPFN yields much better performance than any individual "base-level" classification algorithm, such as gradient-boosting via XGBoost (Chen and Guestrin, 2016), LightGBM (Ke et al., 2017) and CatBoost (Prokhorenkova et al., 2018), and in less than a second yields performance competitive to what the best available AutoML frameworks (Erickson et al., 2020, Feurer et al., 2021) achieve in one hour. Our in-depth qualitative analysis shows the TabPFN's predictions to be smooth and intuitive. Yet, its errors are quite uncorrelated to the errors of existing approaches, allowing additional performance improvements by ensembling. We also validate TabPFN's performance on an additional 67 datasets from OpenML Vanschoren et al. (2014).

We expect the revolutionary character of our claims to be met with initial skepticism and thus open-source all our code and the pre-trained TabPFN for scrutinization by the community, along with a scikit-learn-like interface, a Colab notebook and two online demos. Please see Section 8 for the links.

2 Background on Prior-Data Fitted Networks

First, we summarize how PFNs work; we refer to Müller et al. (2022) for the full details.

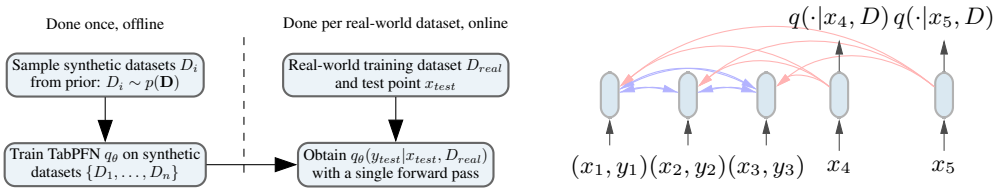
The Posterior Predictive Distribution for Supervised Learning In the Bayesian framework for supervised learning, the prior defines a space of hypotheses Φ on the relationship of a set of inputs x to the output labels y. Each hypothesis $\phi \in \Phi$ can be seen as a mechanism that generates a data distribution from which we can draw samples forming a dataset. For example, given a prior based on structural causal models, Φ is the space of structural causal models, a hypothesis ϕ is one specific SCM, and a dataset comprises samples generated through the SCM. In practice, a dataset comprises training data with observed labels and test data where labels are missing or held out to assess predictive performance. The PPD for a test sample x_{test} specifies the distribution of its label $p(\cdot|x_{test}, D_{train})$, which is conditioned on the set of training samples $D_{train} := \{(x_1, y_1), \ldots, (x_n, y_n)\}$. The PPD can be obtained by integration over the space of hypotheses Φ, where the weight of a hypothesis $\phi \in \Phi$ is determined by its prior probability $p(\phi)$ and the likelihood $p(D|\phi)$ of the data D given ϕ:

$$p(y|x, D) \propto \int_{\Phi} p(y|x, \phi)p(D|\phi)p(\phi)d\phi. \tag{1}$$

Synthetic Prior-fitting Prior-fitting is the training of a PFN to approximate the PPD and thus do Bayesian prediction. We implement it with a prior which is specified by a prior sampling scheme of the form $p(D) = \mathbb{E}_{\phi \sim p(\phi)}[p(D|\phi)]$, which first samples hypotheses (generating mechanisms) with $\phi \sim p(\phi)$ and then synthetic datasets with $D \sim p(D|\phi)$. We repeatedly sample such synthetic datasets $D := (x_i, y_i)_{i \in \{1,\ldots,n\}}$ and optimize the PFN's parameters θ to make predictions for $D_{test} \subset D$, conditioned on the rest of the dataset $D_{train} = D \setminus D_{test}$. The loss of the PFN training thus is the cross-entropy on held-out examples of synthetic datasets. For a single test point $\{(x_{test}, y_{test})\} = D_{test}$, the training loss can be written as

$$\mathcal{L}_{PFN} = \mathop{\mathbb{E}}_{(\{(x_{test}, y_{test})\} \cup D_{train}) \sim p(D)} [-\log q_\theta(y_{test}|x_{test}, D_{train})]. \tag{2}$$

As shown by Müller et al. (2022), minimizing this loss approximates Bayesian prediction. We visualize this in Figure 1a and detail the full training setup in Algorithm 1 in the appendix. Crucially, this *synthetic prior-fitting* phase is performed only once for a given prior $p(D)$ as part of algorithm development, learning to do *real-world inference* on new datasets.



(a) Prior-fitting and inference (b) Architecture and attention mechanism

Figure 1: Left (a): The PFN learns to approximate the PPD of a given prior in the offline stage to yield predictions on a new dataset in a single forward pass in the online stage. Right (b): Training samples $\{(x_1, y_1), \ldots, (x_3, y_3)\}$ are transformed to 3 tokens, which attend to each other; test samples x_4 and x_5 attend only to the training samples. Plots based on Müller et al. (2022).

Real-World Inference During inference, the trained model is applied to unseen real-world datasets. For a novel dataset with training samples D_{train} and test features x_{test}, feeding $\langle D_{train}, x_{test} \rangle$ as an input to the model trained above yields the PPD $q_\theta(y|x_{test}, D_{train})$ in a single forward-pass. The PPD class probabilities are then used as predictions for our real-world task. Thus, PFNs perform training and prediction in one step (similar to prediction with Gaussian Processes) and do not use gradient-based learning on data seen at inference time.

Architecture PFNs rely on a Transformer (Vaswani et al., 2017) that encodes each feature vector and label as a token, allowing token representations to attend to each other, as depicted in Figure 1b. They accept a variable length training set D_{train} of feature and label vectors (treated as a set-valued input to exploit permutation invariance) as well as a variable length query set of feature vectors $x_{test} = \{x_{(test,1)}, \ldots, x_{(test,m)}\}$ and return estimates of the PPD for each query.

Tabular Data Amongst other experiments, Müller et al. (2022) demonstrated PPD approximation with PFNs on binary classification for very small, balanced tabular datasets with 30 training examples. Here, we substantially improve performance and scale up to realistically-sized small datasets with up to 2 000 data points and up to 10 imbalanced classes. We discuss detailed changes in Appendix C.2.6.

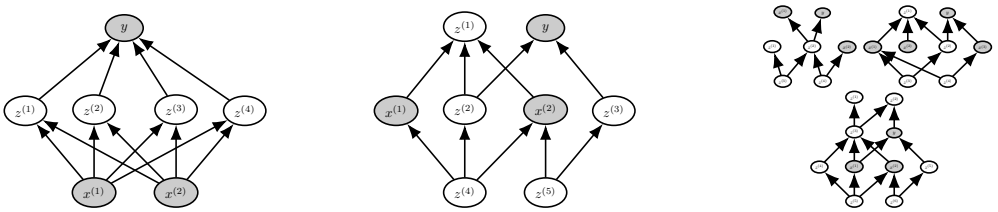
3 The TabPFN: A PFN fitted on a new prior for tabular data

Our TabPFN is a Prior-data Fitted Network (PFN, see Section 2) that is fitted on data sampled from a novel prior for tabular data we introduce in Section 4. We modify the original PFN architecture (Müller et al., 2022) in two ways. i) We make slight modifications to the attention masks, yielding shorter inference times. ii) Additionally, we enable our model to work on datasets with different numbers of features by zero-padding. These architectural enhancements alongside our main contributions compared to Müller et al. (2022) are detailed in Appendix E.2.

In the prior-fitting phase, we train the TabPFN once on samples from the prior described in Section 4. To be more precise, we trained a 12-layer Transformer for 18 000 batches of 512 synthetically generated datasets each, which required a total of 20 hours on one machine with 8 GPUs (Nvidia RTX 2080 Ti). This yielded a single network that is used for all our evaluations. While this training step is moderately expensive, it is done offline, in advance, and only once for the TabPFN, as part of our algorithm development. The same TabPFN model is used for all experiments in this paper. Full details about our TabPFN training are given in Appendix E.

During inference, the TabPFN approximates the PPD for our dataset prior, i.e., it approximates the marginal predictions across our spaces of SCMs and BNNs (see Section 4), including a bias towards simple and causal explanations for the data. In our experiments, we present predictions for a single forward pass of our TabPFN, as well as predictions that ensemble 32 forward passes of datasets augmented with shuffled feature columns and permuted class labels.[1]

[1] Since we already perform these shuffling and permutation operations while training the TabPFN, it can in principle learn a representation that are invariant to them. We expect that, using a longer training time or a larger Transformer, TabPFN would learn these invariances and make these permutations obsolete.



(a) A BNN (b) An SCM (c) SCMs sampled from the prior

Figure 2: Overview of graphs generating data in our prior. Inputs x are mapped to the output y through unobserved nodes z. Plots based on Müller et al. (2022).

4 A Prior for Tabular Data

The performance of our method crucially depends on the specification of a suitable prior, as the PFN approximates the PPD for this prior. Section 4.1 outlines a fundamental technique for our prior: we use distributions instead of point-estimates for almost all of our prior's hyperparameters. Section 4.2 motivates simplicity in our prior, while Sections 4.3 and 4.4 describe how we use structural causal models (SCMs) and Bayesian Neural Networks (BNNs) as fundamental mechanisms to generate diverse data in our prior. Since our SCM and BNN priors only yield regression tasks, we show how to convert them to classification tasks in Section 4.5. We describe additional refinements to our prior to reflect peculiarities of tabular data (correlated and categorical features, exponentially scaled data and missing values) better in Appendix C.2.

4.1 Fundamentally Probabilistic Models

Fitting a model typically requires finding suitable hyperparameters, e.g., the embedding size, number of layers and activation function for NNs. Commonly, resource-intensive searches need to be employed to find suitable hyperparameters (Zoph and Le, 2017; Feurer and Hutter, 2019). The result of these searches, though, is only a point estimate of the hyperparameter choice. Ensembling over multiple architectures and hyperparameter settings can yield a rough approximation to a distribution over these hyperparameters and has been shown to improve performance (Zaidi et al., 2021; Wenzel et al., 2020). This, however, scales linearly in cost with the number of choices considered.

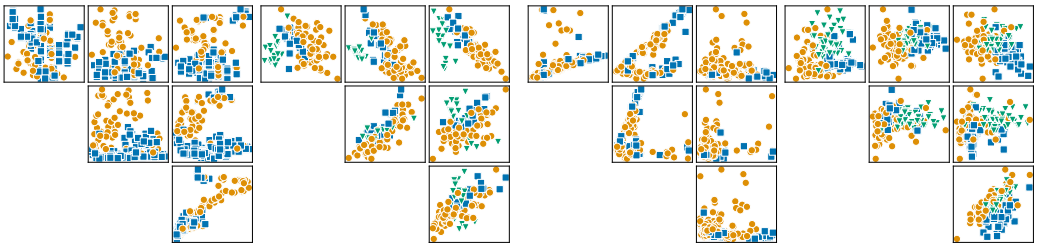
In contrast, PFNs allow us to be fully Bayesian about our prior's hyperparameters. By defining a probability distribution over the space of hyperparameters in the prior, such as BNN architectures, the PPD approximated by our TabPFN jointly integrates over this space and the respective model weights. We extend this approach to a mixture not only over hyperparameters but distinct priors: we mix a BNN and an SCM prior, each of which again entails a mixture of architectures and hyperparameters.

4.2 Simplicity

We base our priors on a notion of simplicity, such as stated by Occam's Razor or the Speed Prior (Schmidhuber, 2002). When considering competing hypotheses, the simpler one is to be preferred. Work in cognitive science has also uncovered this preference for simple explanations in human thinking (Wojtowicz and DeDeo, 2020). Any notion of simplicity, however, depends on choosing a particular criterion that defines simplicity. In the following, we introduce priors based on SCMs and BNNs, in which simplicity materializes as graphs with few nodes and parameters.

4.3 SCM Prior

It has been demonstrated that causal knowledge can facilitate various ML tasks, including semi-supervised learning, transfer learning and out-of-distribution generalization (Schölkopf et al., 2012; Janzing, 2020; Rothenhäusler et al., 2018). Tabular data often exhibits causal relationships between columns, and causal mechanisms have been shown to be a strong prior in human reasoning (Waldmann and Hagmayer, 2013; Wojtowicz and DeDeo, 2020). Thus, we base our TabPFN prior on SCMs that model causal relationships (Pearl, 2009; Peters et al., 2017). An SCM consists of a collection $Z := (\{z_1, \ldots, z_k\})$ of structural assignments (called mechanisms): $z_i = f_i(z_{\mathrm{PA}_\mathcal{G}(i)}, \epsilon_i)$, where



(a) Synthetic datasets (b) Actual datasets

Figure 3: Each sub-plot shows the combination of two features, each dot represents a sample, color indicates the class label. (a) Two synthetic datasets generated by our causal tabular data prior. Numeric SCM outputs are mapped to classes as described in Section 4.5. (b) Two datasets from our validation datasets: Parkinsons (Left) and Wine (Right).

$\text{PA}_{\mathcal{G}}(i)$ is the set of parents of z_i (its direct causes) in an underlying DAG $\mathcal{G}$ (the causal graph), f_i is a (potentially non-linear) deterministic function and ϵ_i is a noise variable. Causal relationships in $\mathcal{G}$ are represented by directed edges pointing from causes to effects and each mechanism z_i is assigned to a node in $\mathcal{G}$, as visualized in Figure 2.

Predictions based on ideas from causal reasoning Previous works have applied causal reasoning to predict observations on unseen data by using causal inference, a method which seeks to identify causal relations between the components of a system by the use of interventions and observational data (Pearl, 2010; Pearl and Mackenzie, 2018; Lin et al., 2021). The predicted causal representations are then used to make observational predictions on novel samples or to provide explainability. Most existing work focuses on determining a single causal graph to use for downstream prediction, which can be problematic since most kinds of SCMs are non-identifiable without interventional data, and the number of compatible DAGs explodes due to the combinatorial nature of the space of DAGs. In contrast, our TabPFN considers a broad family of SCMs and their respective weights but without any causal guarantees. We skip any explicit graph representation in our inference step and approximate the PPD directly. Thus, we do not perform causal inference but solve the downstream prediction task directly. This implicit assumption of SCM-like processes generating our data can be explained in Pearl's "ladder of causation", an abstraction of inference categories, where each higher rung represents a more involved notion of inference (Pearl and Mackenzie, 2018). At the lowest rung lies association, which includes most of ML. At the second rung, we find predicting the effect of interventions, i.e., what happens when we influence features directly. Our work can be considered as "rung 1.5", similar to Kyono et al. (2020; 2021): we do not perform causal reasoning, but make association-based predictions on observational data assuming SCMs model common datasets well. In Figure 8, we experimentally show that TabPFN's predictions indeed align with simple SCM hypotheses.

Defining a prior based on causal models To create a PFN prior based on SCMs, we have to define a sampling procedure that creates supervised learning tasks (i.e., datasets). Here, each dataset is based on one randomly-sampled SCM (including the DAG structure and deterministic functions f_i). Given an SCM, we sample a set z_X of nodes in the causal graph $\mathcal{G}$, one for each feature in our synthetic dataset, as well as one node z_y from $\mathcal{G}$. These nodes are observed nodes: values of z_X will be included in the set of features, while values from z_y will act as targets. For each such SCM and list of nodes z_X and z_y, n samples are generated by sampling all noise variables in the SCM n times, propagating these through the graph and retrieving the values at the nodes z_X and z_y for all n samples. Figure 2b depicts an SCM with observed feature- and target-nodes in grey. The resulting features and targets are correlated through the generating DAG structure. This leads to features conditionally dependent through forward and backward causation, i.e., targets might be a cause or an effect of features. In Figure 3a, we show scatter plots of samples generated by two distinct SCMs, demonstrating the diversity in the space of datasets our prior can model.

In this work, we instantiate a large subfamily of DAGs and deterministic functions f_i to build SCMs described in Appendix C.1. Since efficient sampling is the only requirement we have, the instantiated subfamily is very general, including multiple activation functions and noise distributions.

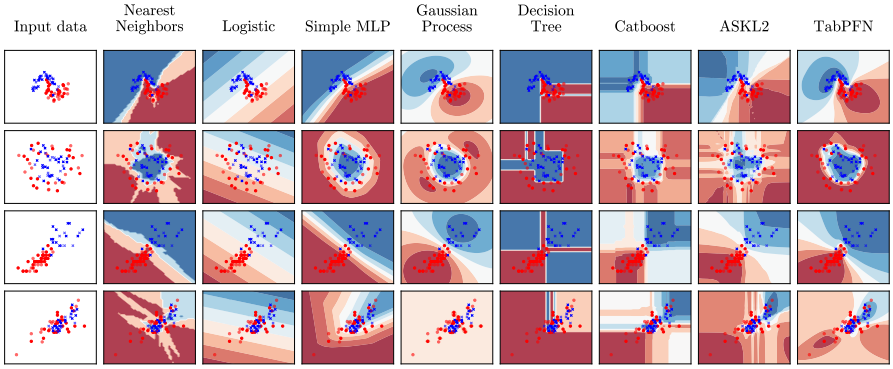


Figure 4: Decision boundaries on toy datasets generated with *scikit-learn* (Pedregosa et al., 2011).

4.4 BNN Prior

We also consider a BNN prior as introduced by Müller et al. (2022) and mix it with the SCM prior described above by randomly sampling datasets during PFN training from either one or the other prior with equal probability. To sample a dataset from the BNN prior, we first sample an NN architecture and its weights. Then, for each data point in the to-be-generated dataset, we sample an input x, feed it through the BNN with sampled noise variables and use the output y as a target (see Figure 2a).We show an experimental ablation of the BNN prior based on Müller et al. (2022) to our final mixture of both SCM-based and BNN-based priors prior in Appendix B.4.

4.5 Multi-class Prediction

So far, the described priors return scalar labels. In order to generate synthetic classification labels for imbalanced multi-class datasets, we need to transform our scalar labels $\hat{y}$ to discrete class labels y. We do so by splitting the values of $\hat{y}$ into intervals that map to class labels:

i) We sample the number of classes $N_c \sim p(N_c)$, where $p(N_c)$ is a distribution over integers.
ii) We sample $N_c - 1$ class bounds B_i randomly from the set of continuous targets $\hat{y}$.
iii) We map each scalar label $\hat{y}_i$ to the index of the unique interval that contains it: $y_i \leftarrow \sum_j [B_j < \hat{y}_i]$, where $[\cdot]$ is the indicator function.

For example, with $N_c = 3$ classes the bounds $B_c = \{-0.1, 0.5\}$ would define three intervals $\{(-\infty, -0.1], (-0.1, 0.5], (0.5, \infty)\}$. Any $\hat{y}_i$ would be mapped to the label 0 if it is smaller than -0.1, to 1 if lies in $(-0.1, 0.5]$ and to 2 otherwise. Finally, we shuffle the labels of classes, i.e. we remove the ordering of class labels w.r.t. the ranges.

5 Experiments

5.1 Evaluation on Toy Problems

We first qualitatively compare our TabPFN to standard classifiers (without hyperparameter tuning) in Figure 4. The top row shows the *moons* dataset with noise. The TabPFN accurately models the decision boundary between samples; also, similar to Gaussian processes, uncertainties are large for points far from observed samples. The second row shows the *circles* dataset with noise: the TabPFN accurately models the circle's shape with high confidence anywhere outside the region where samples are mixed. The third row shows two classes and features from the iris dataset, while the fourth rows shows two classes and features from the wine dataset (both in *scikit-learn* (Pedregosa et al., 2011)).

5.2 Evaluation on Tabular ML Tasks

Now, we turn to an empirical analysis of our method for real-world classification tasks. We compare our method against state-of-the-art ML and AutoML methods for tabular classification.

Datasets As test datasets, we used all datasets from the curated open-source OpenML-CC18 benchmark suite (Bischl et al., 2021) that contain up to 2 000 samples (1 000 for the training split),

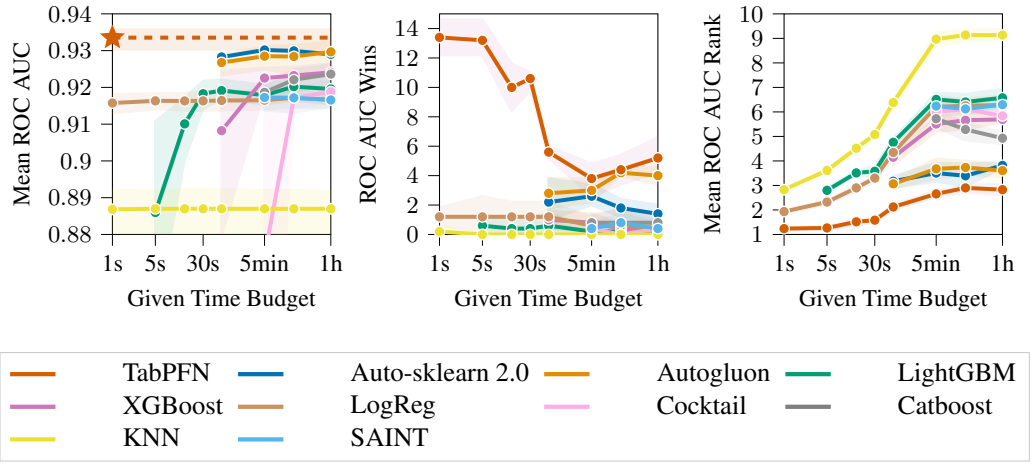


Figure 5: ROC AUC as a function of the time allowed to train & tune methods, on 18 numerical datasets from the OpenML-CC18 Benchmark. We report the mean, mean wins and rank and 95% C.I. across 5 splits for increasing budgets. The red star indicates performance of TabPFN with 32 permutations (which requires 0.42s on GPU). We report continued results in Table 1.

100 features and 10 classes. The resulting set comprises 30 datasets. We split these datasets into 18 datasets that contain only numerical features and no missing values, and 12 other datasets that contain categorical features and/or missing values. In the main paper, in Figure 5 and Table 1, we limit our analysis to the case of numerical datasets without missing values, which we focussed the development of our TabPFN prior on. Results on all of the 30 datasets are given in Appendix B.1 and still show strong aggregate performance for TabPFN, albeit not as strong as for the purely numerical case, due to generally worse performance of TabPFN on datasets with categorical features and/or missing values. We focus on small datasets because (1) small datasets are often encountered in real-world applications (Dua and Graff, 2017), (2) existing DL methods are most limited in this domain (Grinsztajn et al., 2022) and (3) the TabPFN would be significantly more expensive to train and evaluate for larger datasets, a limitation detailed in Appendix A.

Baselines We compare against five standard ML methods and two state-of-the-art AutoML systems for tabular data. As ML models we considered two simple and fast baselines, *K-nearest-neighbors (KNN)* and *Logistic Regression (LogReg)*. Additionally, we considered three popular tree-based boosting methods, *XGBoost* (Chen et al., 2020), *LightGBM* (Ke et al., 2017) and *CatBoost* (Prokhorenkova et al., 2018). For each ML model, we used 5-fold cross-validation to evaluate randomly drawn hyperparameter configurations until a given budget was exhausted or 10 000 configurations were evaluated (for the search spaces, see Appendix F.2). We then chose the best-performing hyperparameter configuration (maximum ROC AUC OVO) and refit on the whole training set. Where necessary, we imputed missing values with the mean, one-hot or ordinal encoded categorical inputs, normalized features and passed categorical feature indicators. As more complex but powerful baselines, we chose two state-of-the-art AutoML systems: *AutoGluon* (Erickson et al., 2020), which combines ML models including neural networks and tree-based models into a stacked ensemble, and *Auto-sklearn 2.0* (Feurer et al., 2015; 2021) which uses Bayesian Optimization and combines the evaluated models into a weighted ensemble.[2] We note that previous works have found that DL baselines do not outperform or match the performance of GBDT or AutoML methods for small to medium-sized tabular data (< 10,000 samples; while matching GBDT performance on larger datasets) (Borisov et al., 2021; Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022). Furthermore, DL methods such as TabNet, SAINT, Regularization Cocktails, Non-parametric Transformers (Arik and Pfister, 2021; Somepalli et al., 2021; Kadra et al., 2021; Kossen et al., 2021) are evaluated on much larger datasets and often use custom parameter tuning and preprocessing. We still evaluate two prominent DL methods: *Regularization Cocktails*[3] and *SAINT* (Kadra et al., 2021; Somepalli et al., 2021).

[2]Auto-sklearn 2.0 optimizes ROC AUC for binary classification and cross-entropy for multi-class classification (as multi-class ROC AUC is not implemented).

[3]Regularization Cocktails are optimized for cross-entropy, which performed better than balanced accuracy (the default). Regularization Cocktails cannot be optimized for multi-class AUC ROC.

	LightGBM 0	CatBoost 0	XGBoost 0	ASKL2.0 0	AutoGluon 1	TabPFN$_{n.e.}$ 3	TabPFN **4**	TabPFN + AutoGluon 3
M. rank AUC OVO	6.9722	4.9444	6.1944	4.4722	4	3.8056	2.9444	**2.6667**
Mean rank Acc.	6.8889	4.9722	6.0556	5.1667	3.8889	3.8889	2.8889	**2.25**
Mean rank CE	5.7778	5.4444	6	6.4167	3.1111	4.1389	3.0278	**2.0833**
Mean AUC OVO	0.92±.013	0.924±.011	0.924±.01	0.929±.0096	0.93±.0091	0.932±.0088	**0.934**±.0086	**0.934**±.0084
Mean Acc.	0.862±.012	0.864±.011	0.866±.011	0.87±.014	0.881±.01	0.873±.0095	0.879±.0089	**0.886**±.0094
Mean CE	0.75±.039	0.747±.029	0.759±.04	0.813±.073	0.714±.014	0.727±.021	0.716±.019	**0.711**±.014
Mean time (s) (Tune + Train + Predict)	3280	3746	3364	3601	3077	**0.9478** (CPU) **0.0204** (GPU)	26.80 (CPU) 0.4832 (GPU)	3104 (CPU) 3077 (GPU)

Table 1: Results on the 18 numerical datasets in OpenML-CC18 for 60 minutes requested time per data-split, ± values indicate a metric's standard deviation. If available, all baselines optimize ROC AUC. TabPFN n.e. is a faster version of our method, without ensembling, while TabPFN + AutoGluon is an ensemble of TabPFN and AutoGluon. Separate inference times in Table 2.

Evaluation Protocol For each dataset and method, we evaluated 5 repetitions, each with a different random seed and a train- and test split (50% train and 50% evaluation samples; all methods used the same split given a seed). To aggregate results across datasets, we report the ROC AUC (one-vs-one (OVO) for multi-class classification) average, ranks and wins including the 95% confidence interval and compare to the performance of the baselines with a budget of $\{30, 60, 300, 900, 3600\}$ seconds.[4] Our TabPFN uses 32 data permutations for ensembling as described in Section 3; we also evaluate TabPFN without permutations, which we label "TabPFN$_{n.e.}$" in Table 1.

Results We now present results for the 18 purely numerical datasets without missing values, detailed in Figure 5 and Table 1. Figure 5 shows that TabPFN achieves a dramatically better tradeoff of accuracy and training speed than all the other methods: it makes predictions within less than a second on one GPU that tie with the performance of the best competitors (the AutoML systems) after training one hour, and that dominate the performance of tuned GBDT methods. Unsurprisingly, the simple baselines (*LogReg*, *KNN*) already yield results with a small budget but perform worst overall. GBDTs (*XGBoost*, *CatBoost*, *LightGBM*) perform better but are still outperformed by TabPFN and the state-of-the-art AutoML systems (*Auto-sklearn 2.0*, *Autogluon*).

TabPFN is much faster than comparably performing methods. In the following, we compare the times for training and prediction (and tuning if applicable) together. TabPFN$_{n.e.}$ requires 0.0187s on a GPU or 0.87s on a CPU on average to predict for one dataset. It performs comparably to the strongest baselines at one minute, yielding a $70\times$ speedup on CPU and a $3\,200\times$ speedup using a GPU. Given more budget (≥ 5 minutes), the strongest baseline achieves competitive performance with the original TabPFN (including ensembling), which uses a time budget of 0.42 seconds on a GPU or 24.5s on a CPU. If we compare TabPFN against the 5 minutes required for the baseline, it yields a $12\times$ speedup on a CPU and a $710\times$ speedup using a GPU. These times assume that our trained model is in (GPU) memory already, which otherwise required 0.2s for us. We ignore computational development costs of each method, see Appendix F.5 for why.

We would like to emphasize that the discussed results are *aggregate* results across datasets, and that no classification method, including TabPFN, performs best on all individual datasets. Indeed, there do exist datasets for which TabPFN is outperformed even by default baselines. Generally, TabPFN is less strong when categorical features or missing values are present. Appendix B.1 presents results for all 30 test datasets from the OpenML-CC18 Benchmark, including 12 with categorical features and/or missing values. Appendix B presents a more detailed overview of the results for different kinds of datasets, including on an additional 149 validation datasets from OpenML, confirming TabPFN's strong performance for purely numerical datasets. We also show per dataset results for each of 179 datasets (our 30 test datasets and the 149 validation datasets); these show that, while TabPFN generally does better for datasets with numerical features, there also exist several purely numerical datasets for which TabPFN does *not* perform better than the baselines, as well as categorical datasets for which it *does* clearly perform best.

[4] When comparing methods to each other for a given time budget in Figure 5, we drop methods that take more than 200% of the requested time budget; some methods also do not use their full budget.

OpenML-AutoML Benchmark Additionally, we evaluated our TabPFN on the small ($\leq 1\,000$ training samples, 100 features, 10 classes) datasets of the OpenML-AutoML Benchmark, for which externally-validated performance numbers are available. We use the setup provided by the OpenML-AutoML Benchmark, i.e.: metrics, official evaluation scripts and pre-released evaluations for an even wider range of AutoML baselines. Using only an average of 4.4 seconds per dataset on a single CPU, compared to 60 minutes for the baselines, TabPFN outperformed all baselines in terms of mean cross-entropy, accuracy and the OpenML Metric[5]. We provide detailed results in Appendix B.2.

In-depth analysis of TabPFN predictions We evaluated our model predictions in a plethora of ways to provide extra insights, such as the inductive biases of our method, see Appendix B.3. We confirm that TabPFN learns to make predictions biased towards simple causal explanations, as detailed in B.3.1, while GBDT methods do not share this inductive bias. We also evaluate our method's invariance to feature rotations and robustness to uninformative features in Appendix B.3.2 and B.3.3. In Figure 6, we observe that TabPFN is especially strong compared to baselines when datasets do not contain categorical features or missing values.

Ensembling We observe that TabPFN performs best on different datasets than our baselines, i.e., the per-dataset correlation of normalized ROC AUC scores between TabPFN and the strong baselines is lower than between the strong baselines, visualized in Figure 11 in our Appendix. This is likely due to the novel inductive biases of our approach, which lead to distinct predictions. Ensembling predictions is more effective when the considered methods make fewer correlated errors (Breimann, 2001), which encourages the use of more diverse strategies, making TabPFN an ideal candidate for ensembling with baseline methods (Wu et al., 2021). Also, TabPFN is evaluated so quickly, that predictions are almost free, compared to the long runtimes of the baselines. To demonstrate this, we include an entry *"TabPFN + AutoGluon"* in Table 1, generated by averaging the predictions of TabPFN and *AutoGluon*; this strongly outperforms all other methods.

Model Generalization The PFN architecture accepts datasets of any length as input. However, when training our model in the *synthetic prior-fitting* phase, we limited synthetic data to a maximum size of $1\,024$, as prior-fitting becomes more expensive with larger datasets. Nevertheless, we wondered: Would TabPFN generalize to larger training set sizes that were never seen during training? To test this, we used a collection of 18 datasets from the OpenML-AutoML Benchmark (see Table 11) from which we selected $10\,000$ samples each. We evaluated TabPFN on up to $5\,000$ training samples, using 5 random data splits and 5000 test samples. Surprisingly, our models generalize beyond sample sizes seen during training, as shown in Figure 10.

6 Conclusions & Future Work

We have shown how a single Transformer, the TabPFN, can be trained to do the work of a full AutoML framework for tabular data and can yield predictions in 0.4 seconds that are competitive with the performance that the best available AutoML frameworks achieve in 5 to 60 minutes. This slashes the computational expense of AutoML, enabling affordable, green, AutoML.

The TabPFN still has important limitations: the underlying Transformer architecture only scales to small datasets as detailed in Appendix A; our evaluations focused on classification datasets with only up to $1\,000$ training samples, 100 purely numerical features without missing values and 10 classes, which motivates work on (1) scaling up to large datasets. Our in-depth analysis of the inductive biases of TabPFN (see Appendix B.3), points towards extensions for (2) improved handling of categorical features, (3) missing values and (4) robustness to unimportant features. Also, our work motivates a multitude of exciting follow-ups regarding (5) integration of TabPFN into existing AutoML frameworks; (6) ensembling to continue making improvements given more time; (7) dataset-dependent choices of the prior; (8) generalizations to non-tabular data and (9) regression tasks, as well as studies of the TabPFN regarding the dimensions of trustworthy AI (e.g., (10) out-of-distribution robustness; (11) algorithmic fairness, (12) robustness to adversarial examples, and (13) explainability). The almost instant state-of-the-art predictions of TabPFN are also likely to give rise to (14) novel exploratory data analysis methods, (15) novel feature engineering methods and (16) novel active learning methods. Finally, our advances in causal reasoning warrant follow-ups on (17) approximating the effects of interventions and counterfactuals considering a distribution of SCMs.

[5]The OpenML metric evaluates binary classification using ROC AUC and multiclass using Cross Entropy.