TuneTables: Context Optimization for Scalable Prior-Data Fitted Networks

Anonymous Authors[1]

Abstract

While tabular classification has traditionally relied on from-scratch training, a recent breakthrough called prior-data fitted networks (PFNs) challenges this approach. Similar to large language models, PFNs make use of pretraining and in-context learning to achieve strong performance on new tasks in a single forward pass. However, current PFNs have limitations that prohibit their widespread adoption. Notably, TabPFN achieves very strong performance on small tabular datasets but is not designed to make predictions for datasets of size larger than 1000. In this work, we overcome these limitations and substantially improve the performance of PFNs by developing context optimization techniques for PFNs. Specifically, we propose TuneTables, a novel prompt-tuning strategy that compresses large datasets into a smaller learned context. TuneTables scales TabPFN to be competitive with state-of-the-art tabular classification methods on larger datasets, while having a substantially lower inference time than TabPFN on average. Furthermore, we show that TuneTables can be used as an interpretability tool and can even be used to mitigate biases by optimizing a fairness objective.

1. Introduction

Tabular data, or data organized into rows and columns consisting of distinct features, are the oldest and one of the most ubiquitous types of data in machine learning in practice (Shwartz-Ziv & Armon, 2022; Borisov et al., 2021). Tabular data has numerous applications across medicine (Johnson et al., 2016; Ulmer et al., 2020), online advertising (Richardson et al., 2007; McMahan et al., 2013; Guo et al., 2017), finance (Arun et al., 2016; Clements et al., 2020), and other areas (Chandola et al., 2009; Buczak & Guven, 2015; Urban & Gates, 2021).

Competitive classification algorithms for tabular data include gradient-boosted decision trees (Chen & Guestrin, 2016; Prokhorenkova et al., 2018) and deep neural networks (Somepalli et al., 2021; Gorishniy et al., 2021; Kadra et al., 2021). Both approaches fit their respective models on a labeled dataset containing samples from a distribution reflecting the task at hand. A recent breakthrough, prior-data fitted networks (PFNs) (Hollmann et al., 2023; Müller et al., 2022), are transformers pretrained on data generated from a prior, to perform approximate Bayesian inference in a single forward pass using in-context learning. PFNs do not require optimizing parameters or fitting a model on downstream training data, instead feeding training data into the context and conditioning on it. In particular, TabPFN achieved state-of-the-art classification on small tabular datasets (Hollmann et al., 2023; McElfresh et al., 2023).

The in-context learning approach of PFNs parallels that of large language models (LLMs) (Zhao et al., 2023). Both approaches can be viewed as approximate Bayesian inference, whether implicitly (Xie et al., 2022) or explicitly (Müller et al., 2022). While researchers have successfully used various context optimization strategies for enhancing LLM performance (Liu et al., 2023), no prior work has studied context optimization strategies for PFNs. Furthermore, although TabPFN achieves very strong performance on small datasets, its limitations currently prohibit its widespread adoption: it only runs on datasets whose number of training samples, number of features, and number of classes are at most 1000, 100, and 10, respectively.

In this work, we perform the first investigation into context optimization strategies for PFNs, allowing us to substantially improve their performance when scaled to large datasets. Specifically, we introduce **TuneTables**, a novel prompt-tuning technique for PFNs that compresses large datasets into a smaller learned context (Figure 1), overcoming all of TabPFN's aforementioned constraints. TuneTables compares favorably to state-of-the-art tabular classification algorithms, such as CatBoost (Prokhorenkova et al., 2018), on datasets up to 50k in size, and there even exist datasets with one million data points for which it is better; this is three orders of magnitude larger than the original dataset size limit for TabPFN.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

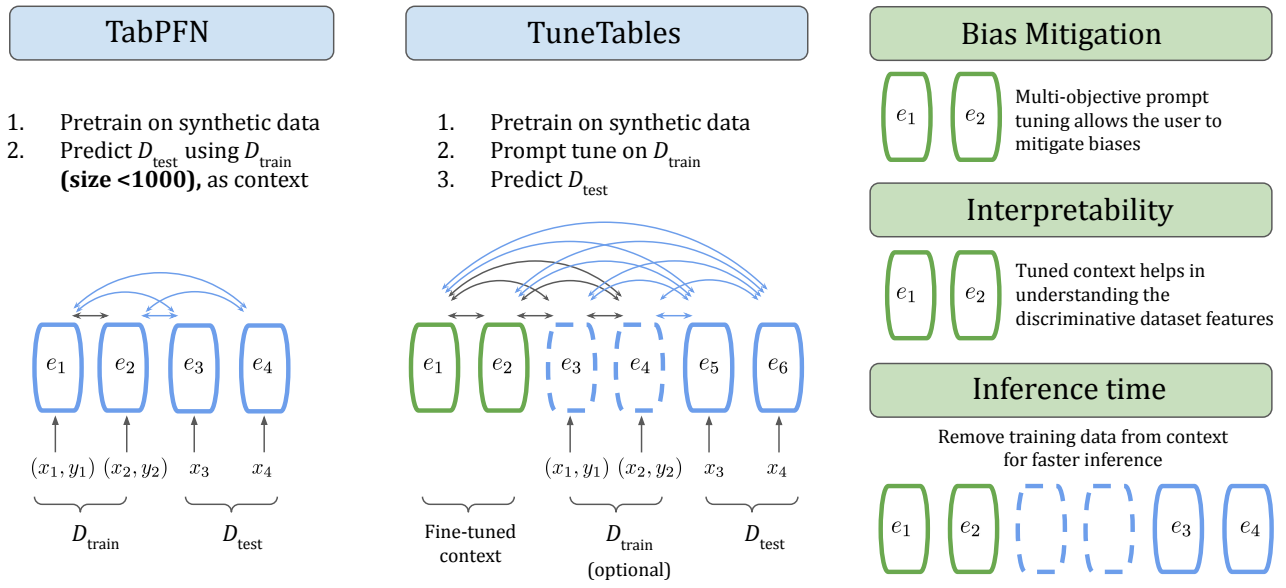

Figure 1. **TuneTables:** a novel prompt-tuning technique for prior-data fitted networks. TuneTables summarizes a large dataset, achieving stronger performance and shorter inference time than TabPFN. It can also be used for bias mitigation and as an interpretability tool.

We show that TuneTables improves over TabPFN across multiple other axes. Unlike in TabPFN, the *training dataset is not needed in the context during inference*, significantly speeding up inference time (similar to works on neural processes; see Section 2). Furthermore, we show how to use prompt tuning for **multi-objective optimization**, such as optimizing both accuracy and fairness simultaneously, allowing users to mitigate biased predictions of a pretrained PFNs with just a lightweight tuning procedure. Finally, we show that TuneTables can be used as a tool for **interpretability**. We open-source our code and raw results at `https://anonymous.4open.science/r/TuneTables`.

Our contributions.

- We introduce **TuneTables**, a novel prompt-tuning technique for PFNs. TuneTables substantially improves the performance of TabPFN on large datasets, making it competitive with the state of the art. Its inference time is significantly lower than for TabPFN on average.
- We show how to use prompt tuning for **multi-objective optimization**, such as optimizing both accuracy and fairness simultaneously, allowing users to mitigate biased predictions of a pretrained PFNs with just a lightweight tuning procedure.
- We demonstrate that TuneTables' condensed dataset representations can be used as an **interpretability** tool.
- We conduct an extensive study on context optimization strategies for PFNs by performing an ablation study on TuneTables, as well as studying sketching and feature selection techniques, across datasets with a wide variety of sizes and number of features.

2. Related Work

Neural Processes and Prior-Data Fitted Networks. Prior-data Fitted Networks (PFNs) (Müller et al., 2022; Hollmann et al., 2023) are a recently-proposed paradigm for machine learning, which show that fast approximate Bayesian inference is possible by training a neural network to mimic the posterior predictive distribution (PPD) in a single forward pass using in-context learning (Müller et al., 2022; 2023; Nagler, 2023). PFNs were shown to yield state-of-the-art empirical performance on small tabular datasets (Hollmann et al., 2023; McElfresh et al., 2023). PFNs have been used in other applications, including Bayesian optimization (Müller et al., 2023) and learning curve extrapolation (Adriaensen et al., 2023).

PFNs are neural processes (NPs) (Müller et al., 2022). Recent advances in NP are similar in nature to our work, especially recent works that aim to scale attention-based methods to larger contexts. Feng et al. (2023a) propose Latent Bottlenecked Attentive Neural Processes (LBANPs), a transformer-based neural process which overcomes the quadratic complexity of transformers by encoding the context dataset into a constant number of latent vectors. Guo et al. (2023) propose Versatile Neural Processes (VNP), which increases the capability of NPs to handle compex signals by using a new bottleneck encoder. (Rastogi et al., 2023) introduces semi-parametric inducing point networks (SPIN), which can attend to a training set at inference time

with linear complexity via inducing point methods. (Feng et al., 2023b) introduce Constant Memory Attention Block (CMAB), an attention block that is permutation-invariant and has constant memory complexity when computing the output, as well as Constant Memory Attentive Neural Processes (CMANPs), a NP that requires constant memory.

For additional related work, see Appendix A.

3. Background

3.1. PFNs: Review and Limitations

In this section, we give a background on PFNs and discuss their limitations. For a complete description, see Müller et al. (2022); Hollmann et al. (2023); Nagler (2023). Assume that we have a classification problem with features $\mathcal{X} \subseteq \mathbb{R}^d$ and labels $\mathcal{Y}$. Given a dataset $D = D_{\text{train}} \cup D_{\text{test}}$, where $D_{\text{train}} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and $D_{\text{test}} = \{(x_{\text{test}}, y_{\text{test}})\}$, our goal is to predict the conditional class probabilities $p(\cdot \mid x_{\text{test}})$. In the Bayesian framework for supervised learning, the mechanism for generating the data distribution is a hypothesis φ, drawn from Φ, the space of all hypotheses. Φ encodes our prior beliefs on the system before observing data. In this framework, datasets D are generated by first drawing $\varphi \sim \Phi$, and then drawing *i.i.d.* samples according to φ.

The posterior predictive distribution (PPD) for a test sample x_{test} is the label distribution $p(\cdot \mid x_{\text{test}}, D_{\text{train}})$ that follows from our prior. We can obtain the PPD by integrating over the space of hypotheses Φ:

$$p(y \mid x, D) \propto \int_{\Phi} p(y \mid x, \varphi) p(D \mid \varphi) p(\varphi) d\varphi. \quad (1)$$

A PFN is a transformer-based architecture trained to approximate the PPD via *synthetic prior-fitting*. Given a prior, we first sample hypotheses $\varphi \sim p(\varphi)$ and then synthetic datasets $D \sim p(D \mid \varphi)$. We optimize the parameters of the PFN by predicting the class labels of $D_{\text{test}} \subseteq D$, conditioned on $D_{\text{train}} = D \setminus D_{\text{test}}$. We compute the loss by:

$$\mathcal{L}_{\text{PFN}} = \mathbb{E}_{D \sim p(D)} \left[-\log q_\theta(y_{\text{test}} \mid x_{\text{test}}, D_{\text{train}}) \right], \quad (2)$$

for simplicity assuming all training and test sets are size n and 1, respectively. We then approximately solve this optimization problem, $\hat{\theta} = \arg\min_\theta \mathcal{L}_{\text{PFN}}$, allowing q_θ to approximate Equation (1):

$$q_\theta(y_{\text{test}} \mid x_{\text{test}}, D_{\text{train}}) \approx p(y_{\text{test}} \mid x_{\text{test}}, D_{\text{train}}). \quad (3)$$

Scaling challenges for PFNs. While PFNs, and specifically TabPFN, have shown remarkable success in classification by in-context learning, several important obstacles constrain their more widespread adoption:

1. **PFNs only accept a fixed number of features.** The current design of PFNs fixes the quantity of features at the time of pretraining. This quantity cannot be changed without retraining the PFN.
2. **PFNs scale poorly with the dataset size**. While PFN accuracy can improve with more real-data samples at inference time (Hollmann et al., 2023; Nagler, 2023), the memory requirements scale quadratically with the context length, making scaling beyond a certain number of samples (such as 3000) impractical.
3. **PFNs only select from a fixed number of classes**. The MLP decoder head co-trained with the PFN fixes the number of classes that can be identified at test time.

In the remainder of this section, we present sketching, feature selection, and fine-tuning as an attempt to remedy *(1)* and *(2)*. Then in Section 4, we describe novel prompt-tuning and class-extension strategies to create TuneTables, a robust classifier which remedies *(1)-(3)*.

3.2. Classical sketching and feature selection for PFNs

Sketching. The number of samples that a PFN can handle is limited to ∼3000 by conventional GPU sizes. However, in the real world, datasets are often much larger. Going forward, we refer the maximum allowable context size of the PFN as n, and to the size of the real-world dataset as N.

Given a training dataset D_{train} of size $N >> n$, one option is to select a representative subset of the dataset, $D_{\text{compact}} \subseteq D_{\text{train}}$, to use as the context. In general, researchers have studied a variety of data summarization techniques, often called *sketching*, for tabular data (Munteanu & Schwiegelshohn, 2018). In the context of a pretrained PFN q_θ, the goal is to find a sketching function $s : \mathbb{R}^{N \times d} \mapsto \mathbb{R}^{n \times d}$ such that

$$\begin{aligned} &\mathbb{E}_{D \sim p(D)} \left[-\log q_\theta(y_{\text{test}} \mid x_{\text{test}}, s(D_{\text{train}})) \right] \quad (4) \\ \approx\ &\mathbb{E}_{D \sim p(D)} \left[-\log q_\theta(y_{\text{test}} \mid x_{\text{test}}, D_{\text{train}}) \right]. \end{aligned}$$

Here, we consider three sketching methods for TabPFN: *random*, in which we simply select a random subset of n datapoints from the full training set; *k-means*, in which we compute the n-means clustering (Ahmed et al., 2020) of D_{train} and select the n centers; and *CoreSet*, in which we compute a core-set of size n (Agarwal et al., 2005).

Feature selection. In addition to the limitation on dataset size, PFNs, such as TabPFN, also impose a limitation on the number of features d. Similar to sketching, given a dataset with $D >> d$ features, we can perform feature selection or summarization in order to adhere to the constraint (formally, finding a function that reduces the feature dimension and approximates an expression similar to Equation (4)). Feature selection is a critical part of tabular classification, and

there are several popular feature selection methods (Chandrashekar & Sahin, 2014; Cherepanova et al., 2023). We investigate three different methods: *random*, in which we randomly select a set of d features; *mutual information*, in which we select d features with the highest mutual information of the target dataset (Vergara & Estévez, 2014); and *principal component analysis (PCA)*, in which we take the d first principal components.

In Section 5, we find that all sketching and feature selection methods plateau in performance well before approaching parity with GBDTs, which motivates our investigation of new scaling techniques in Section 4.1.

3.3. Fine-tuning PFNs

We conclude this section with a discussion of another potential approach for scaling PFNs: **fine-tuning**. In this approach, given a dataset of size $N >> n$, we use gradient descent to continue training *all parameters* of the PFN. However, we show in Section 5 that fine-tuning takes up considerable memory resources while still not achieving competitive accuracy on large datasets. Therefore, we present new solutions for scaling PFNs in the next section.

4. TuneTables: scaling PFNs

While the previous section described the limitations of PFNs and proposed initial attempts at solving it, we now present new prompt tuning and class extension methods for PFNs. In Section 4.3, we describe how to combine the different approaches to create TuneTables, a robust classifier for datasets of arbitrary dimensions.

4.1. Prompt tuning as a scalable context for PFNs

Motivated by the limitations of sketching for large contexts, we explore *soft prompt tuning* as an alternative. In soft prompt tuning (Lester et al., 2021), given a dataset $D = D_{\text{train}} \cup D_{\text{test}}$, a parameterized matrix $D_{\text{tune}}^{p \times e}$ is prepended to the input embedding $D_{\text{train}}^{n \times e}$, where e is the transformer embedding dimension, and p is a hyperparameter controlling the size of the tuned prompt.

Lester et al. (2021) demonstrate the effectiveness of soft prompt tuning for NLP tasks by prepending a small task-specific prompt ($p \approx 5$). These task-specific learned tokens prove effective at the extremely large model scales commonly encountered in NLP. Somewhat surprisingly, we show that prompt tuning is effective at similar scales of p even for the much smaller tabular data models (see Appendix C and Appendix D). However, prompt tuning increases in effectiveness when p is larger.

Soft prompt tuning for tabular data. Unlike in NLP, transformers for tabular data, including PFNs, generally accept two input embeddings; a continuous-valued embedding $D_{\text{train}\,X}$, and a categorically-valued $D_{\text{train}\,y}$ which is passed through directly. We adapt the method of Lester et al. (2021) by fitting the parameters of $D_{\text{tune}\,X}^{p \times e}$ to $D_{\text{train}\,X}$ and randomly initializing $D_{\text{tune}\,y}^{p \times 1}$ with an equal number of labels from each class in D_{train}. These synthetic datapoints are optimized on the entire labeled set, and therefore give PFNs access to a much larger training set not accessible by existing methods.

We further adjust the method of Lester et al. (2021) to allow for the possibility that D_{tune} has learned, in essence, a distilled version of D_{train}; at test time, we evaluate two settings, hereafter referred to as C ('context') and NC ('no context'). In the C setting, following Lester et al. (2021), we have $D_{\text{tune}}^{p \times e}$ prepended to the input embedding $D_{\text{train}}^{n \times e}$. In the NC setting, we provide only $D_{\text{tune}}^{p \times e}$ at test time. In Section 5, we empirically evaluate both approaches. We ablate specifics of our implementation choices in Appendix C. Unlike prompt tuning for NLP, we show that the NC setting is often competitive with, if not better than, the C setting. We also ablate this choice during training; see Appendix C for implementation specifics.

4.2. Extending the number of predicted classes

TabPFN uses a pretrained transformer, with a two-layer MLP as a final classifier. The pretraining procedures limit the naïve use of TabPFN to classification tasks with at most 10 classes. Yet, in many cases, datasets of interest might contain a larger number of classes, which would require pretraining a new PFN from scratch.

We present a new prompt tuning-based technique to alter the cardinality of classes in a pretrained PFN. Following the method of last-layer retraining (Kirichenko et al., 2022; Le et al., 2023), we fit a PFN to new posteriors given by real-world classification datasets with more than 10 classes, freezing all weights except for those of the decoder MLP and the tuned prompt. Additional implementation details can be found in Appendix C.

4.3. TuneTables overview

Motivated by the strong performance of prompt tuning, we now introduce **TuneTables**, a new tabular classification algorithm. Using a pretrained PFN (TabPFN, in our experiments) as a base model, TuneTables overcomes the limitations of PFNs, allowing them to be applied to any tabular classification problem. The key mechanism behind TuneTables is prompt tuning as described in the previous two sections. Given a new dataset, TuneTables runs in three stages:

1. In the **initial control flow** stage, TuneTables uses inexpensive metadata to efficiently determine which variations of the PFN to fit.

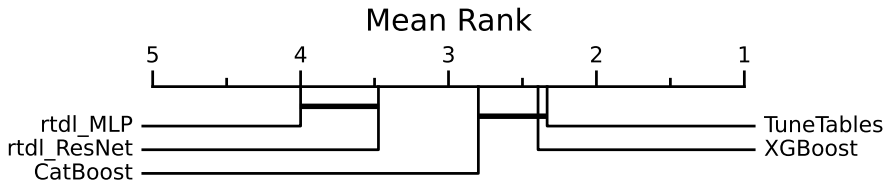

Figure 2. **Critical difference plot** according to mean accuracy rank across 19 datasets. Algorithms which are *not significantly different* ($p > 0.05$) are connected with a horizontal black bar.

2. In the **fitting** stage, the chosen PFN variants are fitted to the training data (subject to an optional time constraint).
3. In the **selection** stage, TuneTables selects the optimal variant, measured by performance on a validation set.

In the first stage, TuneTables uses three metadata properties (num. samples, num. features, and num. classes) to determine PFN variants on which to fit the data. The complete control flow is provided in Appendix Algorithm 2. We summarize the general guidelines as follows: *(a)* if a dataset is small enough to run with the original zero-shot version of TabPFN, we do so, as the TabPFN is already highly optimized for such datasets; *(b)* if there are too many features, we perform grid search over a set of feature subselection methods and select the one which performs best on the validation set; *(c)* orthogonally, if there are too many labels or samples, we fit a new tuned prompt (and decoder, if needed) to a frozen TabPFN.

See Appendix C for the full details of the algorithm. In the next section, we conduct rigorous benchmarking of our method, comparing it to state-of-the-art GBDTs and neural nets and conducting a thorough ablation study.

5. Experiments

Algorithms and datasets used. We compare TuneTables to eighteen baselines, including three GBDTs: CatBoost (Prokhorenkova et al., 2018), LightGBM (Ke et al., 2017), and XGBoost (Chen & Guestrin, 2016); 11 neural networks: DANet (Chen et al., 2022), FT-Transformer (Gorishniy et al., 2021), two MLPs (Gorishniy et al., 2021), NODE (Popov et al., 2020), ResNet (Gorishniy et al., 2021), SAINT (Somepalli et al., 2021), STG (Yamada et al., 2020), TabNet (Arik & Pfister, 2021), TabPFN (Hollmann et al., 2023), and VIME (Yoon et al., 2020); and five baselines: Decision Tree (Quinlan, 1986), KNN (Cover & Hart, 1967), Logistic Regression (Cox, 1958), Random Forest (Liaw et al., 2002), and SVM (Cortes & Vapnik, 1995). For all algorithms, we use their official implementation; see Appendix B for more details.

We run the algorithms on 98 classification datasets from OpenML (Vanschoren et al., 2014). These are used in recent tabular data studies (Grinsztajn et al., 2022; McElfresh et al., 2023) with a diversity of sizes and number of features (McElfresh et al., 2023). See Table 3 in Appendix B for a list of all datasets with their statistics.

Experimental setup. For all algorithms, we perform light hyperparameter tuning by running one default setting and 29 iterations of random search using Optuna (Akiba et al., 2019)); see Appendix B for details. For each dataset, each algorithm is allowed a maximum runtime of 10 hours for the entire search process on an Nvidia L4 TPU with 24GB VRAM. For TuneTables, light hyperparameter tuning is inherent in the algorithm (fewer than 30 configurations, described in Section 4.3), including embedding length and whether or not to include the training data in context. We perform prompt tuning for 30 epochs and early stopping using validation accuracy. For all algorithms, we report the test performance of the hyperparameter set with the best performance on the validation set, averaged over three train/test folds from OpenML. In order to average results across datasets, we report the mean accuracy as well as the mean normalized accuracy (after Z-score normalization) for each algorithm.

We start with a motivating example for performing context optimization. In Figure 3 (left), we present a comparison between CatBoost and the original TabPFN. Although the original TabPFN performs very well on datasets with fewer than 1000 datapoints and 100 features, it significantly underperforms CatBoost beyond those constraints, when naïvely scaling by randomly sampling features and datapoints.

Algorithm comparison. We compare the performance of all baselines to TuneTables across all datasets of size less than or equal to 50 000. We compute *statistically significant* performance differences among algorithms averaged across all datasets, as done in prior work (Kadra et al., 2021). We first use a Friedman test to check whether performance differences between all algorithms are significant (Friedman, 1937). We reject the null hypothesis for $p < 0.05$. Then we use a Wilcoxon signed-rank test to check which pairs of algorithms have significant performance differences (Conover, 1999). We use a Holm-Bonferroni correction to account for multiple comparisons (Holm, 1979). See

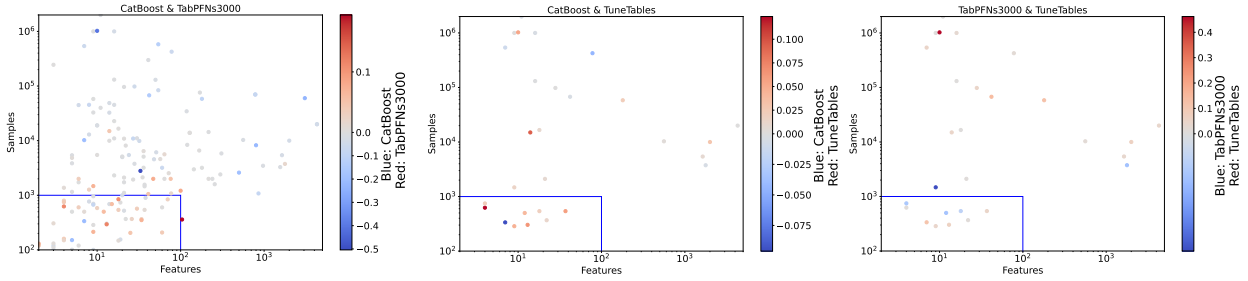

Figure 3. **Algorithm comparisons as a function of dataset size and num. features.** (Left) Motivating example (using the public results on 176 datasets from McElfresh et al. (2023)): the original TabPFN achieves very strong performance compared to CatBoost on small datasets. However, when naïvely scaled past 3000 datapoints and 100 features by randomly sampling datapoints and features, it significantly underperforms CatBoost. (Middle) CatBoost vs. TuneTables: TuneTables is competitive with CatBoost even on large datasets, mitigating the limitations of TabPFN. (Right) TabPFN vs. TuneTables: we show that TuneTables especially outperforms TabPFN on datasets with a high number of datapoints or features.

Figure 2. TuneTables achieves the highest average rank, although the difference among the top three algorithms is not statistically significant.

In Table 1, we present the accuracy, rank, and Z-score of all algorithms, averaged across all datasets of size less than or equal to 50 000. We find that TuneTables performs exceptionally well, achieving nearly the top score across all metrics. This result is in contrast to the original TabPFN, which is not meant to run on any dataset of size $> 1\,000$.

Large Datasets. While TuneTables achieves state-of-the-art performance across datasets up to size 50 000, we also show that it remains quite competitive up to and including size 1 000 000 (see Appendix Table 6), including outright winning on `poker-hand`, a dataset of size 1 025 009. Across all 98 datasets (sizes ranging from 286 to 1.9 million), TuneTables achieves the greatest number of wins across all datasets and achieves a mean rank of 2.74, compared to XGBoost's 2.31 and CatBoost's 2.61. We give the full results in Appendix B.

To better understand the relative strength of TuneTables across dataset sizes, we compare CatBoost to TuneTables in different regimes; see Figure 3 (middle). Interestingly, we find that TuneTables performs slightly better overall, and neither algorithm especially favors datasets with more features. This result is in contrast to Figure 3 (left), in which CatBoost significantly outperformed TabPFN on datasets with a large number of features or number of samples.

Class extension. While the previous experiments show performance on datasets with 10 or fewer classes, here we present results on datasets with more than 10 classes, using the extended classes version of TuneTables described in Section 4.2 and fine tuning for 100 epochs. In Table 5 we compare TuneTables to seven other algorithms on eight datasets with more than 10 classes. TuneTables outperforms CatBoost and is second only to XGBoost, despite the number of classes going far beyond the classes from the pretraining routine.

Comparison with TabPFN. We consider the improvement of TuneTables compared to the original TabPFN. Across all 29 datasets, TuneTables achieves a mean accuracy of 83.0%, compared to TabPFN's 78.1%; see Figure 3 (right). We find that TuneTables performs significantly better on average, but prompt tuning does not improve over TabPFN for datasets with fewer than 1000 datapoints and fewer than 100 features. This is because prompt tuning overfits to the small validation set; while validation accuracy improves, the test accuracy suffers. In terms of inference time, we find that TuneTables is significantly faster than TabPFN on average; see Table 8 for the full details.

We also compare TuneTables to fully fine-tuning TabPFN as described in Section 3.3, with implementation details described in Appendix C. Surprisingly, we find that TuneTables outperforms fine-tuning TabPFN, while being significantly more memory efficient; see Appendix Table 9.

TuneTables ablation study We conduct an ablation study for TuneTables, in order to give a thorough analysis of the efficiency and effectiveness of TuneTables. We ablate the prompt length and the use of ensembles in Appendix Table 9 and Table 10. We tested prompt lengths of 10, 100, and 1000; while some datasets are not sensitive to the prompt length, others vary significantly and generally prefer longer prompts. Similarly, we find some datasets are not sensitive to ensembling, and others, especially smaller datasets, generally prefer ensembling.

Next, we consider prompt tuning with and without keeping the real data as additional context (referred to as C and NC, respectively, in Section 4); see Appendix Table 9. We find that for smaller datasets, keeping the size-1000 real data in

Table 1. **Comparison of algorithms over 98 datasets** For each algorithm, we compute its mean accuracy, mean runtime, and mean rank in terms of accuracy. We also compute the mean Z-score, computed by normalizing the set of results on each dataset (by mean 0 std. 1), so that each dataset has the same weight, and averaging each algorithm's normalized performances. We see that TuneTables performs the best across all performance-oriented metrics. Fractional num. wins values are averaged over three splits per dataset, and reflect the presence of multi-way ties on certain splits.

Method	Mean Acc.	Mean Rank	Mean Z-Score	Std. Z-Score	Med. Z-Score	Num. Wins
CatBoost	0.857	7.206	0.472	0.461	0.517	8.344
TuneTables	0.855	7.286	0.440	0.407	0.443	13.137
XGBoost	0.855	7.551	0.397	0.480	0.531	9.307
RandomForest	0.844	9.230	0.209	0.506	0.335	4.003
LightGBM	0.847	8.600	0.286	0.521	0.439	9.350
NODE	0.839	8.784	0.230	0.381	0.368	3.910
SAINT	0.842	9.022	0.161	0.439	0.366	6.795
DANet	0.841	9.080	0.214	0.480	0.344	3.295
rtdl_ResNet	0.841	9.133	0.184	0.495	0.373	4.956
rtdl_FTTransformer	0.840	9.199	0.209	0.465	0.309	5.698
SVM	0.836	9.284	0.139	0.380	0.309	7.678
rtdl_MLP	0.815	10.207	-0.021	0.438	0.217	2.649
TabNet	0.806	10.944	-0.170	0.547	0.117	4.708
STG	0.811	11.488	-0.257	0.415	-0.006	3.708
DecisionTree	0.824	11.522	-0.266	0.523	0.055	3.227
MLP	0.803	11.611	-0.210	0.446	-0.142	1.278
LinearModel	0.792	12.434	-0.500	0.432	-0.318	2.881
KNN	0.782	13.396	-0.654	0.415	-0.459	0.860
VIME	0.755	14.024	-0.777	0.428	-0.637	2.218

context during prompt tuning and inference, along with the tuned prompt, achieves the best performance. For larger datasets, the tuned prompt alone suffices, and in some cases even outperforms having the real data in the context.

Sketching and feature selection Finally, in Appendix B.1 we give a study on the three sketching and three feature selection techniques described in Section 3. As described earlier, these methods plateau in performance well before approaching parity with GBDTs.

6. Mitigating Bias with Prompt Tuning

The previous section showed that prompt tuning allows PFNs to scale to large datasets that do not fit into the context, while also improving accuracy. In this section, we show another important use case enabled by prompt tuning: *optimizing for multiple objectives*, and in particular, *mitigating bias*.

Many real-world applications of machine learning involve a set of protected attributes (such as race or gender) that partition the dataset into groups, in which some groups have higher model performance than others. Even removing the sensitive attributes does not fix the algorithmic bias, because the sensitive attributes are often non-trivially correlated with other attributes in the dataset. Due to this issue, researchers have put in significant effort into mitigating the bias of ML models, with the majority of techniques devising new training strategies (Barocas et al., 2023; Mehrabi et al., 2021).

Given TabPFN's pretrained nature and prohibitive retraining cost, the only options for mitigating biased predictions are to run a post-processing routine on the output predictions, which generally do not perform as well as in-processing strategies (Savani et al., 2020). We show how to use prompt tuning to substantially reduce the bias of predictions while also improving accuracy.

We conduct experiments on four datasets widely used for research in fairness: the Adult Census Income database (with sex as the sensitive attribute) (Asuncion & Newman, 2007), speed dating (with same race as the sensitive attribute) (Zheng et al., 2018), COMPAS (with sex as the sensitive attribute) (Angwin et al., 2022), and National Longitudinal Survey (with gender as the sensitive attribute) (US Bureau of Labor Statistics, 2023). To quantify bias, we use *demographic parity* (Verma & Rubin, 2018; Dwork et al., 2012), which measures the difference in probability of a positive outcome among the protected and unprotected groups. Formally, given protected group G_1, unprotected group G_0, and protected attribute $x_{\cdot,a}$, it can be computed as

$$P_{(x_i,y_i)\in G_0}(y_i = 1 \mid x_{i,a}) - P_{(x_i,y_i)\in G_1}(y_i = 1 \mid x_{i,a}).$$

During prompt tuning, we employ a demographic parity regularizer that aims to minimize the difference in positive outcome probabilities between the two groups:

$$\left|\sum_{(x_i,y_i)\in G_0} P(y_i = 1 \mid x_{i,a}) - \sum_{(x_i,y_i)\in G_1} P(y_i = 1 \mid x_{i,a})\right|$$

Table 2. **Prompt tuning for mitigating bias.** We compare the default TabPFN to TuneTables, tuning for accuracy alone vs. accuracy and demographic parity. The latter significantly improves both accuracy and demographic parity, compared to the default TabPFN.

	Adult		Speeddating		Compas		NLSY	
	Acc ↑	DP ↓	Acc ↑	DP ↓	Acc ↑	DP ↓	Acc ↑	DP ↓
TabPFN	0.832	0.174	0.86	0.012	0.688	0.22	**0.986**	0.326
TuneTables (Acc)	**0.845**	0.13	**0.865**	0.006	0.688	0.209	0.974	0.302
TuneTables (Acc + DP)	0.837	**0.034**	0.863	**0.003**	**0.693**	**0.121**	0.965	**0.277**

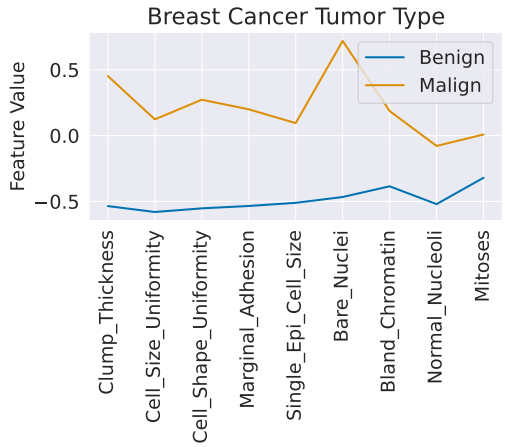

Figure 4. **Dataset with high accuracies from just two datapoints**. Shown is a two-example prompt dataset for the breast cancer dataset (Wolberg et al., 1995). Malign class example has higher values for all features than benign class.

We use the same experimental setup as in Section 5, except that we report one shift rather than the average of three, and TuneTables is fitted to a single prompt rather than ensembled. We compare the default TabPFN to TuneTables, fine-tuning for accuracy alone vs. accuracy and demographic parity. The latter significantly improves demographic parity, compared to the default TabPFN and TuneTables fine-tuned for accuracy, and enhances accuracy relative to the default TabPFN across all datasets but one; see Table 2.

7. Summarizing and Understanding Datasets with Prompt Tuning

While we have demonstrated that TuneTables scales and improves the performance of PFNs, now we show that it can also help understand the discriminative features in a dataset. Often, besides a good predictive model for a dataset, users want to gain further insights into the dataset. Here, prompt tuning can help: the tuned smaller dataset can be seen as a summary of the complete dataset that emphasizes discriminative features for the given task. As an example, in Figure 4 we show that on the Breast Cancer dataset (Wolberg et al., 1995), a prompt with just two synthetic examples is enough to reach high accuracies and at the same time allows an understanding of the predictive features. For example, in the Breast Cancer dataset, the malign example has higher values for all features compared to the benign example, suggesting high feature values indicate malignancy. We give further examples in Appendix D.

8. Conclusions and Future Work

In this work, we gave the first investigation into context optimization techniques for PFNs, allowing us to substantially improve their performance when scaled to large datasets. In particular, we introduced TuneTables, which uses a novel prompt-tuning technique to achieve strong performance on large datasets, and even outperforms state-of-the-art algorithms on some datasets of size one million. We demonstrate that TuneTables mitigates the constraints of TabPFN on the dataset size, number of features, and number of class labels. Additionally, we use prompt tuning to mitigate bias without retraining TabPFN and as an interpretability tool. We open-source our code, results, and all other materials needed to reproduce our work. As PFN models continue to improve and scale up, the context optimization techniques explored in our work will allow researchers to further optimize and scale such models. For example, a next-generation TabPFN might have a longer total context length, and prompt tuning will allow us to push the dataset size even further.

In the future, prompt tuning can be used with PFNs to ensure high-quality, differentially private predictions, given that the pretraining is done on purely synthetic data, and prompt tuning only updates a small number of parameters. Studying additional parameter-efficient fine-tuning methods for PFNs would be another interesting area of future study. For example, low-rank adaptation (LoRA) (Hu et al., 2022) and quantized low-rank adaptation (QLoRA) (Dettmers et al., 2023) have been used successfully for large language models and would be a promising technique for parameter-efficient fine-tuning of PFNs. Designing a sparse mixture-of-experts PFN using router networks is another promising technique, due to its success in the field of large language models (Jiang et al., 2024).

9. Broader Societal Impact Statement

The goal of our work is to investigate context optimization strategies for PFNs, such as prompt-tuning and sketching, which we use to scale and improve the performance of TabPFN. We do not see any negative broader societal impacts of our work that do not already exist in other classification methods. In fact, our work may further facilitate the adoption of TabPFN, which has the benefit of being pretrained and therefore having a lower carbon footprint compared to most deep learning approaches that must be trained from scratch. For example, throughout our work, we did not once train a PFN from scratch, instead relying on the original pretrained TabPFN (Hollmann et al., 2023).

Furthermore, we demonstrate that our prompt-tuning strategy makes it possible to mitigate the bias of TabPFN while only fine-tuning a small set of embedding vectors; the authors of TabPFN mentioned that it is critical to study and improve TabPFN under the lens of algorithmic fairness and other dimensions of trustworthy AI (Hollmann et al., 2023). This may allow practitioners to use TabPFN in sensitive settings for the first time, in which the original TabPFN would lead to biased preditions. Overall, our hope is that our work will have a positive impact for both practitioners and researchers, by facilitating the adoption of a model with a low carbon footprint, and by providing the tools to mitigate the bias of PFNs. Likewise, we hope that the possibility to summarize datasets with prompt tuning will add to the toolbox of machine learning practitioners aiming to analyze and interpret their data better, and therefore may have a positive impact on the trustworthieness of machine learning.

References

Adriaensen, S., Rakotoarison, H., Müller, S., and Hutter, F. Efficient bayesian learning curve extrapolation using prior-data fitted networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Agarwal, P. K., Har-Peled, S., Varadarajan, K. R., et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 2005.

Ahmed, M., Seraj, R., and Islam, S. M. S. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 2020.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Angwin, J., Larson, J., Mattu, S., and Kirchner, L. Machine bias. In *Ethics of data and analytics*, pp. 254–264. Auerbach Publications, 2022.

Arik, S. Ö. and Pfister, T. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

Arun, K., Ishan, G., and Sanmeet, K. Loan approval prediction based on machine learning approach. *IOSR J. Comput. Eng*, 18(3):18–21, 2016.

Asuncion, A. and Newman, D. Uci machine learning repository, 2007.

Barocas, S., Hardt, M., and Narayanan, A. *Fairness and machine learning: Limitations and opportunities*. MIT Press, 2023.

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

Buczak, A. L. and Guven, E. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18 (2):1153–1176, 2015.

Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3): 1–58, 2009.

Chandrashekar, G. and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering*, 2014.

Chen, J., Liao, K., Wan, Y., Chen, D. Z., and Wu, J. Danets: Deep abstract networks for tabular data classification and regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Cherepanova, V., Levin, R., Somepalli, G., Geiping, J., Bruss, C. B., Wilson, A. G., Goldstein, T., and Goldblum, M. A performance-driven benchmark for feature selection in tabular deep learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Clements, J. M., Xu, D., Yousefi, N., and Efimov, D. Sequential deep learning for credit risk monitoring with tabular financial data. *arXiv preprint arXiv:2012.15330*, 2020.

Conover, W. J. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 1995.

Cover, T. and Hart, P. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 1967.

Cox, D. R. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1958.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Dooley, S., Khurana, G. S., Mohapatra, C., Naidu, S. V., and White, C. Forecastpfn: Synthetically-trained zero-shot forecasting. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Dwork, C., Hardt, M., Pitassi, T., Reingold, O., and Zemel, R. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference*, 2012.

Feng, L., Hajimirsadeghi, H., Bengio, Y., and Ahmed, M. O. Latent bottlenecked attentive neural processes. In *The Eleventh International Conference on Learning Representations*, 2023a.

Feng, L., Tung, F., Hajimirsadeghi, H., Bengio, Y., and Ahmed, M. O. Memory efficient neural processes via constant memory attention block. *OpenReview*, 2023b.

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

Friedman, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 1937.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*, 2017.

Guo, Z., Lan, C., Zhang, Z., Lu, Y., and Chen, Z. Versatile neural processes for learning implicit neural representations. In *The Eleventh International Conference on Learning Representations*, 2023.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. Tabpfn: A transformer that solves small tabular classification problems in a second. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Huang, X., Khetan, A., Cvitkovic, M., and Karnin, Z. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Jiang, Z., Xu, F. F., Araki, J., and Neubig, G. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 2020.

Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L.-w. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., and Mark, R. G. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

Kadra, A., Lindauer, M., Hutter, F., and Grabocka, J. Well-tuned simple nets excel on tabular datasets. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Kirichenko, P., Izmailov, P., and Wilson, A. G. Last layer re-training is sufficient for robustness to spurious correlations. *arXiv preprint arXiv:2204.02937*, 2022.

Le, P. Q., Schlötterer, J., and Seifert, C. Is last layer re-training truly sufficient for robustness to spurious correlations? *arXiv preprint arXiv:2308.00473*, 2023.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Levin, R., Cherepanova, V., Schwarzschild, A., Bansal, A., Bruss, C. B., Goldstein, T., Wilson, A. G., and Goldblum, M. Transfer learning with deep tabular models. *ICLR*, 2023.

Liaw, A., Wiener, M., et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 2023.

McElfresh, D., Khandagale, S., Valverde, J., Ramakrishnan, G., Prasad, V., Goldblum, M., and White, C. When do neural nets outperform boosted trees on tabular data? In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., et al. Ad click prediction: a view from the trenches. In *Proceedings of the Annual Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1222–1230, 2013.

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., and Galstyan, A. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 2021.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. Transformers can do bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

Müller, S., Feurer, M., Hollmann, N., and Hutter, F. Pfns4bo: In-context learning for bayesian optimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

Munteanu, A. and Schwiegelshohn, C. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 2018.

Nagler, T. Statistical foundations of prior-data fitted networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Popov, S., Morozov, S., and Babenko, A. Neural oblivious decision ensembles for deep learning on tabular data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

Quinlan, J. R. Induction of decision trees. *Machine learning*, 1986.

Rastogi, R., Schiff, Y., Hacohen, A., Li, Z., Lee, I., Deng, Y., Sabuncu, M. R., and Kuleshov, V. Semi-parametric inducing point networks and neural processes. In *The Eleventh International Conference on Learning Representations*, 2023.

Richardson, M., Dominowska, E., and Ragno, R. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pp. 521–530, 2007.

Rubachev, I., Alekberov, A., Gorishniy, Y., and Babenko, A. Revisiting pretraining objectives for tabular deep learning. *arXiv preprint arXiv:2207.03208*, 2022.

Savani, Y., White, C., and Govindarajulu, N. S. Intra-processing methods for debiasing neural networks. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

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

Smith, J. W., Everhart, J. E., Dickson, W., Knowler, W. C., and Johannes, R. S. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the annual symposium on computer application in medical care*, pp. 261. American Medical Informatics Association, 1988.

Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C. B., and Goldstein, T. Saint: Improved neural networks for tabular data via row attention and contrastive pretraining. *arXiv preprint arXiv:2106.01342*, 2021.

Tsimpoukelli, M., Menick, J. L., Cabi, S., Eslami, S., Vinyals, O., and Hill, F. Multimodal few-shot learning with frozen language models. *Advances in Neural Information Processing Systems*, 34:200–212, 2021.

Ulmer, D., Meijerink, L., and Cinà, G. Trust issues: Uncertainty estimation does not enable reliable ood detection on medical tabular data. In *Machine Learning for Health*, pp. 341–354. PMLR, 2020.

Urban, C. J. and Gates, K. M. Deep learning: A primer for psychologists. *Psychological Methods*, 2021.

US Bureau of Labor Statistics. National longitudinal surveys of youth data set, 2023. URL https://www.bls.gov/nls/.

Vanschoren, J., Van Rijn, J. N., Bischl, B., and Torgo, L. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 2014.

Vergara, J. R. and Estévez, P. A. A review of feature selection methods based on mutual information. *Neural computing and applications*, 24:175–186, 2014.

Verma, S. and Rubin, J. Fairness definitions explained. In *Proceedings of the international workshop on software fairness*, pp. 1–7, 2018.

Wallace, E., Feng, S., Kandpal, N., Gardner, M., and Singh, S. Universal adversarial triggers for attacking and analyzing nlp. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

Wolberg, W., Mangasarian, O., Street, N., and Street, W. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5DW2B.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. In *International Conference on Learning Representations*, 2022.

Yamada, Y., Lindenbaum, O., Negahban, S., and Kluger, Y. Feature selection using stochastic gates. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

Yoon, J., Zhang, Y., Jordon, J., and van der Schaar, M. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

Zheng, Y., Dave, T., Mishra, N., and Kumar, H. Fairness in reciprocal recommendations: A speed-dating study. In *Adjunct publication of the 26th conference on user modeling, adaptation and personalization*, 2018.

Zhong, Z., Friedman, D., and Chen, D. Factual probing is [mask]: Learning vs. learning to recall. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021.

Zhou, K., Yang, J., Loy, C. C., and Liu, Z. Conditional prompt learning for vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022a.

Zhou, K., Yang, J., Loy, C. C., and Liu, Z. Learning to prompt for vision-language models. *International Journal of Computer Vision*, 2022b.

A. Additional Related Work

Tabular classification. Tabular datasets are the oldest and among the most widely used dataset types in machine learning (Borisov et al., 2021; Shwartz-Ziv & Armon, 2022). GBDTs (Friedman, 2001) build an ensemble of decision trees, with each tree fitting the residual of the loss from the previous tree. XGBoost (Chen & Guestrin, 2016) and CatBoost (Prokhorenkova et al., 2018) are two of the most widely-used and highest-performing GBDTs. Researchers have also explored many methods based on neural nets (Gorishniy et al., 2021; Huang et al., 2020; Kadra et al., 2021).

There is an active debate in the community on which family of methods perform best on tabular data: neural nets (Kadra et al., 2021; Arik & Pfister, 2021; Popov et al., 2020; Rubachev et al., 2022; Levin et al., 2023) or GBDTs (Shwartz-Ziv & Armon, 2022; Borisov et al., 2021; Gorishniy et al., 2021; Grinsztajn et al., 2022), with the exception of small datasets, on which TabPFN performs the best (McElfresh et al., 2023; Hollmann et al., 2023). Finally, sketching and feature selection methods have been extensively studied in prior works (Munteanu & Schwiegelshohn, 2018; Chandrashekar & Sahin, 2014).

Neural Processes and Prior-Data Fitted Networks. Prior-data Fitted Networks (PFNs) (Müller et al., 2022; Hollmann et al., 2023) are a recently-proposed paradigm for machine learning, which show that fast approximate Bayesian inference is possible by training a neural network to mimic the posterior predictive distribution (PPD) in a single forward pass using in-context learning (Müller et al., 2022; 2023; Nagler, 2023). PFNs were shown to yield state-of-the-art empirical performance on small tabular datasets (Hollmann et al., 2023; McElfresh et al., 2023). PFNs have been used in other applications, including Bayesian optimization (Müller et al., 2023), forecasting (Dooley et al., 2023), and learning curve extrapolation (Adriaensen et al., 2023).

PFNs are neural processes (NPs) (Müller et al., 2022). Recent advances in NP are similar in nature to our work, especially recent works that aim to scale attention-based methods to larger contexts. Feng et al. (2023a) propose Latent Bottlenecked Attentive Neural Processes (LBANPs), a transformer-based neural process which overcomes the quadratic complexity of transformers by encoding the context dataset into a constant number of latent vectors. Guo et al. (2023) propose Versatile Neural Processes (VNP), which increases the capability of NPs to handle compex signals by using a new bottleneck encoder. (Rastogi et al., 2023) introduces semi-parametric inducing point networks (SPIN), which can attend to a training set at inference time with linear complexity via inducing point methods. (Feng et al., 2023b) introduce Constant Memory Attention Block (CMAB), an attention block that is permutation-invariant and has constant memory complexity when computing the output, as well as Constant Memory Attentive Neural Processes (CMANPs), a NP that requires constant memory.

For additional related work, see Appendix A.

Prompt tuning. Researchers have extensively studied prompting techniques for large language models (LLMs) Liu et al. (2023). In such methods, one modifies the input into a *prompt*, and the LLM predicts the subsequent tokens, from which it derives the output predictions. *Prompt tuning* techniques typically start with a pretrained LLM and use the training data of a downstream task to find the best prompt for this task. 'Hard' prompt tuning involves finding the best natural language prompts using discrete search techniques (Jiang et al., 2020; Wallace et al., 2019), while 'soft' prompt tuning optimizes a prompt directly in the embedding space of the model (Lester et al., 2021; Zhong et al., 2021). Soft prompt tuning has also been applied to multi-modal models such as vision-language models (Zhou et al., 2022b;a; Tsimpoukelli et al., 2021).

B. Additional Experimental Details

First, we list all datasets used in our experiments. See Table 3.

Here, we give the extended details of our experimental setup from Section 5. For the baselines, we use the same hyperparameter search space as in prior work (McElfresh et al., 2023) (which is itself similar to other works (Kadra et al., 2021)). For TuneTables, we describe the method further in Appendix C.

B.1. Feature Selection and Sketching

Here, we consider sketching and feature selection for scaling TabPFN. We consider three sketching methods (*random*, *k-means*, and *CoreSet*) and three feature selection methods (*random*, *PCA*, *mutual information*) as described in Section 3. In addition to sketching, we consider two different methods for sampling class labels: *equal* and *proportional*. We limit our algorithmic comparison to TabPFN to CatBoost, which is the overall best-performing model in McElfresh et al. (2023). We compare all combinations of sketching and feature selection with both CatBoost and TabPFN; see Table 4. Interestingly, we

find that *random* sketching performs just as well as the more involved algorithms, k-means and *CoreSet*. On the other hand, PCA significantly outperforms mutual information and random feature selection methods, when the original number of features is large.

B.2. Additional Results

First, we present the large class table from Section 5. See Table 5.

In Table 1, we gave aggregate statistics for TuneTables and baselines across all datasets of size less than 50 000. Now, in Table 6, we present results for TuneTables, CatBoost, and XGBoost on all datasets individually, including datasets of size nearly two million.

Neural net comparison. In Table 1 and Table 6, we compared TuneTables to GBDTs and two other neural nets. Now, we compare TuneTables to two additional neural nets: MLP and TabTransformer (Huang et al., 2020) (for four total neural net baselines). Since transformer-based methods have a higher memory consumption, we obtained full results on 17 total datasets. See Table 7. We find that TuneTables substantially outperforms all other neural nets on average across all datasets, achieving the lowest average rank at 1.93, and achieving the highest average Z-Score of 0.85, far above the second-place neural net's value of 0.18.

Inference Time We give the full details for inference time; see Table 8.

B.3. Ablation study

We report our results ablating the core components of our methods: the prompt length, its use during training and inference, the use of ensembles, and the prompt tuning procedure itself (against regular finetuning). Beginning with the prompt length, we see in Table 9 that while some dataset results are not sensitive to the prompt length, others vary significantly, and generally enjoy longer prompts. Ablating the importance of having real-data context during inference (same table) we find that it is important for smaller datasets. On the larger datasets, the variant with no real data at all is better in specific cases. However, having no such data during inference, a model would perform better not having it during training as well. Introducing real data first during inference is usually harmless, but significantly deteriorates the results on specific cases. Fine tuning the entire model is not only significantly more memory-intensive but also underperforms TuneTables in terms of accuracy.

In Table 10 we show the results of our method with or without ensembles, and with or without context (in training and inference). Our ensembling strategy is beneficial in both cases. When using ensembles, the results with or without context are generally similar, although one variant may outperform the other on specific datasets.

C. TuneTables Additional Details

We present the entire TuneTables algorithm in Algorithm 1 (the fit and selection stages) and Algorithm 2 (the full algorithm, which uses Algorithm 1 as a subroutine).

Prompt tuning in TuneTables. While nearly any tuned prompt improves on the performance of zero-shot TabPFN for large-sample datasets, we find that there are many trade-offs involved in the particulars of the tuned prompt. We describe some of them below.

Hyperparameters and HPO. For the experiments Section 5, we use a fixed learning rate of 0.1, a batch size 1, and no gradient aggregation. We perform no HPO (beyond the light grid search in Algorithm 2) for any of the experiments described in the paper. We conduct all of our prompt tuning experiments on GCP, using a single Nvidia L4 TPU with 24GB VRAM for each experiment.

Size of tuned prompts. In this paper, we ablate the size of tuned prompts, considering prompts of size 10, 100 and 1000. The experimental results in Section 5 are reported on prompts of size 1000 (the same size as the dataset limit for TabPFN). We ablate this choice in Table 9, and we find that in settings where prompt tuning improves on TabPFN, larger prompts often outperform smaller ones.

Algorithm 1 Fit and Grid Search Algorithm

Require: Labeled dataset D, optimization objectives o, search space $search\_space$ (list of lists of hyperparameters, models, and feature subselection methods), maximum search time t
Ensure: Explored spaces and outputs

```
explored_spaces ← []
outputs ← []
i ← 0
while i < t do
    next_space ← select without replacement a new combination from search_space
    start ← CurrentTime()
    outputs ← Model(D, HParams, Features)
    explored_spaces ← next_space
    i ← i + (CurrentTime() − start)
end while
b, idx_b ← 0, −1
for out, idx_out in Enum(outputs) do
    for obj in o do
        if obj > b then
            b ← obj
            idx_b ← idx_out
        end if
    end for
end for
return explored_spaces[idx_b], outputs[idx_b]
```

Duration of training. We fit our tuned prompts for 30 epochs per prompt, regardless of prompt size. We also use 30 epochs for our fine-tuning experiments. When we fit both decoder and the tuned prompt (for class extension), we train for 100 epochs with a low learning rate. We use early stopping over individual ensemble members, and no early stopping for ensembles.

Evaluation of tuned prompts. We evaluate our tuned prompts once every 3 epochs, and for any given tuned prompt, we retain the weights associated with the highest validation accuracy.

Fine-tuned setting. In the fine-tuned setting, our parameters are identical to that of single-prompt prompt tuning, except that we use a much lower fixed learning rate of $1e-3$.

CT and NCT settings. We implement our tuned prompts by prepending randomly initialized vectors of identical dimension to the TabPFN encoder to the real data points fed to TabPFN as context during training.

In the **CT** ('context in training') setting, we continue to provide real data points as part of the training context; the quantity of real data provided for each batch is drawn with uniform probability from a random variable which ranges from zero to a fixed upper bound, which is passed to the model as a hyperparameter. Usually that upper bound is 1152, the default setting for TabPFN, unless the dataset is extremely small, in which case we select it according to the limitations of that dataset.

In the **NCT** ('no context in training') setting, no real data is provided as part of the training context; all data in the training set is used to fit the randomly initialized prompt. There are always a fixed number of data points provided, and it is the same for every batch. Usually that number is 128, unless the dataset is extremely small, in which case we select it according to the limitations of that dataset.

We note that it is also possible to evaluate models trained in either setting either with context (C) or without (NC). In our experiments, we always evaluate all models on both, and report the setting with higher validation accuracy at test time.

We find that on smaller datasets, the setting with context (C) sometimes outperforms the setting without context (NC), even after the prompt has been fitted, perhaps because the tuned prompt overfits on a small amount of available training data. However, when using ensembling, for datasets with more than 3000 samples in the training set, the NC setting is as good or better than the C setting. See Table 10. We leave further investigation of this phenomenon to future work.

Algorithm 2 Optimized Model Selection and Feature Selection Algorithm. FeatSS wraps standard feature subselection algorithms as described in Section 3. ZeroPad pads the number of features to exactly 100, following (Hollmann et al., 2023). Model variant ZS is the standard TabPFN model of (Hollmann et al., 2023). $PT + DEC$ co-trains prompt and decoder layer, as described in Section 4.2. $PT + ENS$ refers to prompt tuning with ensembling, described in Appendix C. CT and NCT settings are also defined in Appendix C.

Require: dataset D, model variants ZS, $PT + ENS$, $PT + DEC$, sample cutoff k, feature cutoff j, class cutoff l, maximum runtime t, feature selection algorithm list fs, HPO search space hl, optimization objective list o

$search\_space \leftarrow$ [hparams $= hl$, models $= []$, feature_subsel $= []$]
$d_f \leftarrow \#$features of D
$d_c \leftarrow \#$classes of D
$d_n \leftarrow \#$samples of D
if $d_f > j$ **then**
  $feature\_subsel \leftarrow$ [FeatSS$(j, d_f,$ alg) for alg in fs]
else if $d_f < j$ **then**
  $feature\_subsel \leftarrow$ [ZeroPad(j, d_f)]
else
  $feature\_subsel \leftarrow$ [Identity()]
end if
if $d_c > l$ **then**
  $models \leftarrow [PT + DEC(D)]$
else if $d_n < k$ **then**
  $models \leftarrow [ZS(D[:k])]$
else
  $models \leftarrow [ZS(D[:k]), PT + ENS(D, CT), PT + ENS(D, NCT)]$
end if
return FitAndGridSearch$(D, o, search_space, t)$

Ensembling over tuned prompts. While individual tuned prompts are already a substantial improvement on TabPFN for sample-large datasets, we find that these improvements are compounded if we ensemble over multiple tuned prompts.

We draw the inspiration for our ensembling approach from the internal ensembling used in TabPFN, which averages predictions over permutations of feature indices and label indices(Hollmann et al., 2023). For the results presented in Section 5, we ensemble over ten permutations of each dataset, averaging the top two ensemble member predictions (as measured by NC accuracy on the validation set in the NCT setting, or C accuracy in the CT setting). See Table 10 for ablation studies of the effectiveness of ensembling.

D. Summarization Details and Decision Boundaries of Prompt-Tuned TabPFN on Toy 2D Problems

For the experiments with prompts of only two examples, we prompt-tune one example per class for 500 epochs. The prompt is input directly into TabPFN without undergoing any preprocessing steps that the original data went through. Therefore, the tuned feature values do not correspond directly to the original values in the dataset. For the breast cancer dataset, it reaches 100% accuracy; for the diabetes dataset, it reaches 78.7% accuracy.

We also show decision boundaries of TabPFN and prompt-tuned TabPFN on toy 2d classification problems using scikit-learn (Pedregosa et al., 2011) in Fig. 5. The prompt contains four datapoints for each class. One can see how the points are tuned to recover the decision boundaries. Due to the very low prompt length and the low dataset size, the prompt-tuned decision boundaries are slightly less accurate than TabPFN (consistent with our results for small datasets in Table 9).

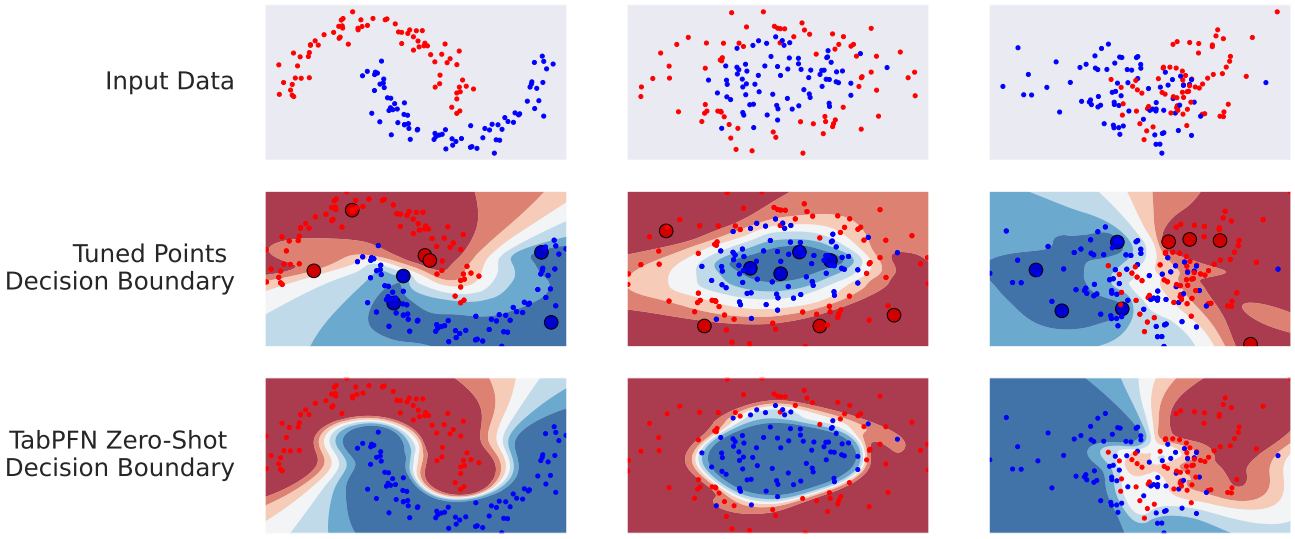

Figure 5. **Decision boundaries on 2D datasets.** Shown are prompt-tuned TabPFN decision boundaries as well as the regular zero-shot TabPFN decision boundaries. Larger dots in middle row represent the tuned points for the two classes.

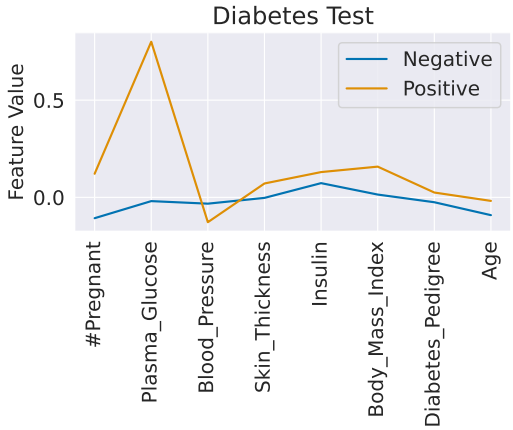

Figure 6. **Diabetes dataset (Smith et al., 1988) with high accuracies from just two samples.**

Table 3. List of all datasets used in Section 5 and Appendix B. Datasets in part one of the table were used in the core experiments for the paper. Datasets in part two were used for the class extension experiments. Datasets in part three were used for experiments in Section 6. The additional dataset in part four was used for experiments in Section 7. Datasets in the final part were used for the ablations on sketching and feature selection in Appendix B.

Dataset	num. classes	num. features	num. samples
click-prediction-small	2	11	1997410
poker-hand	10	10	1025009
agrawal1	2	9	1000000
BNG (labor)	2	16	1000000
airlines	2	7	539383
albert	2	78	425240
BNG (vote)	2	16	131072
connect-4	2	28	98050
higgs	3	42	67557
volkert	10	180	58310
riccardo	2	4296	20000
elevators	2	18	16599
eeg-eye-state	2	14	14980
har	6	561	10299
dilbert	5	2000	10000
robert	10	7200	10000
christine	2	1636	5418
bioresponse	2	1776	3151
kc-1	2	21	2109
car	4	6	1728
cmc	3	9	1473
blood-transfusion	2	4	748
balance-scale	3	4	625
climate	2	18	540
cylinder-bands	2	37	540
dresses-sales	2	12	500
colic	2	22	368
ecoli	8	7	336
heart-c	2	13	303
breast-cancer	2	9	286
walking-activity	22	4	149332
chess	18	6	28056
soybean	19	35	683
adult-census	2	15	32561
SpeedDating	2	121	8378
compas-two-years	2	12	4966
nlsy (national-longitudinal-survey-binary)	2	17	4908
Diabetes	322	10	442
skin-segmentation	2	3	245057
FM (Fashion-MNIST)	10	784	70000
CIFAR-10	10	3072	60000
gddc (gas-drift-different-concentrations)	6	129	13910
pendigits	10	16	10992
mfeat-factors	10	216	2000
mfeat-pixel	10	240	2000
semeion	10	256	1593
hill-valley	2	100	1212

Table 4. **Comparative performance of TabPFN and CatBoost with sketching, feature selection, and sampling methods.** We compare at a fixed feature size of 100 and a fixed sample size of 3000. When both models are limited to 3000 samples, TabPFN performs better on 12 of 17 datasets where significant differences exist. When CatBoost is allowed access to the entire training data, the win rate is identical. In most cases, random sample selection is sufficient for optimal performance. Both models benefit from PCA and mutual information dimension reduction when the feature space is large. **Bold** indicates the best-performing model(s).

	Full CatBoost	**Best** CatBoost	**Random** CatBoost	**Best** TabPFN	**Random** TabPFN	**SKT / FTS / SMP** CatBoost	**SKT / FTS / SMP** TabPFN
airlines_189354	**0.653**	0.637	0.637	0.594	0.589	RND / RND / PR	RND / RND / PR
albert_189356	**0.698**	0.657	0.657	0.64	0.64	RND / RND / PR	RND / RND / PR
CIFAR_10_167124	**0.434**	0.37	0.342	0.373	0.372	RND / PCA / PR	RND / RND / PR
connect-4_146195	**0.749**	0.716	0.716	0.66	0.659	RND / RND / PR	RND / RND / PR
eeg-eye-state_14951	0.832	0.808	0.806	**0.932**	**0.932**	RND / RND / PR	RND / RND / EQ
elevators_3711	0.855	0.838	0.838	**0.9**	**0.899**	RND / MUT / PR	RND / RND / PR
FM_146825	**0.843**	0.787	0.787	**0.835**	0.812	RND / RND / PR	RND / PCA / PR
gddc_9987	0.97	0.976	0.955	**0.994**	**0.993**	RND / PCA / EQ	RND / RND / PR
higgs_146606	**0.71**	0.684	0.684	0.665	0.661	RND / RND / PR	RND / RND / PR
hill-valley_145847	0.514	0.514	0.514	**0.56**	**0.56**	RND / RND / PR	RND / RND / PR
mfeat-factors_12	0.954	0.95	0.943	**0.973**	**0.973**	KMN / RND / EQ	RND / RND / PR
mfeat-pixel_146824	0.955	0.951	0.951	**0.971**	**0.97**	RND / RND / PR	RND / RND / PR
pendigits_32	0.972	0.966	0.964	**0.995**	**0.993**	RND / RND / PR	RND / RND / PR
poker-hand_9890	**0.664**	0.572	0.561	0.519	0.515	RND / RND / PR	RND / RND / PR
riccardo_168338	0.951	0.956	0.93	**0.991**	0.982	RND / PCA / EQ	RND / MUT / EQ
robert_168332	**0.446**	0.367	0.367	0.384	0.359	RND / RND / PR	RND / PCA / EQ
semeion_9964	0.887	0.869	0.863	**0.915**	**0.915**	RND / MUT / EQ	RND / RND / PR
ss_9965	**0.994**	0.989	0.987	**0.999**	**0.999**	RND / RND / PR	RND / RND / PR
volkert_168331	**0.608**	0.56	0.56	0.557	0.555	RND / RND / PR	RND / RND / PR

Table 5. **Comparison of algorithms over 8 datasets with a large number of classes.** TuneTables can effectively handle datasets with more classes than the ones used for pretraining, which was not possible with TabPFN. We report test accuracy. For each algorithm, we compute its mean accuracy, mean runtime, and mean rank in terms of accuracy. We also compute the mean Z-score, computed by normalizing the set of results on each dataset (by mean 0 std. 1), so that each dataset has the same weight, and averaging each algorithm's normalized performances. We see that TuneTables performs the best across all performance-oriented metrics. Fractional num. wins values are averaged over three splits per dataset, and reflect the presence of multi-way ties on certain splits.

Method	Mean Acc.	Mean Rank	Mean Z-Score	Std. Z-Score	Med. Z-Score	Num. Wins
XGBoost	0.882	2.438	0.879	0.081	1.024	4.000
TuneTables	0.780	2.583	0.372	0.305	0.646	2.333
KNN	0.800	3.688	0.178	0.084	0.150	0.333
rtdl_ResNet	0.787	3.854	-0.134	0.066	-0.031	0.667
CatBoost	0.791	4.521	-0.121	0.098	0.128	0.667
RandomForest	0.760	5.438	-0.644	0.075	-0.548	0.000
rtdl_MLP	0.655	5.479	-1.057	0.166	-1.382	0.000

Table 6. Comparison of the top-performing methods across 29 datasets. For each algorithm, we show its test accuracy and runtime (seconds). We also show the number of samples in each dataset. All datasets above the line were included in Table 1.

Dataset	Size	TabPFN		TuneTables		CatBoost		XGBoost	
		Acc.	Runtime	Acc.	Runtime	Acc.	Runtime	Acc.	Runtime
breast-cancer	286	0.765	29	0.770	29	0.770	37	0.724	4
heart-c	303	0.848	40	0.903	40	0.903	21	0.839	2
ecoli	336	0.848	30	0.843	30	0.892	73	0.833	17
colic	368	0.856	39	0.892	39	0.874	218	0.883	3
dresses-sales	500	0.578	41	0.580	41	0.640	40	0.640	5
cylinder-bands	540	0.800	41	0.846	41	0.827	94	0.864	6
climate	540	0.959	59	0.951	59	0.963	25	0.926	4
balance-scale	625	0.990	29	0.995	29	0.899	53	0.894	145
blood-transfusion	748	0.801	25	0.782	25	0.756	24	0.760	58
cmc	1473	0.554	91	0.556	91	0.561	78	0.561	13
kc-1	2109	0.862	168	0.856	168	0.856	49	0.859	6
bioresponse	3151	0.797	638	0.798	313	0.788	113	0.800	102
christine	5418	0.742	666	0.755	360	0.736	331	0.743	4
robert	10000	0.250	964	0.414	782	0.464	599	0.503	2118
dilbert	10000	0.922	761	0.992	1127	0.949	809	0.978	1144
har	10299	0.936	370	0.981	370	0.985	302	0.994	232
eeg-eye-state	14980	0.940	178	0.986	1574	0.907	28	0.946	18
elevators	16599	0.902	186	0.902	675	0.891	176	0.896	234
riccardo	20000	0.922	1395	0.995	418	0.997	692	0.998	735
volkert	58310	0.567	459	0.693	3744	0.666	230	0.703	604
higgs	67557	0.671	931	0.714	7968	0.724	80	0.725	46
connect-4	98050	0.668	696	0.817	6288	0.807	204	0.855	106
BNG (vote)	131072	0.968	1976	0.974	1733	0.975	42	0.975	16
albert	425240	0.642	2363	0.658	13150	0.706	1078	0.690	76
airlines	539383	0.600	2602	0.653	39713	0.664	155	0.673	184
BNG (labor)	1000000	0.937	5518	0.967	20757	0.971	226	0.970	179
agrawal1	1000000	0.948	5158	0.950	14788	0.951	142	0.951	98
poker-hand	1025009	0.531	2423	1.000	32452	0.912	5331	0.940	439
click-prediction-small	1997410	0.833	10421	0.837	23287	0.842	85	0.843	213

Table 7. **Comparison of neural nets across 17 datasets.** For each algorithm, we compute its different metrics of accuracy and rank. We also compute the mean Z-score, computed by normalizing the set of results on each dataset (by mean 0 std. 1), so that each dataset has the same weight, and averaging each algorithm's normalized performances. We see that TuneTables performs the best across all performance-oriented metrics. Fractional num. wins values are averaged over three splits per dataset, and reflect the presence of multi-way ties on certain splits.

Method	Mean Acc.	Med Acc.	Mean Rank	Med. Rank	Mean Z-Score	Med. Z-Score	Num. Wins
TuneTables	**0.81**	**0.85**	**1.93**	**1.0**	**0.85**	**0.91**	**9.67**
rtdl_ResNet	0.77	0.78	2.68	2.0	0.18	0.19	3.33
TabTransformer	0.76	0.73	3.05	3.0	-0.10	0.05	3.00
rtdl_MLP	0.73	0.76	3.55	4.0	-0.34	-0.55	0.83
MLP	0.72	0.74	3.79	4.0	-0.59	-0.55	0.17

Table 8. Comparison of inference times across datasets.

Dataset	N. classes	N. Feats	N. Samples	TabPFNs3000 3000 pts, ens.	TabPFNs3000 per1k	TuneTables-pt10-C 3000 pts	TuneTables-pt10-C per1k	TuneTables-pt10-NC 0 pts	TuneTables-pt10-NC per1k	TuneTables-pt1000-C 3000 pts	TuneTables-pt1000-C per1k	TuneTables-pt1000-NC 0 pts	TuneTables-pt1000-NC per1k
breast-cancer	2	9	286	0.951	3.325	0.103	0.36	0.103	0.36	0.119	0.416	0.109	0.381
heart-c	2	13	303	1.011	3.337	0.103	0.34	0.102	0.337	0.111	0.366	0.107	0.353
ecoli	8	7	336	0.99	2.946	0.094	0.28	0.092	0.274	0.094	0.28	0.091	0.271
colic	2	22	368	1.045	2.84	0.103	0.28	0.102	0.277	0.124	0.337	0.112	0.304
dresses-sales	2	12	500	1.024	2.048	0.098	0.196	0.09	0.18	0.106	0.212	0.092	0.184
climate	2	18	540	1.067	1.976	0.098	0.181	0.09	0.167	0.11	0.204	0.095	0.176
cylinder-bands	2	37	540	1.317	2.439	0.088	0.163	0.091	0.169	0.108	0.2	0.093	0.172
balance-scale	3	4	625	0.7	1.12	0.115	0.184	0.105	0.168	0.147	0.235	0.104	0.166
blood-transfusion	2	4	748	0.595	0.795	0.123	0.164	0.117	0.156	0.149	0.199	0.107	0.143
cmc	3	9	1473	1.485	1.008	0.099	0.067	0.093	0.063	0.155	0.105	0.093	0.063
kc-1	2	21	2109	2.623	1.244	0.12	0.057	0.11	0.052	0.203	0.096	0.116	0.055
bioresponse	2	1776	3151	5.623	1.785	0.23	0.073	0.116	0.037	0.338	0.107	0.114	0.036
christine	2	1636	5418	5.392	0.995	0.486	0.09	0.183	0.034	0.725	0.134	0.167	0.031
dilbert	5	2000	10000	9.355	0.936	0.74	0.074	0.204	0.02	1.126	0.113	0.232	0.023
robert	10	7200	10000	9.369	0.937	0.718	0.072	0.202	0.02	1.143	0.114	0.208	0.021
har	6	561	10299	9.399	0.913	0.804	0.078	0.219	0.021	1.27	0.123	0.236	0.023
eeg-eye-state	2	14	14980	8.235	0.55	1.086	0.072	0.245	0.016	1.683	0.112	0.284	0.019
elevators	2	18	16599	8.501	0.512	1.169	0.07	0.278	0.017	1.799	0.108	0.294	0.018
riccardo	2	4296	20000	5.994	0.3	1.369	0.111	0.325	0.018	2.21	0.111	0.358	0.018
volkert	10	180	58310	17.448	0.299	3.931	0.067	0.772	0.013	6.336	0.109	0.897	0.015
higgs	3	42	67557	21.991	0.326	6.449	0.095	1.356	0.02	10.404	0.154	1.517	0.022
connect-4	2	28	98050	19.966	0.204	4.491	0.046	0.922	0.009	7.064	0.072	0.992	0.01
BNG(vote)	2	16	131072	28.229	0.215	8.898	0.068	1.636	0.012	14.25	0.109	1.95	0.015
albert	2	78	425240	77.546	0.182	27.988	0.066	5.748	0.014	47.855	0.113	6.378	0.015
airlines	2	7	539383	85.146	0.158	36.797	0.068	6.751	0.013	61.477	0.114	8.123	0.015
agrawal1	2	9	1000000	193.083	0.193	70.665	0.071	12.129	0.012	112.533	0.113	14.944	0.015
BNG(labor)	2	16	1000000	190.81	0.191	69.374	0.069	12.094	0.012	108.211	0.108	14.666	0.015
poker-hand	10	10	1025009	198.151	0.193	71.605	0.07	13.097	0.013	112.214	0.109	15.199	0.015
click-prediction-small	2	11	1997410	384.192	0.192	133.212	0.067	24.964	0.012	227.546	0.114	30.982	0.016
AVERAGE			222079.517	44.525	1.109	15.212	0.124	2.839	0.087	24.814	0.162	3.402	0.09

Table 9. **Ablation study.** Comparison of the different variants of our method and the entire backbone fine-tuning (TabPFN-FT) across 29 datasets. For each variant, we show its test accuracy. TabPFN-PT stands for our prompt-tuning methods, with the hyperparameters specified below it. TuneTables is our full method described in Section 4.

	TabPFN-FT	TabPFN-PT	TabPFN-PT	TabPFN-PT	TabPFN-PT	TabPFN-PT	TabPFN-PT	TuneTables
Context in Training	✓			✓	✓	✓	✓	Varied
Context in Inference	✓	✓		✓		✓	✓	Varied
Tuned prompt length	N/A	1000	1000	1000	1000	100	10	Varied
Agrawal1	0.946	0.948	0.950	0.950	0.946	0.950	0.949	0.950
BNG(labor)	0.931	0.961	0.965	0.965	0.966	0.965	0.960	0.967
BNG(vote)	0.964	0.971	0.974	0.973	0.973	0.974	0.971	0.974
Bioresponse	0.754	0.756	0.760	0.747	0.668	0.729	0.748	0.798
Click_prediction_small	0.834	0.834	0.834	0.835	0.835	0.835	0.833	0.837
airlines	0.588	0.628	0.645	0.643	0.639	0.640	0.628	0.653
albert	0.605	0.646	0.648	0.651	0.651	0.657	0.648	0.658
balance-scale	0.937	0.909	0.944	0.940	0.456	0.932	0.933	0.995
blood-transfusion	0.787	0.760	0.785	0.764	0.691	0.773	0.771	0.782
breast-cancer	0.730	0.743	0.713	0.741	0.661	0.724	0.718	0.770
christine	0.711	0.723	0.716	0.714	0.685	0.712	0.700	0.755
climate-model	0.947	0.938	0.938	0.941	0.666	0.923	0.938	0.951
cmc	0.542	0.525	0.559	0.540	0.436	0.532	0.548	0.556
colic	0.888	0.871	0.880	0.816	0.563	0.870	0.842	0.892
connect-4	0.678	0.787	0.810	0.795	0.804	0.767	0.679	0.817
cylinder-bands	0.842	0.791	0.759	0.870	0.556	0.820	0.856	0.846
dilbert	0.825	0.934	0.941	0.895	0.847	0.869	0.834	0.992
dresses-sales	0.550	0.573	0.533	0.567	0.553	0.560	0.577	0.580
ecoli	0.814	0.765	0.775	0.823	0.314	0.833	0.823	0.843
eeg-eye-state	0.579	0.626	0.651	0.651	0.654	0.616	0.601	0.986
elevators	0.894	0.895	0.899	0.896	0.854	0.896	0.892	0.902
har	0.940	0.980	0.982	0.971	0.946	0.962	0.945	0.981
heart-c	0.863	0.839	0.860	0.806	0.718	0.838	0.855	0.903
higgs	0.638	0.700	0.703	0.696	0.700	0.698	0.675	0.714
kc1	0.843	0.848	0.843	0.853	0.218	0.860	0.851	0.856
poker-hand	0.601	0.479	0.993	0.982	0.985	0.690	0.572	1.000
riccardo	0.918	0.990	0.993	0.990	0.990	0.960	0.936	0.995
robert	0.303	0.394	0.404	0.386	0.369	0.354	0.311	0.414
volkert	0.487	0.621	0.647	0.631	0.649	0.605	0.540	0.693

Table 10. **Ensembled models outperform models trained on a single tuned prompt; with ensembling, the training and testing without real-data context (NC) setting matches or exceeds the setting with context (C).** Runs with prompt tuned only once are noted as TabPFN-PT, and ensembles of such runs as TabPFN-PT-Ens. Although the improvements are often quite small, we find that ensembles generally outperform single tuned prompts in both C and NC settings. We also find that ensembles trained without additional real data context at train time or test time are as good or better than ensembles trained and tested with real data context on datasets larger than 3000 samples.

Dataset	TabPFN-PT-C	TabPFN-PT-Ens-C	TabPFN-PT-NC	TabPFN-PT-Ens-NC
agrawal1	0.949	**0.95**	**0.95**	0.949
airlines	0.645	**0.649**	0.645	0.646
albert	0.657	**0.66**	0.648	**0.66**
balance-scale	0.921	0.968	**0.984**	0.952
bioresponse	0.763	**0.795**	0.776	0.776
blood-transfusion	0.813	**0.84**	0.827	0.747
BNG (labor)	0.965	0.966	0.965	**0.967**
BNG (vote)	0.974	0.976	0.975	**0.977**
breast-cancer	**0.793**	**0.793**	0.759	0.69
car	0.96	0.971	**0.977**	0.965
christine	**0.76**	0.738	0.734	0.756
click-prediction-small	0.834	**0.836**	0.834	**0.836**
climate	0.926	0.944	0.963	**0.981**
cmc	0.534	**0.581**	0.547	0.446
colic	0.811	**0.892**	0.865	0.865
connect-4	0.796	**0.814**	0.808	0.812
cylinder-bands	0.815	0.815	**0.926**	0.778
dilbert	0.87	0.951	0.948	**0.968**
dresses-sales	0.6	**0.68**	0.66	0.6
eeg-eye-state	0.74	0.977	0.666	**0.983**
elevators	0.903	**0.908**	0.902	0.902
har	0.94	0.984	0.976	**0.987**
heart-c	0.871	**0.903**	0.871	0.871
higgs	0.695	**0.712**	0.709	0.709
kc-1	0.867	**0.872**	0.867	**0.872**
poker-hand	**1**	**1**	0.992	**1**
riccardo	0.991	**0.996**	0.994	**0.996**
robert	0.391	0.415	0.421	**0.444**
volkert	0.633	0.665	0.658	**0.672**