TabPFN Unleashed: A Scalable and Effective Solution to Tabular Classification Problems

Anonymous Authors[1]

Abstract

TabPFN has emerged as a promising in-context learning model for tabular data, capable of directly predicting the labels of test samples given labeled training examples. It has demonstrated competitive performance, particularly on small-scale classification tasks. However, despite its effectiveness, TabPFN still requires further refinement in several areas, including handling high-dimensional features, aligning with downstream datasets, and scaling to larger datasets. In this paper, we revisit existing variants of TabPFN and observe that most approaches focus either on reducing bias or variance, often neglecting the need to address the other side, while also increasing inference overhead. To fill this gap, we propose BETA (**B**agging and **E**ncoder-based Fine-tuning for **T**abPFN **A**daptation), a novel and effective method designed to *minimize both bias and variance*. To reduce bias, we introduce a lightweight encoder to better align downstream tasks with the pre-trained TabPFN. By increasing the number of encoders in a lightweight manner, BETA mitigate variance, thereby further improving the model's performance. Additionally, bootstrapped sampling is employed to further reduce the impact of data perturbations on the model, all while maintaining computational efficiency during inference. Our approach enhances TabPFN's ability to handle high-dimensional data and scale to larger datasets. Experimental results on over 200 benchmark classification datasets demonstrate that BETA either outperforms or matches state-of-the-art methods.

1. Introduction

Tabular data is one of the most widely used data formats across various domains, including finance (Cao & Tay, 2001), healthcare (Hassan et al., 2020), e-commerce (Nederstigt et al., 2014), and medical analysis (Schwartz et al., 2007; Subasi, 2012). Despite its ubiquity, modeling tabular data with deep learning methods remains a challenge due to its heterogeneous nature (Borisov et al., 2024). Yet recent advancements have led to the development of tabular foundation models (van Breugel & van der Schaar, 2024) , such as TabPFN (Tabular Prior-Fitted Networks) (Hollmann et al., 2023; 2025). TabPFN operates in two stages: *pre-training* and *inference*. During the pre-training stage, the model is pre-trained on a diverse set of synthetic datasets. In the inference stage, given a new task and a set of labeled examples as a "prompt," TabPFN directly predicts the labels of test samples using in-context learning, without requiring further parameter updates. This approach enables TabPFN to achieve performance comparable to or even surpass tree-based methods, particularly on small tabular datasets (McElfresh et al., 2023).

TabPFN has shown potential across a wide range of applications, including tabular data generation (Ma et al., 2024a), data augmentation (Margeloiu et al., 2024), and time series forecasting (Hoo et al., 2025). These use cases highlight the versatility of TabPFN, positioning it as a model worthy of further exploration. However, alongside application-driven studies, there is a growing interest in improving TabPFN's performance from various perspectives (Feuer et al., 2024; Thomas et al., 2024; Xu et al., 2025; den Breejen et al., 2024; Ma et al., 2024b). While previous research has reported performance improvements, the underlying reasons driving these gains remain unclear. These advancements are often fragmented, with each approach focusing on a specific aspect, and some methods even sacrifice efficiency for enhanced performance, without a comprehensive understanding of how to improve TabPFN systematically.

In this paper, we adopt the bias-variance decomposition framework introduced by Nagler (2023) to analyze the generalization error of TabPFN and its variants. This framework allows us to revisit and categorize existing methods, revealing that performance improvements typically arise from addressing either bias or variance. However, these approaches often neglect the other aspect, leading to suboptimal performance. Therefore, there is a need for a method that simultaneously addresses both bias and variance to improve performance.

To this end, we propose a novel, efficient, and scalable approach, BETA. Our method enhances TabPFN's per-

formance by introducing a *fine-tuning* stage, enabling parameter-efficient adaptation that aligns the downstream dataset distribution with the pre-trained TabPFN to mitigate bias. To further reduce variance BETA maps raw data into multiple latent spaces and employs computationally efficient bootstrapped sampling during inference. Beyond performance improvements, BETA offers a lightweight and scalable solution that effectively handles high-dimensional data and large datasets while maintaining inference efficiency.

BETA improves performance by refining both the fine-tuning and inference stages to reduce bias and variance. In the fine-tuning phase, we employ a lightweight encoder module as an input feature adapter, which transforms datasets with arbitrary dimensionality into multiple fixed-dimensional representations, thereby naturally enabling dimensionality reduction. To further enhance generalization, we integrate Batch Ensemble (Wen et al., 2020; Gorishniy et al., 2025) to increase the number of encoders in a lightweight manner, which introduces diversity in learned representations and reduces variance. These enhancements enable BETA to improve robustness and scalability, ensuring better adaptation to downstream datasets. In the inference phase, we introduce bootstrapped sampling, a technique that has been largely overlooked in previous TabPFN variants. By generating multiple subsets of the dataset as support sets for the context composition, BETA reduces variance and improves robustness. Furthermore, BETA seamlessly integrates with Error-Correcting Output Codes (ECOC) to effectively handle multiclass classification tasks with more than 10 classes.

Experimental results on multiple benchmark datasets, including over 200 classification tasks, demonstrate that our method significantly improves TabPFN's performance. We describe our main contributions below.

1. We introduce an adaptation method for TabPFN that addresses key limitations related to dataset size, high-dimensional features, and multiclass classification tasks.
2. By analyzing the generalization error of existing TabPFN variants through bias-variance decomposition and experiments on real-world datasets, we developed BETA, a method that effectively mitigates both bias and variance.
3. We achieve state-of-the-art performance on the largest benchmark datasets to date, demonstrating the robustness and scalability of our method for real-world tabular tasks.

Remark. We have noticed that the latest release of TabPFN-v2 (Hollmann et al., 2025) has partially alleviated some of the limitations previously discussed. Specifically, TabPFN-v2 incorporates design improvements that enable it to handle larger datasets and more features. However, it is important to highlight that TabPFN-v2 is a concurrent work, and while it partially mitigates these limitations, it does not fully resolve the challenges posed by dataset size and feature count. Thus, many of the improvements proposed in this paper are general enhancements that can complement TabPFN-v2 and potentially further its applicability to a broader range of tasks.

2. BETA

To address both bias and variance issues of TabPFN observed during the previous experiments, we propose a unified strategy for improving the performance of TabPFN. In addition to the inference stage, we introduce a fine-tuning stage. Our approach improves performance in both fine-tuning and inference, as shown in Figure 1. During fine-tuning, we refine input representations to better align the downstream data distribution with the pre-trained TabPFN, reducing both bias and variance. In the inference stage, we incorporate bootstrapped sampling to further reduce variance without additional computational overhead.

Minimizing Bias with Encoder-Based *Fine-Tuning*. Our analysis in subsection B.3 highlights the importance of minimizing bias for improved generalization. To achieve this, we introduce a lightweight encoder while keeping the pre-trained TabPFN parameters frozen. This encoder transforms raw input features into a latent space that better aligns the model's prior with the downstream data distribution.

Let $\boldsymbol{x}_i \in \mathbb{R}^d$ denote the raw feature vector, and let $\boldsymbol{E}_\Phi$ represent the encoder with parameters Φ. To enhance expressiveness and incorporate nonlinearity, these layers are constructed using a sequence of operations as described by Gorishniy et al. (2021):

$$\boldsymbol{E}_\Phi(\boldsymbol{x}) = \text{Linear}\left(\text{Dropout}\left(\text{ReLU}\left(\text{Linear}\left(\boldsymbol{x}\right)\right)\right)\right). \quad (1)$$

This transformation can be extended by stacking multiple such blocks, allowing the encoder to learn hierarchical representations suited for complex downstream tasks.

Given a query set $\boldsymbol{X}_q$ and a support set $(\boldsymbol{X}_s, \boldsymbol{y}_s)$, we define their respective latent representations as:

$$\boldsymbol{Z}_q = \boldsymbol{E}_\Phi(\boldsymbol{X}_q), \quad \boldsymbol{Z}_s = \boldsymbol{E}_\Phi(\boldsymbol{X}_s). \quad (2)$$

The posterior predictive distribution over the query labels is then expressed as:

$$q_\theta\left(\boldsymbol{y}_q \mid \boldsymbol{Z}_q, (\boldsymbol{Z}_s, \boldsymbol{y}_s)\right). \quad (3)$$

This formulation ensures that the encoder effectively maps raw features into a structured space, enabling TabPFN to better capture relevant patterns while mitigating bias.

This fine-tuning method provides a lightweight and effective approach to reduce bias by better aligning the model's prior with the downstream tasks. It enables end-to-end fine-tuning, facilitating a more efficient adaptation of the pre-trained TabPFN to the downstream tasks.

Mitigating Variance with Multiple Encoders. To further mitigate variance, we introduce multiple encoders, each

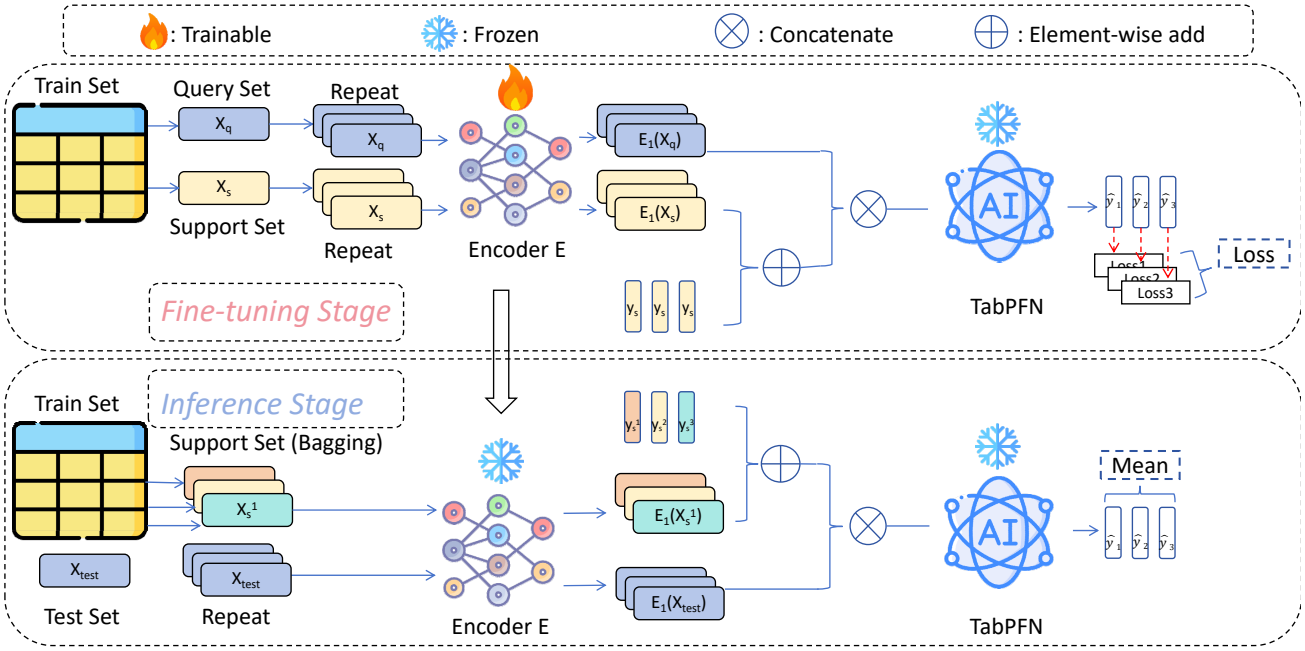


Figure 1: Overview of the proposed method, BETA, which consists of the *fine-tuning stage* and the *inference stage*.

learning a distinct transformation of the input data. By jointly training these encoders while keeping the pre-trained TabPFN parameters θ frozen, the model captures diverse feature representations, reducing variance.

For each encoder$k$, the support and query set representations are encoded as follows:

$$\boldsymbol{Z}_{\mathrm{s}}^{(k)} = \left[\boldsymbol{E}_{\Phi}^{(k)}(\boldsymbol{x}_{\mathrm{s}}^{(i)})\right]_{i=1}^{N_s}, \quad \boldsymbol{Z}_{q}^{(k)} = \left[\boldsymbol{E}_{\Phi}^{(k)}(\boldsymbol{x}_{q}^{(i)})\right]_{i=1}^{N_q},$$

where N_s and N_q denote the number of samples in the support set and the query set, respectively. The multiple encoders are trained jointly by minimizing the sum of the individual losses across all encoders. The optimization objective for the fine-tuning phase is:

$$\min_{\Phi} \mathcal{L}_{\text{total}} = -\sum_{k=1}^{K} \log\left(q_\theta\left(\boldsymbol{y}_q \mid \boldsymbol{Z}_{\mathrm{q}}^{(k)}, \left(\boldsymbol{Z}_{\mathrm{s}}^{(k)}, \boldsymbol{y}_s\right)\right)\right). \tag{4}$$

Minimizing $\mathcal{L}_{\text{total}}$ in Equation 4 jointly trains all encoders to generate TabPFN-compatible latent representations, with the different encoder initializations encouraging diverse representations and thereby reducing variance and providing more stable predictions.

Enhancing Performance with Batch Ensemble for Computational Efficiency. To further enhance performance without introducing additional computational cost, we integrate the Batch Ensemble technique into the encoder. Specifically, we replace the linear layers in $\boldsymbol{E}_{\Phi}$ with Batch Ensemble versions, allowing the model to maintain diversity while avoiding the overhead of training multiple independent encoders. This technique introduces shared weight matrices and member-specific scaling factors, reducing the number of trainable parameters while preserving the benefits of ensembling.

The output of the k-th base model for encoder layer l is given by:

$$l_k(\boldsymbol{x}) = s_k \odot (W(r_k \odot \boldsymbol{x})) + b_k \tag{5}$$

where W is shared across all base models, and r_k, s_k, b_k are specific to each base model (Wen et al., 2020).

Bootstrapped Sampling for Variance Reduction in *Inference Stage*. To further reduce variance during inference, we apply bootstrapped sampling, generating random subsets of the training set as support sets. These are used to compute predictions across multiple encoders, and their aggregation stabilizes the final output without additional computational overhead. For each encoder k, the bootstrapped support set is $D_{\text{bootstrap}}^{(k)} = (\boldsymbol{X}_{\text{bootstrap}}^{(k)}, \boldsymbol{Y}_{\text{bootstrap}}^{(k)})$. The final prediction is obtained by averaging over all encoders:

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} q_\theta\left(\boldsymbol{E}_{\Phi}(\boldsymbol{x}_{\text{test}}), \left(\boldsymbol{E}_{\Phi}^{(k)}(\boldsymbol{X}_{\text{bootstrap}}^{(k)}), \boldsymbol{Y}_{\text{bootstrap}}^{(k)}\right)\right).$$

This approach reduces variance while maintaining computational efficiency, ensuring scalability and robustness.

Expanding to MultiClass Tasks Beyond 10 Classes. To address TabPFN's limitation in handling tasks with more than 10 classes, we integrate an Error-Correcting Output Code (ECOC) (Dietterich & Bakiri, 1995) strategy. Instead of training separate classifiers, distinct encoders within a single model handle individual binary classification tasks, en-

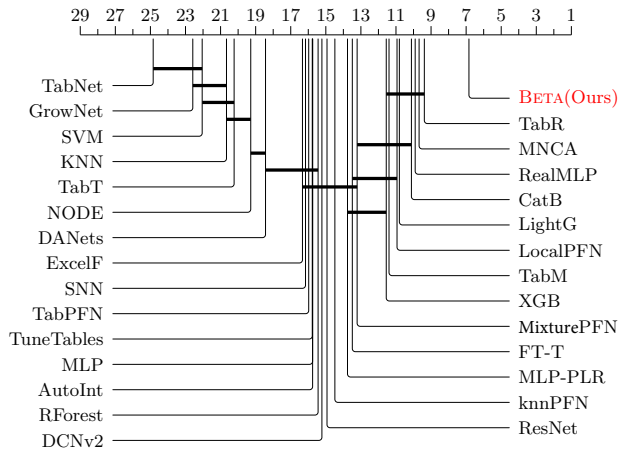


Figure 2: The critical difference diagrams based on the Wilcoxon-Holm test with a significance level of 0.05 to detect pairwise significance for TALENT datasets with fewer than 10 classes.

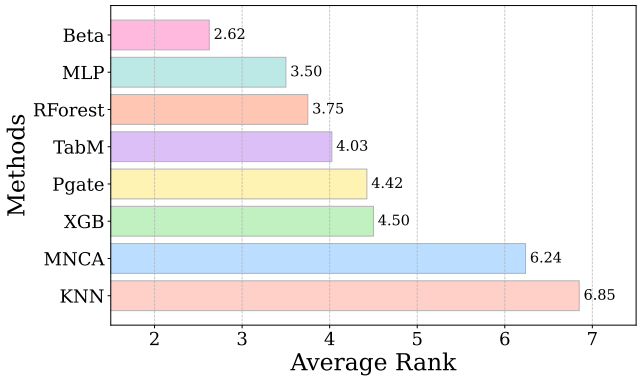


Figure 3: Average ranks of methods on 17 high-dimensional datasets. We compare BETA with TabM, KNN, MLP, XGBoost (XGB), RandomForest (RForest), ProtoGate (Pgate) (Jiang et al., 2024), and ModernNCA (MNCA). Lower ranks indicate better performance.

abling efficient multiclass classification without additional computational overhead.

Summary of Our Approach. Our method reduces bias using a lightweight encoder to align input representations with the pre-trained TabPFN, which is effective for high-dimensional data. Variance is reduced through Bagging with bootstrapped sampling, enabling better generalization on large datasets. For multiclass tasks, we integrate Error-Correcting Output Codes (ECOC) to efficiently handle more than two classes. Additionally, we preserve PFN-style batching, ensuring inference efficiency and scalability for large-scale and high-dimensional datasets.

3. Experiments

3.1. BETA: State-of-the-Art Performance

We conducted pairwise significance testing using the Wilcoxon-Holm test (Demsar, 2006) among BETA and all the compared methods. To ensure a fair comparison, we selected 186 datasets from TALENT with fewer than 10 classes, as TabPFN and its variants are not capable of handling datasets with more than 10 classes. From Figure 2, it is evident that BETA outperforms other methods, **even without hyper-parameter tuning**. This includes methods based on nearest neighbors such as TabR and ModernNCA, ensemble-based approaches like TabM, traditional tree models, and other TabPFN variants. These results underscore the significant potential of pre-trained models for tabular data and demonstrate the effectiveness of our proposed method in adapting to downstream datasets.

Handling High-Dimensional Datasets. To assess the effectiveness of BETA on high-dimensional datasets, we conducted experiments on 20 datasets with extremely high feature dimensions, as detailed in Table 2. These datasets were selected to evaluate the scalability and adaptability of different methods in complex feature spaces. The average ranks of the compared methods are summarized in Figure 3. The results show that BETA achieves the best performance, attaining the lowest average rank and outperforming all other methods. TabM and MLP also demonstrate competitive results, ranking second and third, respectively. In contrast, traditional models such as RandomForest and XGBoost, as well as deep learning-based ModernNCA, exhibit lower ranks, highlighting their limitations in high-dimensional settings. Due to memory constraints, we were unable to compare with methods such as FT-T and TabR in this setting.

4. Conclusion

In this paper, we propose BETA, a novel approach that enhances the performance of TabPFN by simultaneously addressing both bias and variance. Through a combination of lightweight encoder-based fine-tuning and bootstrapped sampling, BETA significantly improves TabPFN's adaptability to high-dimensional, large-scale, and multiclass classification tasks. Our method efficiently reduces bias by aligning downstream data distributions with the pre-trained TabPFN and reduces variance through diverse latent representations and robust inference techniques. Experimental results on over 200 benchmark datasets demonstrate that BETA consistently outperforms or matches state-of-the-art methods, highlighting its potential to handle complex tabular data tasks with enhanced scalability and robustness. These contributions provide a scalable and effective solution for leveraging TabPFN in real-world applications, ensuring its success across a broad range of tabular data challenges.

References

Aghajanyan, A., Gupta, S., and Zettlemoyer, L. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL/IJCNLP (1)*, pp. 7319–7328. Association for Computational Linguistics, 2021.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, pp. 2623–2631, 2019.

Alan, J., Tennison, L., Jonathan, C., Fergus, I., and van der Schaar Mihaela. Tangos: Regularizing tabular neural networks through gradient orthogonalization and specialization. In *ICLR*, 2023.

Arik, S. Ö. and Pfister, T. Tabnet: Attentive interpretable tabular learning. In *AAAI*, pp. 6679–6687, 2021.

Badirli, S., Liu, X., Xing, Z., Bhowmik, A., and Keerthi, S. S. Gradient boosting neural networks: Grownet. *CoRR*, abs/2002.07971, 2020.

Barry, B. and Ronny, K. Adult. UCI Machine Learning Repository, 1996.

Basri, R., Galun, M., Geifman, A., Jacobs, D. W., Kasten, Y., and Kritchman, S. Frequency bias in neural networks for input of non-uniform density. In *ICML*, pp. 685–694, 2020.

Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. Deep neural networks and tabular data: A survey. *IEEE Transactions Neural Networks and Learning Systems*, 35(6):7499–7519, 2024.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *NeurIPS*, pp. 1877–1901, 2020.

Cao, L. and Tay, F. E. H. Financial forecasting using support vector machines. *Neural Computing and Applications*, 10(2):184–192, 2001.

Chen, J., Liao, K., Wan, Y., Chen, D. Z., and Wu, J. Danets: Deep abstract networks for tabular data classification and regression. In *AAAI*, pp. 3930–3938, 2022.

Chen, J., Liao, K., Fang, Y., Chen, D., and Wu, J. Tabcaps: A capsule neural network for tabular data classification with bow routing. In *ICLR*, 2023a.

Chen, J., Zhang, A., Shi, X., Li, M., Smola, A., and Yang, D. Parameter-efficient fine-tuning design spaces. In *ICLR*, 2023b.

Chen, J., Yan, J., Chen, Q., Chen, D. Z., Wu, J., and Sun, J. Can a deep learning model be a sure bet for tabular prediction? In *KDD*, pp. 288–296, 2024.

Chen, K.-Y., Chiang, P.-H., Chou, H.-R., Chen, T.-W., and Chang, T.-H. Trompt: Towards a better deep neural network for tabular data. In *ICML*, pp. 4392–4434, 2023c.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *KDD*, pp. 785–794, 2016.

Demsar, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7: 1–30, 2006.

den Breejen, F., Bae, S., Cha, S., and Yun, S. Why in-context learning transformers are tabular data classifiers. *CoRR*, abs/2405.13396, 2024.

Dietterich, T. G. and Bakiri, G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

Feuer, B., Hegde, C., and Cohen, N. Scaling tabpfn: Sketching and feature selection for tabular prior-data fitted networks. *CoRR*, abs/2311.10609, 2023.

Feuer, B., Schirrmeister, R. T., Cherepanova, V., Hegde, C., Hutter, F., Goldblum, M., Cohen, N., and White, C. Tunetables: Context optimization for scalable prior-data fitted networks. *CoRR*, abs/2402.11137, 2024.

Fu, Z., Yang, H., So, A. M., Lam, W., Bing, L., and Collier, N. On the effectiveness of parameter-efficient fine-tuning. In *AAAI*, pp. 12799–12807, 2023.

Gardner, J., Perdomo, J. C., and Schmidt, L. Large scale transfer learning for tabular data via language modeling. *CoRR*, abs/2406.12031, 2024.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. In *NeurIPS*, pp. 18932–18943, 2021.

Gorishniy, Y., Rubachev, I., and Babenko, A. On embeddings for numerical features in tabular deep learning. In *NeurIPS*, pp. 24991–25004, 2022.

Gorishniy, Y., Rubachev, I., Kartashev, N., Shlenskii, D., Kotelnikov, A., and Babenko, A. Tabr: Tabular deep learning meets nearest neighbors in 2023. In *ICLR*, 2024.

Gorishniy, Y., Kotelnikov, A., and Babenko, A. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. In *ICLR*, 2025.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS*, pp. 507–520, 2022.

Han, Z., Gao, C., Liu, J., Zhang, J., and Zhang, S. Q. Parameter-efficient fine-tuning for large models: A comprehensive survey. *CoRR*, abs/2403.14608, 2024.

Hassan, M. R., Al-Insaif, S., Hossain, M. I., and Kamruzzaman, J. A machine learning approach for prediction of pregnancy outcome following IVF treatment. *Neural Computing and Applications*, 32(7):2283–2297, 2020.

He, H., Cai, J., Zhang, J., Tao, D., and Zhuang, B. Sensitivity-aware visual parameter-efficient fine-tuning. In *ICCV*, pp. 11791–11801. IEEE, 2023.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. Tabpfn: A transformer that solves small tabular classification problems in a second. In *ICLR*, 2023.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. Accurate predictions on small data with a tabular foundation model. *Nature*, 01 2025.

Holzmüller, D., Grinsztajn, L., and Steinwart, I. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. In *NeurIPS*, 2024.

Hoo, S. B., Müller, S., Salinas, D., and Hutter, F. The tabular foundation model tabpfn outperforms specialized time series forecasting models based on simple features. *CoRR*, abs/2501.02945, 2025.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In *ICML*, pp. 2790–2799, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022.

Huang, X., Khetan, A., Cvitkovic, M., and Karnin, Z. S. Tabtransformer: Tabular data modeling using contextual embeddings. *CoRR*, abs/2012.06678, 2020.

Jiang, X., Margeloiu, A., Simidjievski, N., and Jamnik, M. Protogate: Prototype-based neural networks with global-to-local feature selection for tabular biomedical data. In *ICML*, 2024.

Kasneci, G. and Kasneci, E. Enriching tabular data with contextual LLM embeddings: A comprehensive ablation study for ensemble classifiers. *CoRR*, abs/2411.01645, 2024.

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 *NIPS*, pp. 3146–3154, 2017.

Kim, M. J., Grinsztajn, L., and Varoquaux, G. CARTE: pretraining and transfer for tabular learning. In *ICML*, pp. 23843–23866, 2024.

Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S. Self-normalizing neural networks. In *NIPS*, pp. 971–980, 2017.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In *EMNLP (1)*, pp. 3045–3059. Association for Computational Linguistics, 2021.

Li, X. L. and Liang, P. Prefix-tuning: Optimizing continuous prompts for generation. In *ACL/IJCNLP (1)*, pp. 4582–4597. Association for Computational Linguistics, 2021.

Lialin, V., Deshpande, V., and Rumshisky, A. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *CoRR*, abs/2303.15647, 2023.

Liu, L., Fard, M. M., and Zhao, S. Distribution embedding networks for generalization from a diverse set of classification tasks. *Transactions on Machine Learning Research*, 2022, 2022.

Liu, S., Cai, H., Zhou, Q., and Ye, H. TALENT: A tabular analytics and learning toolbox. *CoRR*, abs/2407.04057, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Ma, J., Dankar, A., Stein, G., Yu, G., and Caterini, A. L. Tabpfgen - tabular data generation with tabpfn. *CoRR*, abs/2406.05216, 2024a.

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

Ma, J., Thomas, V., Yu, G., and Caterini, A. L. In-context data distillation with tabpfn. *CoRR*, abs/2402.06971, 2024c.

Mao, Y., Mathias, L., Hou, R., Almahairi, A., Ma, H., Han, J., Yih, S., and Khabsa, M. Unipelt: A unified framework for parameter-efficient language model tuning. In *ACL (1)*, pp. 6253–6264, 2022.

Margeloiu, A., Bazaga, A., Simidjievski, N., Liò, P., and Jamnik, M. Tabmda: Tabular manifold data augmentation for any classifier using transformers with in-context subsetting. *CoRR*, abs/2406.01805, 2024.

McElfresh, D. C., Khandagale, S., Valverde, J., C., V. P., Ramakrishnan, G., Goldblum, M., and White, C. When do neural nets outperform boosted trees on tabular data? In *NeurIPS*, pp. 76336–76369, 2023.

Nagler, T. Statistical foundations of prior-data fitted networks. In *ICML*, pp. 25660–25676, 2023.

Nederstigt, L. J., Aanen, S. S., Vandic, D., and Frasincar, F. Floppies: a framework for large-scale ontology population of product information from tabular data in e-commerce stores. *Decision Support Systems*, 59:296–311, 2014.

Popov, S., Morozov, S., and Babenko, A. Neural oblivious decision ensembles for deep learning on tabular data. In *ICLR*, 2020.

Prokhorenkova, L. O., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. Catboost: unbiased boosting with categorical features. In *NeurIPS*, pp. 6639–6649, 2018.

S., M., P., R., and P., C. Bank Marketing. UCI Machine Learning Repository, 2014.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, S. K. S., Lopes, R. G., Ayan, B. K., Salimans, T., Ho, J., Fleet, D. J., and Norouzi, M. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, pp. 36479–36494, 2022.

Schwartz, L. M., Woloshin, S., and Welch, H. G. The drug facts box: providing consumers with simple tabular data on drug benefit and harm. *Medical Decision Making*, 27 (5):655–662, 2007.

Song, W., Shi, C., Xiao, Z., Duan, Z., Xu, Y., Zhang, M., and Tang, J. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, pp. 1161–1170, 2019.

Subasi, A. Medical decision support system for diagnosis of neuromuscular disorders using dwt and fuzzy support vector machines. *Computers in Biology and Medicine*, 42(8):806–815, 2012.

Thomas, V., Ma, J., Hosseinzadeh, R., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. Retrieval & fine-tuning for in-context tabular models. *CoRR*, abs/2406.05207, 2024.

Valipour, M., Rezagholizadeh, M., Kobyzev, I., and Ghodsi, A. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *EACL*, pp. 3266–3279, 2023.

van Breugel, B. and van der Schaar, M. Position: Why tabular foundation models should be a research priority. In *ICML*, 2024.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Wang, R., Fu, B., Fu, G., and Wang, M. Deep & cross network for ad click predictions. In *ADKDD*, 2017.

Wang, R., Shivanna, R., Cheng, D. Z., Jain, S., Lin, D., Hong, L., and Chi, E. H. DCN V2: improved deep & cross network and practical lessons for web-scale learning to rank systems. In *WWW*, pp. 1785–1797, 2021.

Wang, Z. and Sun, J. Transtab: Learning transferable tabular transformers across tables. In *NeurIPS*, pp. 2902–2915, 2022.

Wen, Y., Tran, D., and Ba, J. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *ICLR*, 2020.

Wu, J., Chen, S., Zhao, Q., Sergazinov, R., Li, C., Liu, S., Zhao, C., Xie, T., Guo, H., Ji, C., Cociorva, D., and Brunzell, H. Switchtab: Switched autoencoders are effective tabular learners. In *AAAI*, pp. 15924–15933, 2024.

Xu, D., Cirit, O., Asadi, R., Sun, Y., and Wang, W. Mixture of in-context prompters for tabular pfns. In *ICLR*, 2025.

Yan, J., Zheng, B., Xu, H., Zhu, Y., Chen, D. Z., Sun, J., Wu, J., and Chen, J. Making pre-trained language models great on tabular prediction. In *ICLR*, 2024.

Yang, Y., Wang, Y., Liu, G., Wu, L., and Liu, Q. Unitabe: A universal pretraining protocol for tabular foundation model in data science. In *ICLR*, 2024.

Ye, H., Zhou, Q., and Zhan, D. Training-free generalization on heterogeneous tabular data via meta-representation. *CoRR*, abs/2311.00055, 2023.

Ye, H., Fan, W., Song, X., Zheng, S., Zhao, H., dan Guo, D., and Chang, Y. Ptarl: Prototype-based tabular representation learning via space calibration. In *ICLR*, 2024a.

Ye, H., Liu, S., Cai, H., Zhou, Q., and Zhan, D. A closer look at deep learning on tabular data. *CoRR*, abs/2407.00956, 2024b.

Ye, H.-J., Yin, H.-H., and Zhan, D.-C. Modern neighborhood components analysis: A deep tabular baseline two decades later. In *ICLR*, 2025.

Zhang, J. O., Sax, A., Zamir, A., Guibas, L. J., and Malik, J. Side-tuning: A baseline for network adaptation via additive side networks. In *ECCV (3)*, volume 12348 of *Lecture Notes in Computer Science*, pp. 698–714. Springer, 2020.

Zhou, Q.-L., Ye, H.-J., Wang, L., and Zhan, D.-C. Unlocking the transferability of tokens in deep models for tabular data. *CoRR*, abs/2310.15149, 2023.

Zhu, B., Shi, X., Erickson, N., Li, M., Karypis, G., and Shoaran, M. Xtab: Cross-table pretraining for tabular transformers. In *ICML*, pp. 43181–43204, 2023.

The Appendix consists of four sections:

1. Appendix A: Related work.
2. Appendix B: Preliminary.
3. Appendix C: Datasets and implementation details.
4. Appendix D: Hardware and limitations.
5. Appendix E: Additional experiments

A. Related Work

A.1. Tabular Data Learning

Tabular data is one of the most widely used dataset types in machine learning. Gradient-boosted decision trees (GBDTs) (Chen & Guestrin, 2016; Prokhorenkova et al., 2018; Ke et al., 2017), remain a strong baseline for tabular tasks due to their efficiency and high performance. As an ensemble-based method, GBDTs construct multiple decision trees to iteratively minimize the residual loss. With the advancement of deep learning, an increasing number of studies have explored using deep learning methods for tabular data prediction. These approaches include MLP variants (Klambauer et al., 2017; Gorishniy et al., 2021; 2022; Holzmüller et al., 2024), neural networks specifically designed for tabular structures (Wang et al., 2017; 2021; Chen et al., 2023a), attention-based models (Song et al., 2019; Huang et al., 2020; Gorishniy et al., 2021; Chen et al., 2024), methods incorporating regularization (Ye et al., 2024a; Alan et al., 2023; Wu et al., 2024), tree-mimic methods (Arik & Pfister, 2021; Popov et al., 2020; Badirli et al., 2020), and context-based methods (Gorishniy et al., 2024; Ye et al., 2025). Despite these advancements, recent benchmarks (Grinsztajn et al., 2022; McElfresh et al., 2023; Ye et al., 2024b) have consistently demonstrated that gradient-boosted decision trees (GBDTs) outperform deep learning in tabular prediction tasks. The superior performance of GBDTs can be attributed to two main factors: (1) their ability to handle heterogeneous tabular datasets, which often describe high-frequency target functions (Basri et al., 2020; Grinsztajn et al., 2022), and (2) the ensemble nature of GBDTs. Prior attempts to introduce ensemble-like mechanisms into tabular deep learning (Badirli et al., 2020; Popov et al., 2020; Chen et al., 2023c), have not been widely successful (Grinsztajn et al., 2022; Ye et al., 2024b). However, recent works like TabM (Gorishniy et al., 2025) integrate Batch Ensemble (Wen et al., 2020) techniques into the tabular domain, showing how efficient ensembling can be achieved with deep learning models.

A.2. Tabular Foundation Models

While tabular foundation models (TFMs) are not as developed as foundation models in other domains, such as computer vision (Saharia et al., 2022) and natural language processing (Brown et al., 2020), recent efforts have introduced various architectures to bridge this gap (van Breugel & van der Schaar, 2024). Some approaches aim to explore model components that can be shared across datasets (Liu et al., 2022; Zhu et al., 2023), while others focus on utilizing the semantic information inherent in tabular datasets (Wang & Sun, 2022; Yan et al., 2024; Gardner et al., 2024; Kim et al., 2024; Yang et al., 2024; Kasneci & Kasneci, 2024). As a Transformer-based model, TabPFN (Hollmann et al., 2023; 2025) stands out for its exceptional performance and efficiency on small datasets. By leveraging the in-context learning capability of transformer (Brown et al., 2020), it can make predictions for unseen instances without parameter updates. However, TabPFN faces limitations related to dataset size and feature dimensionality. To address these challenges, some studies have explored improvements in its architecture and training paradigm (den Breejen et al., 2024; Ma et al., 2024b), while others focus on adaptation techniques that expand TabPFN's applicability to a broader range of downstream tabular datasets (Feuer et al., 2024; Thomas et al., 2024). Our work falls into the latter category, offering a straightforward, efficient, and effective approach to align TabPFN with downstream datasets.

A.3. Parameter-efficient Fine-Tuning

Some approaches for adapting Tabular Foundation Models (TFMs) to downstream tasks simply fine-tune the entire model (Thomas et al., 2024; den Breejen et al., 2024), which leads to performance improvements but incurs high computational and storage costs. Parameter-efficient fine-tuning (PEFT) offers a solution to the challenge by enabling adaptation with a minimal number of trainable parameters (Lialin et al., 2023). Based on their operational mechanisms, PEFT methods can be broadly categorized into four paradigms (Han et al., 2024):

- **Additive Methods**. These methods incorporate additional lightweight modules into the model architecture, such as adapters (Houlsby et al., 2019; Zhang et al., 2020; Xu et al., 2025) or soft prompts (Li & Liang, 2021; Feuer et al., 2024).
- **Selective Methods**. Instead of introducing new parameters, selective methods strategically identify and update the most relevant parameters while freezing the rest (Fu et al., 2023; He et al., 2023).
- **Reparameterized Methods**. These methods employ low-rank decomposition or equivalent transformations to reduce the parameter space during fine-tuning (Aghajanyan et al., 2021; Hu et al., 2022; Valipour et al., 2023).
- **Hybrid Methods**. By combining the strengths of multiple PEFT strategies, hybrid methods create a unified framework that enhances fine-tuning performance while maintaining efficiency (Mao et al., 2022; Chen et al., 2023b).

Our proposed method adopts parameter-efficient tuning focused on input feature adaptation. This design is motivated by the unique characteristics of tabular data: Tabular datasets are inherently heterogeneous, with varying structures and feature distributions across different datasets (Zhou et al., 2023; Ye et al., 2023; Borisov et al., 2024). Adapting TabPFN through input feature alignment effectively mitigates the constraints on input dimensionality, enhancing its applicability across a wider range of tasks. By employing parameter-efficient fine-tuning, we align TabPFN with downstream datasets, addressing its existing limitations (Hollmann et al., 2023).

B. Preliminary

In this section, we provide a brief overview of TabPFN and analyze its properties, while also revisiting existing variants.

B.1. TabPFN

We consider a tabular dataset consisting of N examples and d features. Each instance $\boldsymbol{x}_i \in \mathbb{R}^d$ is represented by d feature values, where $\boldsymbol{x}_{i,j}$ denotes the j-th feature of instance $\boldsymbol{x}_i$. These features can be numerical ($\boldsymbol{x}_{i,j}^{\text{num}} \in \mathbb{R}$) or categorical ($\boldsymbol{x}_{i,j}^{\text{cat}}$), with categorical values often encoded as integers. Each instance is associated with a label $\boldsymbol{y}_i$, where $\boldsymbol{y}_i \in [C] = \{1, \ldots, C\}$ for classification task, and $\boldsymbol{y}_i \in \mathbb{R}$ for regression task. Given a training dataset, $D_{\text{train}} = \{(\boldsymbol{x}_{\text{train}}^{(i)}, \boldsymbol{y}_{\text{train}}^{(i)})\}_{i=1}^{N_{\text{train}}}$, and test samples, $\boldsymbol{X}_{\text{test}} = [\boldsymbol{x}_{\text{test}}^{(i)}]_{i=1}^{N_{\text{test}}}$, the goal is to predict the corresponding labels, $\boldsymbol{Y}_{\text{test}} = [\boldsymbol{y}_{\text{test}}^{(i)}]_{i=1}^{N_{\text{test}}}$, as accurately as possible.

TabPFN follows a two-stage process: a pre-training stage, where the model is trained on synthetic datasets by minimizing the discrepancy between the predicted label of the test instance and its true label, and an inference stage, where it directly predicts the labels of test samples given a set of labeled training examples.

Following pre-training, TabPFN takes the entire training dataset $D_{\text{train}} = (\boldsymbol{X}_{\text{train}}, \boldsymbol{y}_{\text{train}})$ and the features of query points $\boldsymbol{X}_{\text{q}}$ as context to make predictions. This is done using **PFN-Style Batching** (Hollmann et al., 2023), where N_{test} test samples are batched into a single "prompt," as illustrated in Figure 4. Since TabPFN is trained with a fixed input dimensionality, typically set to a predefined value $d_{\max}$ (e.g., 100), datasets with fewer features ($d < d_{\max}$) need to be extended via zero-padding before being processed. Formally, given an input instance $\boldsymbol{x}_i \in \mathbb{R}^d$, the padded input $\tilde{\boldsymbol{x}}_i \in \mathbb{R}^{d_{\max}}$ is obtained as:

$$\tilde{\boldsymbol{x}}_i = [\boldsymbol{x}_i, \boldsymbol{0}] \in \mathbb{R}^{d_{\max}}, \tag{6}$$

where $\boldsymbol{0} \in \mathbb{R}^{d_{\max}-d}$ represents the zero-padding applied to match the required input dimensionality. However, if a dataset exceeds this limit ($d > d_{\max}$), TabPFN cannot directly process such datasets, as its architecture is not designed to accommodate higher-dimensional inputs.

TabPFN outputs a probability distribution over possible labels $\boldsymbol{y}_{\text{q}} \in \{1, \ldots, C\}$. Specifically, let q_θ denote the logits produced by TabPFN, where θ represents the parameters of the pre-trained model. The posterior predictive distribution is given by:

$$p_\theta(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{X}_{\text{q}}, D_{\text{train}}) = \frac{\exp(q_\theta(\tilde{\boldsymbol{X}}_{\text{q}}, \tilde{D}_{\text{train}})_{[\boldsymbol{y}_{\text{q}}]})}{\sum_{c=1}^{C} \exp(q_\theta(\tilde{\boldsymbol{X}}_{\text{q}}, \tilde{D}_{\text{train}})_{[c]})}, \tag{7}$$

where $\tilde{D}_{\text{train}}$ denotes the zero-padded training dataset, acting as the support set for the context composition. Since TabPFN performs inference directly, the model typically applies multiple rounds of feature shuffling on the original features, followed by prediction. The final prediction is obtained by averaging the results from these multiple inferences. For more details about how the model processes inputs, please refer to Appendix C.5.

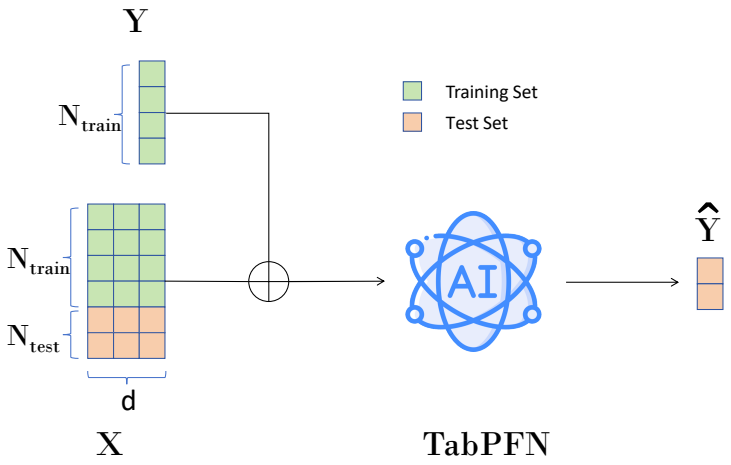
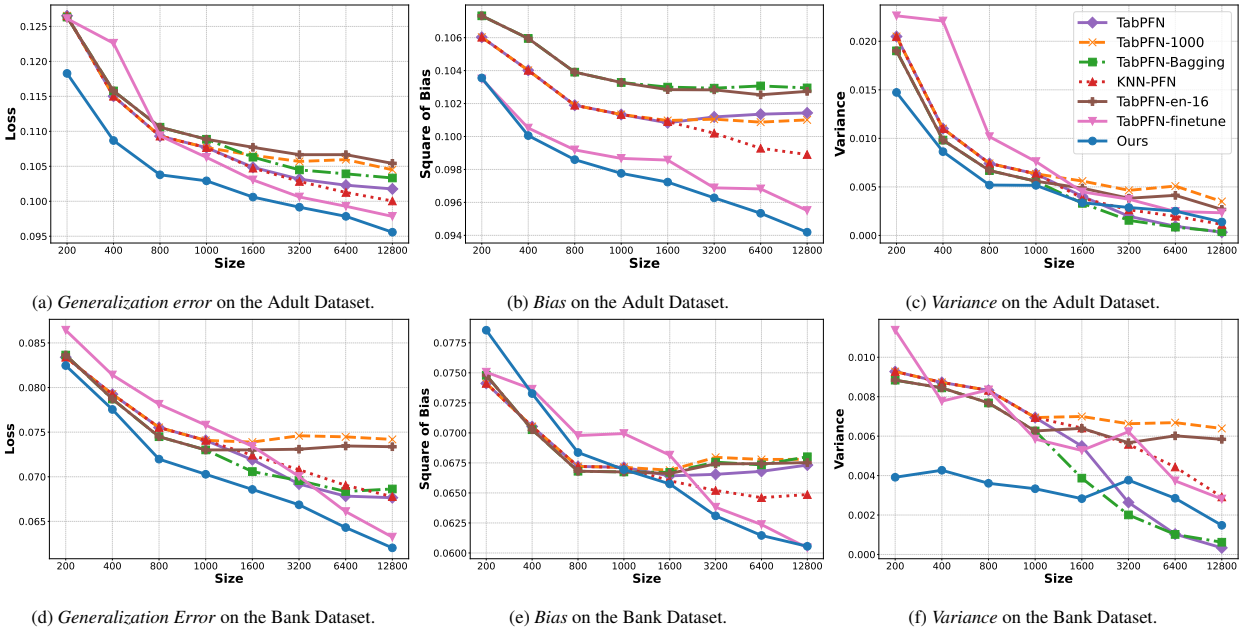


Figure 4: The prediction process of TabPFN. The training set and test set are concatenated, with the training labels added to the input sequence. TabPFN then performs a single forward pass to generate predictions for all test samples.

B.2. TabPFN Variants

Existing research to improve the performance of TabPFN has primarily focused on two different strategies, stemming from its dependency on q_θ and D_{train} in Equation 7. One approach optimizes context selection, refining D_{train} to provide more informative support examples. The other concentrates on fine-tuning TabPFN, improving q_θ to better adapt to downstream tasks. These two strategies offer complementary solutions for enhancing TabPFN's overall performance. **Context Selection for Scaling TabPFN.** The transformer architecture in TabPFN inherently leads to quadratic growth in memory usage as the context length increases. As a result, the size of the support set used as context in each prediction is limited. Meanwhile, a well-chosen D_{train} provides more relevant support examples, improving generalization. A simple approach is random subsampling (McElfresh et al., 2023), but this can degrade performance, especially on large datasets (Ma et al., 2024c). More structured methods include sketching techniques, such as CoreSet, K-Means, and Data Distillation (Ma et al., 2024c), which aim to compress large datasets into representative subsets while preserving essential information (Feuer et al., 2023). TuneTables (Feuer et al., 2024), on the other hand, attempts to encode the dataset into a compact learned representation, reducing memory overhead. Another class of methods selects sample-specific contexts instead of a fixed support set. For instance, MixturePFN (Xu et al., 2025) partitions the training set into multiple subsets and assigns the most relevant one to each test sample. Similarity, LocalPFN (Thomas et al., 2024) and TabDPT (Ma et al., 2024b), instead use nearest neighbors to dynamically construct the support set D_{train} for each query (KNN-based selection). While these approaches improve TabPFN's performance by refining D_{train}, they disrupt PFN-style batching, significantly reducing inference efficiency.

Fine-tuning Strategies for Enhancing TabPFN Performance. The second approach to improving TabPFN's performance focuses on fine-tuning the pre-trained model to better adapt to downstream datasets. As shown in Equation 7, the predictive distribution $p_\theta(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{X}_{\text{q}}, D_{\text{train}})$ depends on both the training set D_{train} and the model parameters θ. While optimizing D_{train} improves the quality of support examples, fine-tuning enhances q_θ, enabling better alignment with downstream tasks. A straightforward approach is to fine-tune all model parameters, as seen in TabForestPFN (den Breejen et al., 2024) and LocalPFN (Thomas et al., 2024), which improves performance by adapting TabPFN's learned prior to specific datasets. However, full fine-tuning incurs substantial computational costs due to the large number of model parameters. To mitigate this, TuneTables (Feuer et al., 2024) offers a more efficient alternative by either fine-tuning the entire model or applying prompt-tuning (Lester et al., 2021), which adjusts only a small set of parameters, reducing resource consumption. Another efficient approach is adapter-based fine-tuning, as employed in MixturePFN (Xu et al., 2025), which fine-tunes additional adapter layers (Houlsby et al., 2019) rather than modifying the entire model. This strategy provides a balance between computational efficiency and performance improvement, allowing the model to adapt while preserving the efficiency of the original pre-trained TabPFN. A detailed related work is presented in Appendix A.



(a) *Generalization error* on the Adult Dataset. (b) *Bias* on the Adult Dataset. (c) *Variance* on the Adult Dataset.

(d) *Generalization Error* on the Bank Dataset. (e) *Bias* on the Bank Dataset. (f) *Variance* on the Bank Dataset.

Figure 5: Generalization Error, Bias, and Variance for different methods on the Adult and Bank datasets. The methods shown include TabPFN-1000 (subsample size = 1000), TabPFN, TabPFN-en-16, TabPFN-Bagging, KNN-PFN, TabPFN-finetune, and Ours (BETA). The legend is located in the top-right plot for clarity.

B.3. Generalization Analysis of TabPFN Variants

While various TabPFN variants have been proposed to enhance performance, their underlying mechanisms remain insufficiently understood. In this section, we leverage the **bias-variance decomposition framework**, as described in Equation 12, which is introduced by Nagler (2023), to analyze the generalization error of TabPFN and its variants on two real-world datasets: *Adult* (Barry & Ronny, 1996) and *Bank* (S. et al., 2014). To assess the impact of different strategies, and based on our previous analysis, we evaluate the original TabPFN (without context length restrictions) alongside several representative variants. These include TabPFN-1000 (random subsampling of 1000 samples), TabPFN-en-16 (randomly sample once and perform feature shuffling 16 times to form an ensemble of 16 predictions), TabPFN-KNN (KNN-based context selection), TabPFN-finetune (full model fine-tuning), and TabPFN-Bagging (bootstrapped sampling with 16 varying contexts). TabPFN-Bagging, unlike ensemble methods that require training multiple independent models, employs bootstrapped sampling to construct diverse support sets within a single inference process. A detailed description of its implementation is provided in Appendix C.6.

To further contextualize these findings, we compare all evaluated methods against our proposed approach, BETA. The experimental results, presented in Figure 5, reveal several key trends in the bias-variance tradeoff across different TabPFN variants. As shown in Figure 5 (b,c,e,f), increasing context length reduces variance while bias plateaus, consistent with prior findings (Nagler, 2023). Similarly, KNN-based context selection reduces bias but increases variance compared to using the full dataset, as it relies on localized subsets. Since these trends align with previous work, we focus on additional findings.

1) Fine-Tuning and its Impact: Figure 5 (b,e) shows that fine-tuning (TabPFN-finetune) effectively reduces bias by aligning the model's prior with the characteristics of the dataset. However, it also increases variance, especially when the training set is small, where overfitting amplifies prediction variability, as shown in Figure 5 (c,f). These results suggest that fine-tuning requires careful regularization to balance bias and variance.

2) Ensemble Strategy and Bias-Variance Tradeoff: Figure 5 (b,c) indicates that ensemble-based methods (TabPFN-en-16) reduce variance but may increase bias due to unsupervised feature transformations. This observation highlights that ensemble methods, while helpful in reducing variance, may require additional bias-reducing strategies to optimize overall model performance.

3) Effectiveness of Bagging: As shown in Figure 5 (c,f), Bagging significantly reduces variance while maintaining stable

Table 1: Comparison of TabPFN and related methods in terms of their impact on bias and variance, as well as their effectiveness in handling large datasets, high-dimensional features, adaptability to multiclass classification, and inference efficiency. The compared methods include TabPFN (Hollmann et al., 2023), TuneTables (Feuer et al., 2024), TabForestPFN (den Breejen et al., 2024), LocalPFN (Thomas et al., 2024), and MixturePFN (Xu et al., 2025). Our proposed method (BETA) is distinguished for its ability to balance bias and variance while maintaining efficient scaling, adaptability to multiclass classification, and lightweight fine-tuning. Different row colors indicate the *strategies used for improving performance* ( ), *scalability* ( ), and *efficiency* ( ).

	BETA (Ours)	TabPFN	TuneTables	TabForestPFN	LocaLPFN	MixturePFN
Reduces Bias	✓	✗	✓	✓	✓	✓
Reduces Variance	✓	✗	✗	✗	✗	✗
Scales to Large Datasets	✓	✗	✓	✗	✓	✓
Handles High-Dimensional Data	✓	✗	✗	✗	✗	✗
Adapts to More Than 10 Classes	✓	✗	✓	✗	✗	✗
No Additional Inference Cost	✓	✓	✓	✓	✗	✗
Fine-Tuning	lightweight encoder	✗	prompt & backbone	backbone	backbone	adapter

bias. By introducing diversity through bootstrapped sampling, it achieves variance reduction comparable to ensemble methods but at a lower computational cost. These results highlight Bagging as a simple yet effective approach for improving TabPFN's performance.

The results above, based on additional findings not covered in (Nagler, 2023), indicate that TabPFN variants typically impact either bias or variance, but rarely both simultaneously. For instance, KNN-based context selection reduces bias but increases variance, while the original ensemble strategy in Hollmann et al. (2023) (TabPFN-en-16) lowers variance but may increase bias. These observations highlight the need for a method that jointly optimizes both aspects, motivating the development of BETA.

Existing approaches to improving TabPFN focus on context selection or fine-tuning, but often introduce trade-offs in computational efficiency, scalability, and adaptability. A comparison in Table 1 highlights how different strategies affect bias, variance, and their ability to handle large datasets and high-dimensional features efficiently.

C. Datasets and implementation details

In this section, we outline the descriptions of the datasets used in the experiments and the preprocessing steps applied to them before training. Additionally, we will describe the implementation details of BETA and the comparison methods.

C.1. Experiment Setup

Datasets. In our experiments, we evaluate BETA on one of the largest publicly available tabular benchmark TALENT (Ye et al., 2024b), which includes 120 binary classification datasets and 80 multi-class classification datasets. These datasets are collected from various sources such as UCI, OpenML, Kaggle, and others. To ensure fairness, we remove two datasets, "PizzaCutter3" and "PieChart3," as they overlap with TabPFN's validation set. In addition, to validate the ability of BETA to handle high-dimensional feature datasets, we also include 20 high-dimensional datasets sourced from the scikit-feature repository.

Evaluation. For the TALENT datasets, we follow the evaluation protocol from (Gorishniy et al., 2021). Each dataset is randomly split into training, validation, and test sets with proportions of 64%, 16%, and 20%, respectively. For each dataset, we train each model using 15 different random seeds and calculate the average performance on the test set. We report accuracy as the evaluation metric, where higher accuracy indicates better performance. For a more detailed comparison with other TabPFN variants, we separately evaluate datasets with fewer than 10 classes, those with more than 10 classes, and high-dimensional datasets. Further details on the experimental setup, are provided in Appendix C.

Methods Compared. We compare BETA against five categories of methods to evaluate its effectiveness comprehensively: (1) **Classical Machine Learning Algorithms:** This category includes widely used classical approaches such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and tree-based methods like Random Forest (RForest) (Breiman, 2001), XGBoost (XGB) (Chen & Guestrin, 2016), CatBoost (CatB) (Prokhorenkova et al., 2018), and LightGBM

(LightG) (Ke et al., 2017). (2) **Tabular Deep Learning Models:** We consider state-of-the-art deep learning models for tabular data, including MLP, ResNet, FT-Transformer (FT-T) (Gorishniy et al., 2021), MLP-PLR (Gorishniy et al., 2022), DCNv2 (Wang et al., 2021), AutoInt (Song et al., 2019), SNN (Klambauer et al., 2017), ExcelFormer (ExcelF) (Chen et al., 2024), DANets (Chen et al., 2022), TabTransformer (TabT) (Huang et al., 2020), and TabNet (Arik & Pfister, 2021). (3) **Neighborhood-Based Methods:** To explore neighbor-based strategies, we evaluate TabR (Gorishniy et al., 2024) and ModernNCA (MNCA) (Ye et al., 2025). (4) **Ensemble-Based Methods:** We include methods that leverage ensemble strategies, such as TabM (Gorishniy et al., 2025), NODE, and GrowNet. (5) **TabPFN and Its Variants:** Lastly, we compare with TabPFN and its recent variants, including knnPFN (Thomas et al., 2024), LocalPFN (Thomas et al., 2024), MixturePFN (Xu et al., 2025), and TuneTables (Feuer et al., 2024).

Implementation Details. All datasets are pre-processed following the methodology outlined in Gorishniy et al. (2021). For deep learning-based methods, we set the batch size to 1024. Hyper-parameters for the compared methods are tuned using Optuna (Akiba et al., 2019), performing over 100 trials. The search ranges for hyper-parameters are determined based on Gorishniy et al. (2021); Liu et al. (2024) and the official implementations of each method. Once the optimal hyper-parameters are identified, they are fixed for the final evaluation using 15 random seeds. To ensure a fair comparison, **all TabPFN variants, including BETA**, are evaluated using their **default hyper-parameters without additional tuning**. More details can be found in Appendix C.

C.2. Datasets information

For both the main experiments and the classification tasks with more than 10 categories, we use the current largest tabular benchmark (Ye et al., 2024b), which includes **120 binary classification datasets and 80 multiclass classification** datasets, spanning diverse domains such as healthcare, biology, finance, education, and physics. For more detailed information about these datasets, please refer to Ye et al. (2024b).

For each dataset, we randomly select 20% of the instances to form the test set. The remaining 80% is further split, with 20% reserved as a validation set. This validation set is used for hyperparameter tuning (for the comparison methods) and early stopping. The hyperparameters that yield the best performance on the validation set are selected for final evaluation on the test set.

Remark: BETA are based on pre-trained models TabPFN (Hollmann et al., 2023). To ensure a fair comparison, we do not perform hyperparameter search, and instead use the default hyperparameters for all methods.

For high-dimensional datasets, we obtained the data from scikit-feature repository, and excluded those with more than 10 classes. This resulted in a final set of 20 datasets, as shown in Table 2. The dataset splitting follows the same approach as in the main experiments, but due to the smaller number of instances in these datasets, we use the default hyperparameters for the experiments and do not perform hyperparameter search. The validation set is only used for early stopping. For each dataset, we perform five random splits and run all methods three times on each split using three different seeds (0, 1, 2), resulting in a total of 15 runs per dataset. The final results are reported as the mean of these 15 runs.

Table 2: Dataset Information for High-Dimensional Data Experiments: A collection of 20 datasets with varying numbers of instances, features, and classes used in our high-dimensional experiments.

Dataset	#Instances	#Features	#Classes	Dataset	#Instances	#Features	#Classes
BASEHOCK	1993	4862	2	lung_discrete	73	325	7
PCMAC	1943	3289	2	warpPIE10P	210	2420	10
RELATHE	1427	4322	2	orlraws10P	100	10304	10
ALLAML	72	7129	2	Prostate_GE	102	5966	2
CLL_SUB_111	111	11340	3	SMK_CAN_187	187	19993	2
colon	62	2000	2	warpAR10P	130	2400	10
GLI_85	85	22283	2	arcene	200	10000	2
GLIOMA	50	4434	4	gisette	7000	5000	2
leukemia	72	7070	2	madelon	2600	500	2
lung	203	3312	5	TOX_171	171	5748	4

C.3. Dataset Pre-processing

Unless otherwise specified, we adopt the data preprocessing pipeline outlined by Gorishniy et al. (2021). For numerical features, we standardize by subtracting the mean and scaling the values to unit variance. Categorical features are converted into a model-compatible format using one-hot encoding.

C.4. Implementations

BETA. For all experiments except for the ablation study, we set the context size to 1000 and fixed the number of bootstrap sampling iterations to 16. For the feature transformation encoder, the main structure is a two-layer MLP, with both the hidden and output dimensions set to 100, using the ReLU activation function. In addition, to enhance the expressive power of the encoder, we also apply periodic activation (Gorishniy et al., 2022). The initialization of Batch Ensemble is inspired by Gorishniy et al. (2025). During fine-tuning, we use the pre-trained model checkpoint[1], and fine-tuning is performed using the AdamW (Loshchilov & Hutter, 2019) optimizer with a learning rate of 0.003, weight decay of 1e-5, and a batch size of 1024. In the fine-tuning phase, only the encoder parameters are updated, while the pre-trained TabPFN parameters are frozen. For experiments on high-dimensional datasets, due to the large number of features and the limitations of device memory, we do not use periodic activation. In the classification experiments with more than 10 classes, we use a code length of 32, which means there are 32 paths in the encoder. Additionally, we observed that for certain datasets, simply adjusting the output dimension of the pre-trained model and fine-tuning it can yield strong results (Feuer et al., 2024). Based on validation set performance, we choose between ECOC and fine-tuning the output layer MLP approaches.

TabPFN and its variants. For TabPFN, we use the implementation from TALENT (Liu et al., 2024); For TuneTables, we adopt the official code[2] and use the prompt tuning mode without performing full model fine-tuning; For LocalPFN, since the code has not been released, we replicate the hyperparameters provided in the original paper. Specifically, we set the number of neighbors $k = 1000$. During our experiments, we found that fine-tuning LocalPFN with a smaller learning rate yields better performance, so we set the learning rate to 1e-5; For MixturePFN, we reproduce the model using the hyperparameters recommended in the original paper. The number of training samples in the prompt is set to $B = 3000$, and the number of experts is set to the training set size divided by B, rounded up to the nearest integer.

Other Methods from TALENT. For the other methods in TALENT, we either use the results provided (Ye et al., 2024b) or perform a hyperparameter search using the hyperparameters provided by TALENT. In the case of hyperparameter tuning, we conduct 100 trials of hyperparameter search for each method[3]. For each search, we run the experiments using 15 different seeds and report the average results across these runs.

C.5. Details of TabPFN

In the original TabPFN approach, the model processes the input data in two phases: pre-training and inference. The pre-training phase involves training the model on synthetic data, while the inference phase is used to make predictions on new, unseen data.

For the input, we consider a training dataset $D_{\text{train}} = (X_{\text{train}}, y_{\text{train}})$, where $X_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times d}$ represents the feature matrix, and $y_{\text{train}} \in \mathbb{R}^{N_{\text{train}}}$ represents the corresponding labels. Here, N_{train} denotes the number of training samples, and d is the number of features. For datasets with fewer features than the fixed input dimensionality $d_{\max}$, zero-padding is applied to extend the feature vectors to the required size $d_{\max}$. Specifically, each feature vector $X_i \in \mathbb{R}^d$ is padded with zeros to form $\tilde{X}_i \in \mathbb{R}^{d_{\max}}$, as follows:

$$\tilde{X}_i = [X_i, \mathbf{0}] \in \mathbb{R}^{d_{\max}}.$$

The zero-padded feature matrix $\tilde{X}_{\text{train}}$ is then used during inference for making predictions.

For inference, we consider a support set S containing the training examples and a query set Q containing the new, unseen instances for which we wish to make predictions. The query set Q consists of N_{test} test samples, and the goal is to predict the corresponding labels y_{test}.

[1] https://github.com/automl/TabPFN/blob/tabpfn_v1/tabpfn/models_diff/prior_diff_real_checkpoint_n_0_epoch_42.cpkt
[2] https://github.com/penfever/TuneTables
[3] https://github.com/qile2000/LAMDA-TALENT/tree/main/LAMDA_TALENT/configs

TabPFN uses **PFN-style batching** for inference, where both the support set S and query set Q are batched together into a single prompt, which is then fed into the pre-trained model for prediction. This batching process allows the model to utilize the support set as context for making predictions on the query set.

During the processing, the features of both the support set and the query set are transformed into token representations. The support tokens L_{support} are obtained as:

$$L_{\text{support}} = \tilde{X}_{\text{support}} W_x + y_{\text{support}} w_y^T,$$

where $\tilde{X}_{\text{support}} \in \mathbb{R}^{|S| \times d_{\max}}$ is the zero-padded feature matrix of the support set, $W_x \in \mathbb{R}^{d_{\max} \times d_{\text{token}}}$ is the embedding matrix for the features, and $w_y \in \mathbb{R}^{d_{\text{token}}}$ is the embedding matrix for the labels. The query tokens L_{query} are similarly transformed as:

$$L_{\text{query}} = X_{\text{query}} W_x,$$

where $X_{\text{query}} \in \mathbb{R}^{|Q| \times d_{\max}}$ represents the query set of test samples.

These token representations are then passed through a standard transformer model, which includes a special attention mechanism. The attention mask ensures that the support tokens can only attend to other support tokens, and query tokens can attend to both the support tokens and themselves. Query tokens do not attend to other query tokens, preventing any information leakage between query samples during inference. Finally, the output tokens corresponding to the test instances are extracted and mapped to 10-class logits for classification.

C.6. Implementation of TabPFN-Bagging

TabPFN-Bagging is a variance reduction technique that leverages **bootstrapped sampling** to create diverse support sets for model inference. Unlike standard ensemble approaches, which require training multiple independent models, TabPFN-Bagging generates multiple resampled versions of D_{train} within a single inference process, reducing variance without additional computational overhead.

Bootstrapped Sampling in TabPFN. Given a training dataset $D_{\text{train}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N_{\text{train}}}$, we generate K bootstrapped support sets $D_{\text{train}}^{(k)}$, where each set is constructed by randomly sampling N_{sub} instances with replacement:

$$D_{\text{train}}^{(k)} = \left\{ \left(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)} \right) \right\}_{i=1}^{N_{\text{sub}}}, \quad k = 1, \ldots, K. \tag{8}$$

Each $D_{\text{train}}^{(k)}$ serves as an alternative context set for model inference, introducing diversity into the prediction process.

Prediction Aggregation. For each test instance $\boldsymbol{x}_{\text{q}}$, we compute the posterior predictive distribution using the bootstrapped support sets. Given the pre-trained TabPFN model parameterized by θ, the predictive probability for label $\boldsymbol{y}_{\text{q}}$ is computed as:

$$p_\theta^{(k)}(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{x}_{\text{q}}, D_{\text{train}}^{(k)}) = \frac{\exp(q_\theta(\boldsymbol{x}_{\text{q}}, D_{\text{train}}^{(k)})_{[\boldsymbol{y}_{\text{q}}]})}{\sum_{c=1}^{C} \exp(q_\theta(\boldsymbol{x}_{\text{q}}, D_{\text{train}}^{(k)})_{[c]})}. \tag{9}$$

where q_θ denotes the logits produced by TabPFN for the given support set $D_{\text{train}}^{(k)}$.

To obtain the final prediction, we aggregate the results across all bootstrapped support sets using either uniform averaging:

$$p_\theta(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{x}_{\text{q}}, D_{\text{train}}) = \frac{1}{K} \sum_{k=1}^{K} p_\theta^{(k)}(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{x}_{\text{q}}, D_{\text{train}}^{(k)}), \tag{10}$$

or a weighted aggregation method, where weights w_k are assigned based on the confidence of each individual model:

$$p_\theta(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{x}_{\text{q}}, D_{\text{train}}) = \sum_{k=1}^{K} w_k p_\theta^{(k)}(\boldsymbol{y}_{\text{q}} \mid \boldsymbol{x}_{\text{q}}, D_{\text{train}}^{(k)}), \quad \sum_{k=1}^{K} w_k = 1. \tag{11}$$

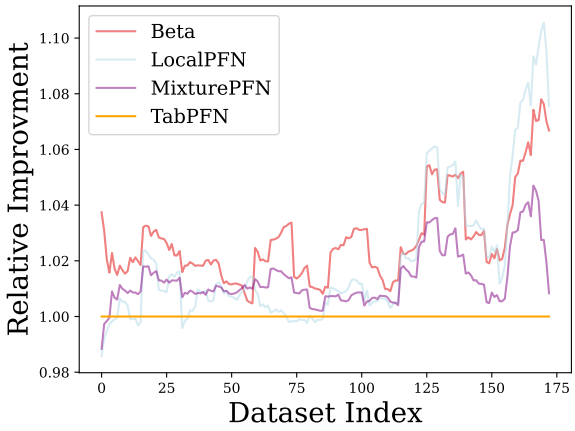


Figure 6: The relative improvement of LocalPFN (Thomas et al., 2024), MixturePFN (Xu et al., 2025), and BETA (Ours) over TabPFN across 173 tabular datasets sorted by dataset size (number of rows). The curves represent smoothed results to better illustrate trends in performance improvement as dataset size increases.

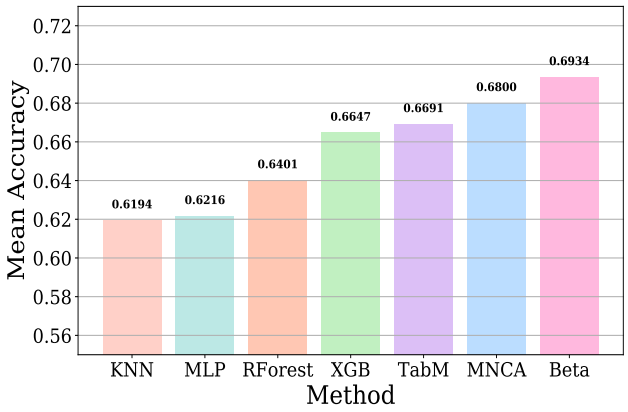


Figure 7: Comparison of the mean accuracy across 12 classification datasets with more than 10 classes. **Since the original TabPFN cannot process classification tasks with more than ten categories**, we compare BETA against alternative methods, including TabM, KNN, MLP, XGBoost (XGB), RandomForest (RForest), and ModernNCA (MNCA).

Computational Efficiency. Since bootstrapped sampling only modifies the selection of D_{train} without altering the model architecture, TabPFN-Bagging incurs no additional computational cost beyond standard inference with ensemble. Unlike ensemble-based methods that require multiple independent model evaluations, all computations occur within a single forward pass of the pre-trained TabPFN. This makes TabPFN-Bagging an efficient and scalable variance reduction strategy.

D. Hardware and limitations

D.1. Hardware

Most of the experiments were performed with four NVIDIA 4090 GPUs and four NVIDIA A6000 GPUs.

D.2. Limitations

While our approach significantly improves the scalability and adaptability of TabPFN, it has certain limitations that present opportunities for future research. Due to the current design of TabPFN, our method has been evaluated solely on classification tasks, and extending it to regression tasks remains an open challenge. As tabular foundation models continue to evolve, we anticipate that our approach can be adapted to support regression, broadening its applicability to a wider range of real-world problems. Additionally, our experiments assume that training and test instances are drawn from the same underlying distribution. However, in practical applications, distribution shifts such as covariate shift or concept drift may occur, which could impact model performance. Addressing such shifts requires robust adaptation strategies or specialized training techniques, which are beyond the scope of this work. Importantly, we emphasize that handling non-IID scenarios is not a trivial extension of existing methods but rather a distinct challenge that necessitates dedicated research efforts. Future work should explore how TabPFN-based models can be adapted to regression tasks and how their robustness to distributional shifts can be enhanced to ensure broader applicability in real-world settings.

E. Additional Experiments

Scale to Large Datasets. We evaluated BETA and the variants of TabPFN on 173 tabular datasets, excluding those with more than 100 features, which TabPFN cannot handle. These datasets were sorted by the number of rows in ascending order, and the relative improvement of these methods over TabPFN was shown in Figure 6. The results clearly show that

the performance improvement by BETA becomes more pronounced as the dataset size increases, highlighting the scalability and effectiveness of our approach in handling larger datasets. Notably, on the largest datasets in the benchmark, LocalPFN slightly outperforms BETA in terms of accuracy. However, on more other datasets, BETA demonstrates superior performance.

E.1. Performance on MultiClass Classification Tasks with More Than 10 Classes.

In addition, we further investigated the performance of BETA on classification tasks with more than 10 categories. We selected 12 classification datasets with more than 10 classes from TALENT (Ye et al., 2024b) and compared BETA with other methods. We report the mean accuracy of each method across the 12 datasets in Figure **??** and find that BETA outperforms the compared methods, demonstrating its superiority on multiclass classification tasks with a larger number of categories.

E.2. Bias-Variance Analysis, Ablation Study, and Efficiency Comparison

To evaluate the effectiveness of BETA in addressing bias and variance, we conduct comprehensive experiments across diverse dataset sizes. Our findings reveal that BETA consistently achieves lower generalization error than existing methods, demonstrating robust performance regardless of dataset scale. Bias is effectively reduced through encoder-based fine-tuning, aligning TabPFN with downstream data distributions, while variance reduction is achieved via Bagging and the introduction of multiple encoders. Importantly, our method remains effective even on small datasets, where excessive fine-tuning may otherwise increase bias. In addition, we perform an ablation study to examine the contributions of key components, including the number of encoder paths, periodic activation, Batch Ensemble, and partial parameter tuning. Our results indicate that each of these design elements plays a crucial role in enhancing BETA's performance. Lastly, we compare the inference time and parameter count of BETA against other methods to highlight its efficiency. A comprehensive analysis of bias-variance trends, ablation experiments, and efficiency comparisons, including inference time and parameter count, is provided in Appendix E.4, Appendix E.5, and Appendix E.6.

E.3. Performance on MultiClass Classification Tasks with More Than 10 Classes

To evaluate the effectiveness of BETA on multiclass classification tasks, we conducted experiments on 12 datasets containing more than 10 classes, sourced from TALENT(Ye et al., 2024b). These datasets were selected to assess the scalability and adaptability of different methods in handling more complex classification problems beyond the 10-class limitation of TabPFN. Figure 7 presents the mean accuracy of each method across these datasets. We compare BETA against several strong baselines, including KNN, XGBoost (Chen & Guestrin, 2016), RandomForest (Breiman, 2001), MLP (Gorishniy et al., 2021), ModernNCA (Ye et al., 2025), and TabM (Gorishniy et al., 2025). All methods, including BETA, were evaluated using their default hyperparameters without any tuning, ensuring a fair comparison of out-of-the-box performance.

From the results, BETA achieves the highest mean accuracy across all datasets, demonstrating superior generalization ability in high-category classification settings. This advantage stems from our integration of the Error-Correcting Output Codes (ECOC) framework, which effectively decomposes multiclass problems into multiple binary subproblems. This enables TabPFN to handle classification tasks with more than 10 classes efficiently, without requiring significant modifications or additional computational overhead. Notably, tree-based methods like XGBoost and RandomForest exhibit a decline in performance as the number of categories increases.

E.4. Performance on Real Datasets: Bias and Variance

The generalization error of TabPFN can be attributed to two primary components: **bias** and **variance**. These components are influenced by the model's architecture, pretraining assumptions, and dataset characteristics. Mathematically, the generalization error can be decomposed by analyzing the discrepancy between the model's predictions, $q_\theta(y \mid x, D_n)$, and the true conditional distribution, $p_0(y \mid x)$. Specifically, it can be expressed as:

$$q_{\boldsymbol{\theta}}\left(y \mid \boldsymbol{x}, \mathcal{D}_n\right) - p_0(y \mid \boldsymbol{x}) = \underbrace{q_{\boldsymbol{\theta}}\left(y \mid \boldsymbol{x}, \mathcal{D}_n\right) - \mathbb{E}_{\mathcal{D}_n \sim p_0^n}\left[q_{\boldsymbol{\theta}}\left(y \mid \boldsymbol{x}, \mathcal{D}_n\right)\right]}_{\text{Variance}} + \underbrace{\mathbb{E}_{\mathcal{D}_n \sim p_0^n}\left[q_{\boldsymbol{\theta}}\left(y \mid \boldsymbol{x}, \mathcal{D}_n\right)\right] - p_0(y \mid \boldsymbol{x})}_{\text{Bias}}. \tag{12}$$

Here, $\mathbb{E}[\cdot]$ denotes the expectation over training datasets D_n sampled from the true distribution p_0. Understanding these error components is crucial for addressing the limitations of TabPFN and improving its performance across diverse tabular datasets.

Revisiting the results shown in Figure 2, we observe several key phenomena that highlight the effectiveness of BETA in

reducing generalization error across varying dataset sizes:

Lowest Generalization Error Across All Dataset Sizes: Our method consistently achieves the smallest generalization error, demonstrating robust performance regardless of dataset size.

Impact of Fine-Tuning on Small Datasets: For small datasets (e.g., with only a few hundred samples), fine-tuning the encoder may slightly increase bias. This occurs because the limited data volume can lead to overfitting during fine-tuning.

Mechanisms of Bias and Variance Reduction: The bias reduction is primarily attributed to the fine-tuned encoder, which aligns TabPFN with the downstream dataset distribution, improving compatibility between the pre-trained model and the target task. Variance reduction, on the other hand, is not solely due to Bagging. The learning of multiple encoders also contributes significantly. When dataset sizes are small (e.g., fewer than 1000 samples), where each bootstrap sample covers the entire training set, training multiple encoders still introduces diversity in representations, further reducing variance.

These findings highlight the effectiveness of BETA in addressing the bias-variance tradeoff, making it a robust and scalable solution for real-world tabular learning.

E.5. Ablation Study

Influence of Encoder Path Count on Performance Improvement over TabPFN. Figure 6 (a) presents the results of an ablation experiment designed to investigate the impact of different encoder path counts on the performance of the model. The primary goal is to explore how the number of encoder paths influences the model's relative improvement over TabPFN. We display the relative improvement for various methods with different encoder path configurations. From the plot, we observe that as the number of encoder paths (denoted by K) increases, the relative performance improvement of BETA over TabPFN becomes more pronounced. This trend indicates that a higher number of encoder paths contributes positively to the model's ability to outperform TabPFN, suggesting that more encoder paths enhance the model's expressive power and its ability to capture more complex patterns in the data.

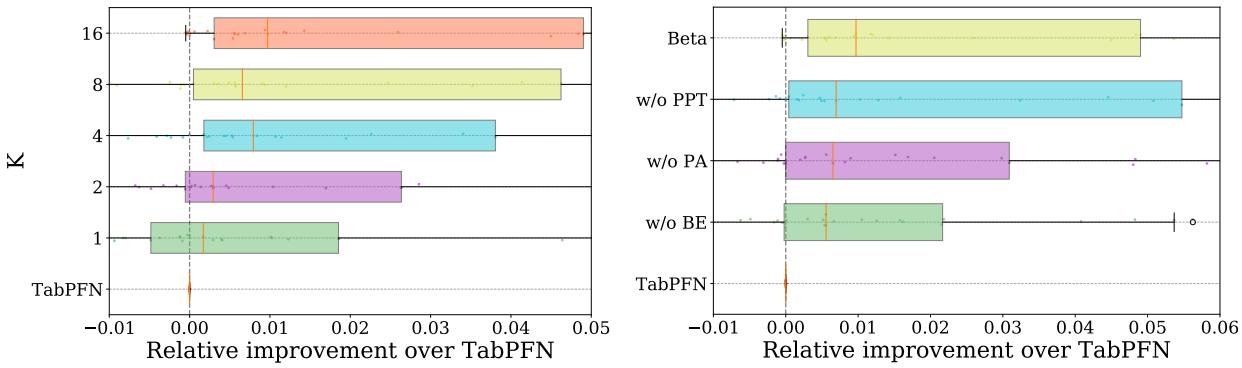
Impact of Periodic Activation, Batch Ensemble, and Partial Parameter Tuning on Model Performance. Figure 6 (b) visualizes the effect of three key components—periodic activation, Batch Ensemble, and partial parameter tuning—on the relative performance improvement of BETA over TabPFN. In this experiment, we compare the original BETA with three variations: one without periodic activation, one without Batch Ensemble, and one with full parameter tuning. The plot reveals that each of these components contributes positively to the model's performance. Specifically, removing any of these elements leads to a noticeable decline in relative improvement over TabPFN, highlighting that all parts are essential for achieving the best performance. Notably, the variation without partial parameter tuning (PPT) shows the highest upper-bound improvement, but it also incurs significantly higher computational costs. Moreover, the median relative improvement decreases compared to BETA , further emphasizing the superior efficiency of our approach. This suggests that periodic activation, Batch Ensemble, and partial parameter tuning each play a crucial role in enhancing the model's ability to outperform TabPFN, and the absence of any of these components compromises the model's ability to leverage its full potential while also increasing computational resource usage.

E.6. Comparison of inference time and the number of learnable parameters.

We compared the inference time (in seconds) of BETA with other methods, as well as the average rank in the main experiment. Additionally, the number of learnable parameters was roughly estimated by the size of the stored checkpoint. The results confirm that BETA not only delivers state-of-the-art classification accuracy but also maintains low inference time and reduced memory overhead, making it highly scalable and efficient for various tabular classification tasks.

Table 3: Comparison of Inference Time, Average Rank, and Number of Learnable Parameters. All metrics are computed as the mean values across multiple datasets.

Metric	BETA	TabPFN	LocalPFN	MixturePFN	FT-T	TabR
Inference Time (s)	0.91	0.78	12.51	3.24	**0.36**	0.38
Average Rank	**6.76**	15.63	12.15	12.86	13.27	9.26
Checkpoint Size (MB)	**0.60**	0	98.65	21.42	7.15	14.07



(a) Infulence of K

(b) Influence of some important components

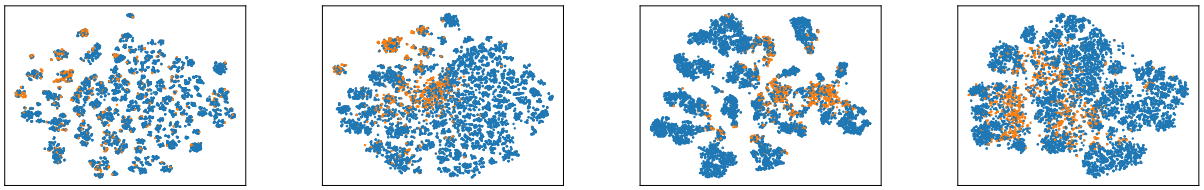
Figure 8: (a) The box plot demonstrates the effect of varying the number of encoder paths, K, on the relative improvement over TabPFN. The horizontal axis represents the relative improvement in performance, while the vertical axis lists numbers of different encoder path settings; (b) This box plot illustrates the effect of different components—periodic activation (PA), Batch Ensemble (BE), and partial parameter tuning (PPT)—on the relative improvement of BETA over TabPFN. The plot compares the original BETAmodel with three variations: (1) BETA with partial parameter tuning (w/o PPT), (2) BETA without periodic activation (w/o PA), and (3) BETA without Batch Ensemble (w/o BE).

E.7. Visialzation results

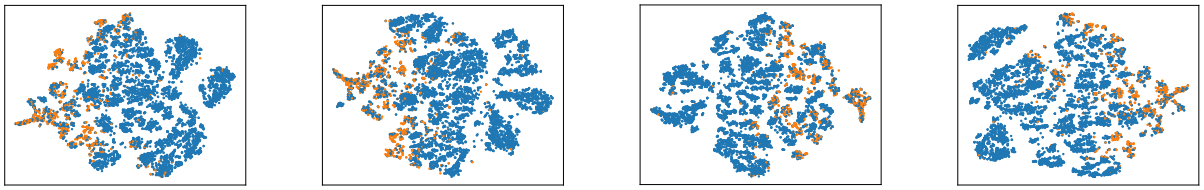
To gain deeper insights into the properties of BETA, we visualize the learned embeddings $\boldsymbol{E}_{\Phi}(\boldsymbol{x})$ for BETA, MLP, ModernNCA, and TabR using T-SNE (Van der Maaten & Hinton, 2008). As shown in Figure 9, we present the embeddings of three datasets (*Bank*, *KDD*, and *Spambase*), comparing different methods as well as the representations learned by different encoder paths within BETA.

All deep tabular methods effectively transform the feature space, making it more conducive for classification compared to raw input features. MLP produces well-separated clusters, grouping same-class samples into distinct, compact regions. In contrast, TabR and ModernNCA form multiple clusters for the same class, positioning similar samples closer to each other while maintaining local class separability. Unlike these methods, BETA does not enforce strict clustering of same-class samples into single compact groups. This behavior aligns with TabPFN's pretraining paradigm, where the model does not require fully separable embeddings to make accurate predictions.

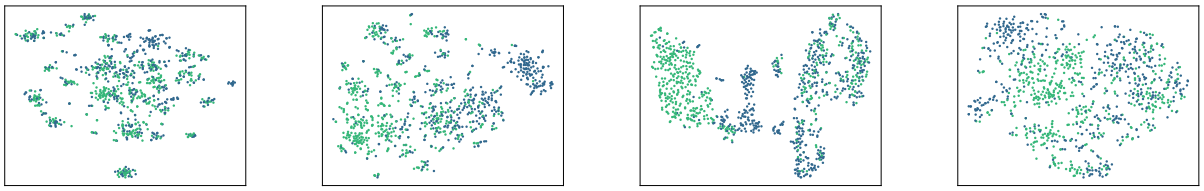
Moreover, we observe that different encoder paths in BETA map raw features into diverse representation spaces, contributing to variance reduction and improved robustness. The variation among encoder paths enhances the diversity of the learned representations, which is a key factor in BETA's superior generalization. The visualization further confirms that BETA preserves the flexibility of the feature space while effectively adapting to downstream classification tasks.



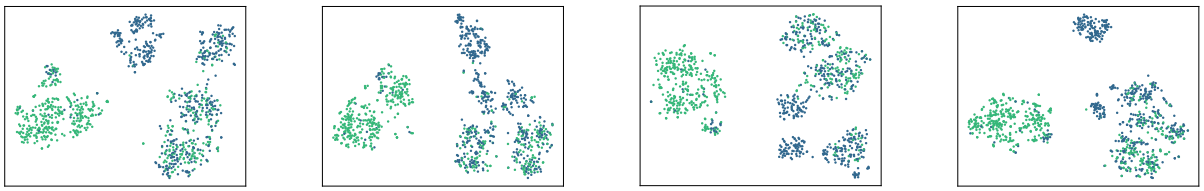
(a) Bank Raw Feature (b) Bank MLP (c) Bank ModernNCA (d) Bank TabR



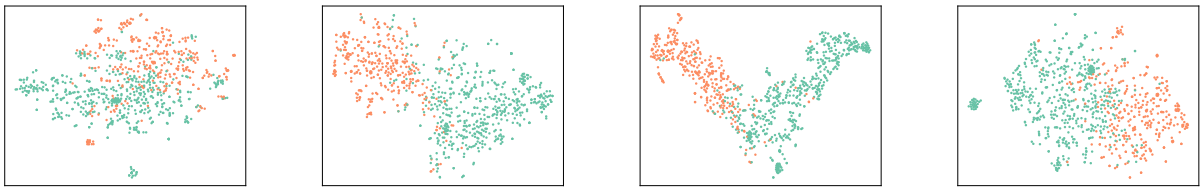
(e) Bank BETA 1 (f) Bank BETA 2 (g) Bank BETA 3 (h) Bank BETA 4



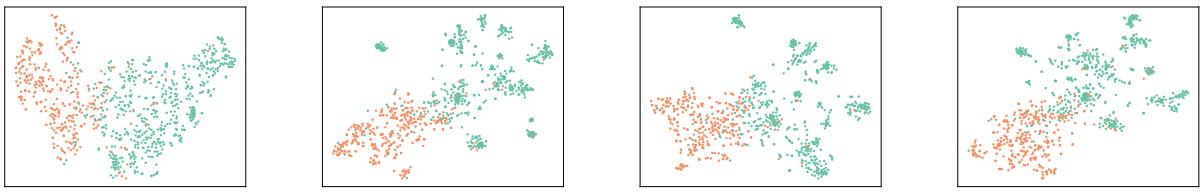
(a) KDD Raw Feature (b) KDD MLP (c) KDD ModernNCA (d) KDD TabR



(e) KDD BETA 1 (f) KDD BETA 2 (g) KDD BETA 3 (h) KDD BETA 4



(a) Spambase Raw Feature (b) Spambase MLP (c) Spambase ModernNCA (d) Spambase TabR



(e) Spambase BETA 1 (f) Spambase BETA 2 (g) Spambase BETA 3 (h) Spambase BETA 4

Figure 9: Visualization of the embedding space of different methods.