Transformers for Tabular Data

Oualid Missaoui

Agenda

- Bayesian Supervised learning
- Prior-Data Fitted Networks (PFNs)
- Tab-PFN
- References

Bayesian supervised learning: hypotheses, data, and notation

- In the **Bayesian framework** for supervised learning, the **prior** defines a *space of hypotheses* Φ on the relationship between inputs x and outputs y.
- Each hypothesis $\phi \in \Phi$ can be seen as a **mechanism** that generates a data distribution from which we can draw samples forming a dataset.
- Example:
  - Given a prior based on **structural causal models (SCMs)**, Φ represents the space of all SCMs.
  - A specific hypothesis ϕ corresponds to **one concrete SCM**, through which training and test data are generated.
- Thus, a dataset is composed of:

$$D_{\text{train}} := \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

where training data have observed labels and test data are held out for prediction.

Posterior Predictive Distribution (PPD)

- The **Posterior Predictive Distribution (PPD)** for a test sample x_{test} specifies the distribution of its label $p(\cdot \mid x_{\text{test}}, D_{\text{train}})$, conditioned on the observed training set.

- The PPD is obtained by integrating over all hypotheses $\phi \in \Phi$, weighted by their posterior probability:

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

- Here:
  - $p(\phi) \rightarrow$ prior probability of a hypothesis
  - $p(D|\phi) \rightarrow$ likelihood of the data under ϕ
  - $p(y|x, \phi) \rightarrow$ predictive likelihood for a given ϕ

> **Intuition:**
> The Bayesian learner averages predictions from all possible mechanisms ϕ,
> weighting each by how well it explains the observed data D.

Posterior Predictive Distribution (PPD) Derivation

We want $p(y \mid x_{\text{test}}, D_{\text{train}})$. Start by marginalizing the hypothesis ϕ:

$$\begin{aligned} p(y \mid x_{\text{test}}, D_{\text{train}}) &= \int_{\Phi} p(y, \phi \mid x_{\text{test}}, D_{\text{train}})\, d\phi \\ &= \int_{\Phi} p(y \mid x_{\text{test}}, \phi, D_{\text{train}})\, p(\phi \mid D_{\text{train}})\, d\phi. \end{aligned}$$

Assume conditional independence of the test label given (x_{test}, ϕ):

$$p(y \mid x_{\text{test}}, \phi, D_{\text{train}}) = p(y \mid x_{\text{test}}, \phi).$$

Use Bayes' rule for the posterior over hypotheses:

$$p(\phi \mid D_{\text{train}}) = \frac{p(D_{\text{train}} \mid \phi)\, p(\phi)}{\int_{\Phi} p(D_{\text{train}} \mid \phi')\, p(\phi')\, d\phi'}.$$

Substitute to obtain both the **normalized** and **proportional** forms:

$$\begin{aligned} p(y \mid x_{\text{test}}, D_{\text{train}}) &= \frac{\displaystyle\int_{\Phi} p(y \mid x_{\text{test}}, \phi)\, p(D_{\text{train}} \mid \phi)\, p(\phi)\, d\phi}{\displaystyle\int_{\Phi} p(D_{\text{train}} \mid \phi)\, p(\phi)\, d\phi} \\ &\propto \int_{\Phi} p(y \mid x_{\text{test}}, \phi)\, p(D_{\text{train}} \mid \phi)\, p(\phi)\, d\phi. \end{aligned}$$

Interpretation: the PPD averages predictions under all mechanisms ϕ, weighted by how well each explains D_{train} and by its prior mass.

Prior-Data Fitted Networks (PFNs)

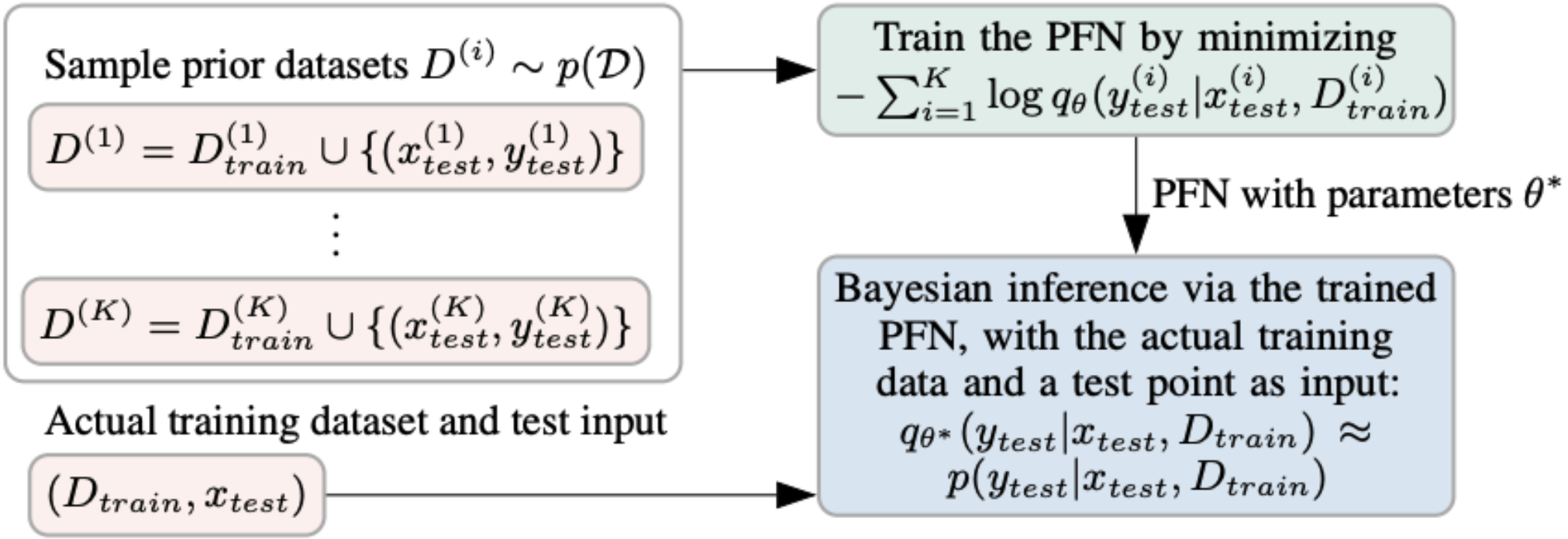


Figure 1: A visualization of Prior-Data Fitted Networks (PFNs). We sample datasets from a prior and fit a PFN on hold-out examples of these datasets. Given an actual dataset, we feed it and a test point to the PFN and obtain an approximation to Bayesian inference in a single forward propagation.

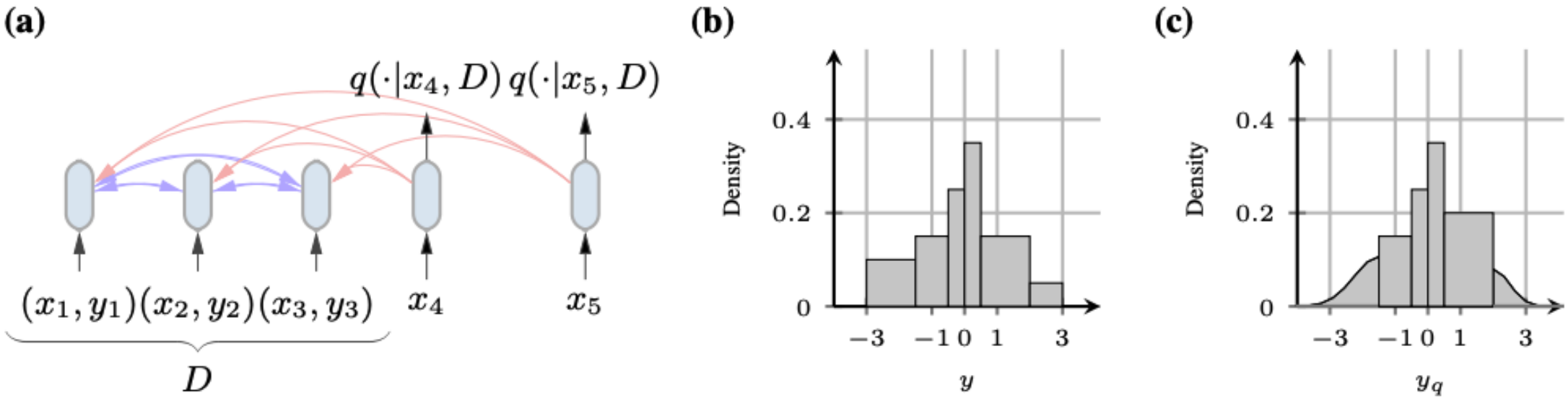


Figure 2: (a) A visualization of the Transformer for $n = 3$ input pairs and $m = 2$ queries. Every bar is the representation of one input and the arrows show what each representation can attend to. (b-c) A visualisation of the Riemann distribution, with and without bounded support.

Classic Bayesian vs PFN

Aspect	Classical Bayesian (MCMC / VI for PPD)	PFN (Amortized PPD via Transformer)
What is approximated?	The **integral** $$\int_{\Phi} p(y \mid x, \phi)\, p(D \mid \phi)\, p(\phi)\, d\phi$$ **per task**, by sampling/optimization.	The **mapping** $$(X_{\text{train}}, Y_{\text{train}}, X_*) \mapsto \hat{p}(Y_* \mid X_*, D_{\text{train}})$$ learned **once** across many synthetic tasks.
When is the heavy compute?	**At test time**: chains/optimization run for each new dataset.	**At meta-train time**: train Transformer on tasks from a meta-prior; **test time** is a single forward pass.
Per-task latency	High (minutes → hours) depending on model/data.	Low (milliseconds → seconds).
Statistical guarantees	Asymptotically exact (MCMC); VI is approximate but analyzable.	No exactness; quality depends on meta-prior/task diversity and generalization.
Prior handling	Explicit **model + prior** specified and inspected.	**Implicit prior** encoded by training task generator and learned weights.
Flexibility across tasks	Re-specify model/prior; rerun inference.	Same network handles many tasks if meta-prior covers them.
Scaling with data size	Cost grows with data and model complexity.	Inference cost ~ one forward pass; attention/memory still scale with tokens.
Failure modes	Poor mixing, bad variational family, hyperparameter sensitivity.	Meta-prior mismatch, domain shift, out-of-support features/labels.
Interpretability	Clear posterior over ϕ; diagnostics available.	Black-box; posterior not explicit (only predictive).
Best use case	Few tasks, need exact Bayesian diagnostics/explainability.	Many tasks, tight latency budget, need "PPD-like" predictions fast.
Transferability	Strong *within-model* transfer via reuse of priors/posteriors across related datasets, but typically **requires rerunning inference** on each new dataset.	Strong **across-dataset** transfer **if** new tasks are within the meta-prior's support; zero-shot generalization possible. **Degrades under domain shift** unless the meta-prior/training tasks are expanded or the model is fine-tuned.

TabPFN

- Uses an encoder-style Transformer without positional encodings for permutation-invariant set processing.
- Training tokens self-attend; query tokens cross-attend to the training tokens (no autoregressive, no separate decoder stack)

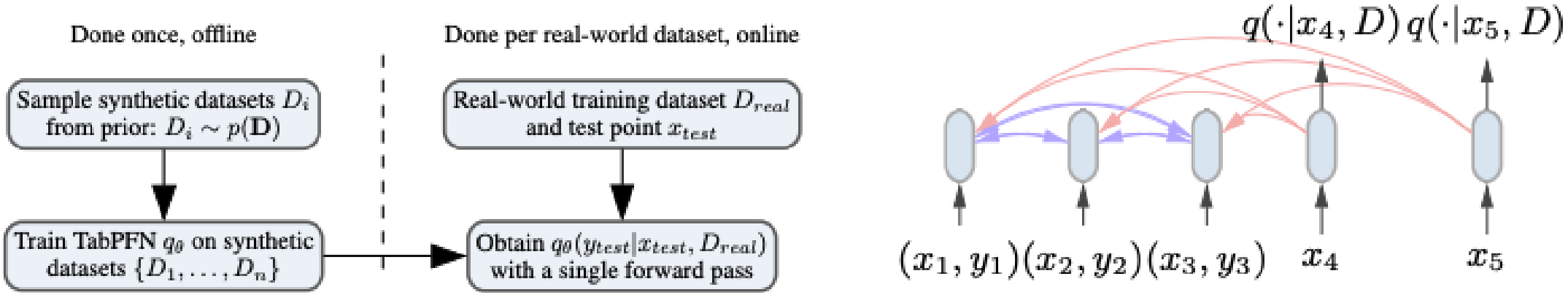
Published as a conference paper at ICLR 2023

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

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

ABSTRACT

We present TabPFN, a trained Transformer that can do supervised classification for small tabular datasets in *less than a second*, needs no hyperparameter tuning and is competitive with state-of-the-art classification methods. TabPFN performs in-context learning (ICL), it learns to make predictions using sequences of labeled examples (x, f(x)) given in the input, without requiring further parameter updates. TabPFN is fully entailed in the weights of our network, which accepts training and test samples as a set-valued input and yields predictions for the entire test set in a single forward pass. TabPFN is a Prior-Data Fitted Network (PFN) and is trained offline once, to approximate Bayesian inference on synthetic datasets drawn from our prior. This prior incorporates ideas from causal reasoning: It entails a large space of structural causal models with a preference for simple structures. On the 18 datasets in the OpenML-CC18 suite that contain up to 1 000 training data points, up to 100 purely numerical features without missing values, and up to 10 classes, we show that our method clearly outperforms boosted trees and performs on par with complex state-of-the-art AutoML systems with up to 230× speedup. This increases to a 5 700× speedup when using a GPU. We also validate these results on an additional 67 small numerical datasets from OpenML. We provide all our code, the trained TabPFN, an interactive browser demo and a Colab notebook at https://github.com/automl/TabPFN.



(a) Prior-fitting and inference

(b) Architecture and attention mechanism

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

Constructing the Synthetic Prior Datasets in TabPFN

- **Objective:** build a *meta-training distribution* of datasets that reflects the diversity of real-world tabular tasks.
- Each dataset (task) is sampled from a *meta-prior* over generative mechanisms:

$$p(D) = \int p(D \mid \phi)\, p(\phi)\, d\phi$$

- **Sampling procedure for each synthetic task:**
  1. **Choose task type:** regression or classification.
  2. **Sample number of features** $d \sim \mathrm{Uniform}\{2, 100\}$
     (dimension varies across tasks).
  3. **Generate feature covariance:**
     - Random correlation matrix Σ with block-structured or sparse correlations.
     - Draw inputs $X_{\text{train}} \sim \mathcal{N}(0, \Sigma)$.
  4. **Define ground-truth mechanism** $f_\phi(x)$**:**
     Random linear, polynomial, logistic, or small MLP function with random weights.
  5. **Generate labels:**
     - Regression: $y = f_\phi(x) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \sigma^2)$
     - Classification: $y \sim \mathrm{Categorical}(\mathrm{softmax}(f_\phi(x)))$
  6. **Split into train/test sets:**
     $D_{\text{train}}, D_{\text{test}}$
- **Outcome:** millions of small, diverse datasets sampled from $p(D)$.
  Each acts as a single draw from the *synthetic prior* over real-world tabular problems.

Generative Mechanisms Used for Task Sampling

TabPFN's meta-prior combines multiple *families of mechanisms* to expose the model to a broad range of inductive biases.

Mechanism	Description	Purpose / What it teaches
Linear / Logistic regression	$f_\phi(x) = x^\top w + b$ or $y \sim \text{Bernoulli}(\sigma(x^\top w + b))$	Captures linear dependencies and class-separation geometry.
Polynomial models	Random polynomial expansions of inputs	Adds controlled non-linearity and feature interactions.
Bayesian Neural Networks (BNNs)	Sample weights $w \sim p(w)$; small MLPs with priors on weights	Exposes the PFN to smooth nonlinear mappings with parameter uncertainty.
Decision-tree mechanisms	Piecewise-constant or piecewise-linear partitions of feature space	Teaches handling of discontinuities and threshold-type features.
Structural Causal Models (SCMs)	Sample DAG, structural equations, and noises; generate data via causal graph	Encodes causal dependencies, confounding, and heteroscedasticity.
Mixture & noise models	Gaussian mixtures, heteroscedastic or correlated noise patterns	Builds robustness to multimodal or noisy data distributions.

Meta-prior construction:

By mixing these generators with random hyperparameters (feature correlations, noise levels, nonlinearities, causal graphs), TabPFN forms a **synthetic universe of tabular datasets**.

> **Key intuition:** the diversity of mechanisms defines the *support* of the learned prior —
> the broader the generator family, the more generalizable the amortized Bayesian predictor.

Architectural Differences

Aspect	TabPFN / PFN-Style Transformer	Vaswani et al. (2017) Transformer
Overall layout	Single **encoder-style stack** specialized for sets of (x, y) training examples plus query x; no separate decoder	**Encoder–decoder** (sequence-to-sequence) architecture
Token semantics	Tokens represent **training examples** and **query examples**	Tokens represent **positions in a sequence** (words/subwords)
Positional encodings	❌ **None** — ensures permutation invariance over the training set	✅ **Sinusoidal positional encodings** added to encoder & decoder inputs
Attention pattern	**Training tokens:** self-attend (including diagonal). **Query tokens:** cross-attend to training tokens only (no query↔query).	**Encoder:** full self-attention. **Decoder:** masked self-attention + cross-attention to encoder.
Masking strategy	Specialized masks enforce query → train only; training tokens can attend to each other	Causal mask in decoder prevents attention to future positions
Parameter sharing	**Two attention modules share weights** (train↔query flow)	No weight sharing between encoder and decoder stacks
Invariance goal	Permutation-invariant over examples	Sequence-order sensitive
I/O heads	Linear projections for inputs/queries; classification head for outputs	LM or task-specific output head (softmax over vocab)
Autoregression	❌ None — predictions made in parallel for all queries	✅ Yes — decoder predicts next token sequentially
Stack type	Encoder-style, **non-autoregressive**	Full encoder + decoder, **autoregressive**

Hyperparameter and Complexity Differences

Hyperparameter	TabPFN (ICLR 2023)	Vaswani et al. (2017, Base)
# Layers (depth)	12 (encoder-style)	6 encoder + 6 decoder
Model dimension (d_{model})	512	512
Feed-forward dimension (d_ff)	1024	2048
# Attention heads	4	8
Positional encodings	None	Sinusoidal (fixed, non-trainable)
Attention cost per forward pass	$n^2 + nm$ (train–train + query→train)	$\mathcal{O}(L^2 \cdot d_{model})$ per layer
Parameter count (approx.)	≈ 25.8 M	≈ 60 M + (depends on vocab size)
Permutation invariance	✅ Yes	❌ No
Autoregressive decoding	❌ No	✅ Yes
Positional dependence	None	Explicit via sinusoidal PEs

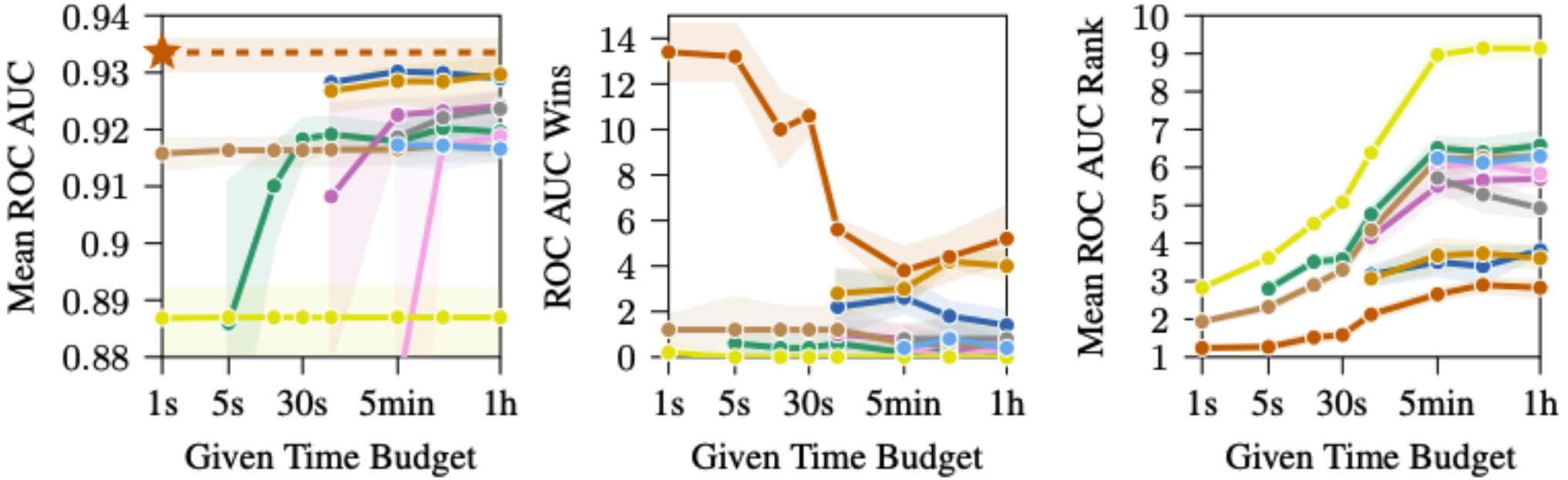


Figure 5: ROC AUC as a function of the time allowed to train & tune methods, on 18 numerical datasets from the OpenML-CC18 Benchmark. We report the mean, mean wins and rank and 95% confidence interval across 5 splits for increasing budgets. The red star indicates performance of TabPFN with 32 permutations (which requires 0.62s on GPU). We report more results in Table 1.

TabPFN v2

Accurate predictions on small data with a tabular foundation model

https://doi.org/10.1038/s41586-024-08328-6

Received: 17 May 2024

Accepted: 31 October 2024

Published online: 8 January 2025

Open access

Check for updates

Noah Hollmann[1,2,3,7]✉, Samuel Müller[1,7]✉, Lennart Purucker[1], Arjun Krishnakumar[1], Max Körfer[1], Shi Bin Hoo[1], Robin Tibor Schirrmeister[4,5] & Frank Hutter[1,3,6]✉

Tabular data, spreadsheets organized in rows and columns, are ubiquitous across scientific fields, from biomedicine to particle physics to economics and climate science[1,2]. The fundamental prediction task of filling in missing values of a label column based on the rest of the columns is essential for various applications as diverse as biomedical risk models, drug discovery and materials science. Although deep learning has revolutionized learning from raw data and led to numerous high-profile success stories[3–5], gradient-boosted decision trees[6–9] have dominated tabular data for the past 20 years. Here we present the Tabular Prior-data Fitted Network (TabPFN), a tabular foundation model that outperforms all previous methods on datasets with up to 10,000 samples by a wide margin, using substantially less training time. In 2.8 s, TabPFN outperforms an ensemble of the strongest baselines tuned for 4 h in a classification setting. As a generative transformer-based foundation model, this model also allows fine-tuning, data generation, density estimation and learning reusable embeddings. TabPFN is a learning algorithm that is itself learned across millions of synthetic datasets, demonstrating the power of this approach for algorithm development. By improving modelling abilities across diverse fields, TabPFN has the potential to accelerate scientific discovery and enhance important decision-making in various domains.

TabPFN v2

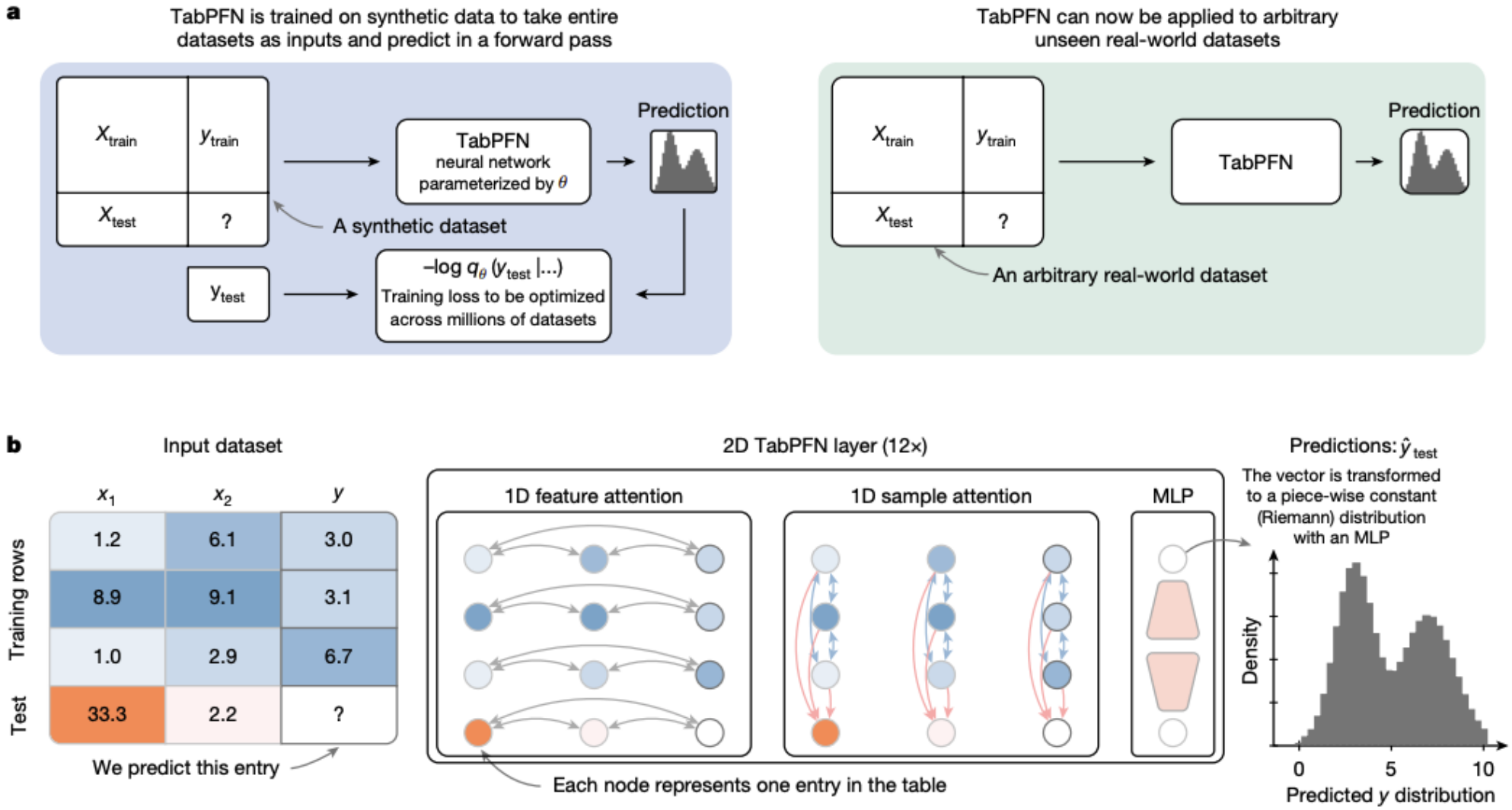


Fig. 1 | Overview of the proposed method. a, The high-level overview of TabPFN pre-training and usage. **b**, The TabPFN architecture. We train a model to solve more than 100 million synthetic tasks. Our architecture is an adaptation of the standard transformer encoder that is adapted for the two-dimensional data encountered in tables.

Noah Hollmann et al (2025), Accurate predictions on small data with a tabular foundation model

TabPFN-TS

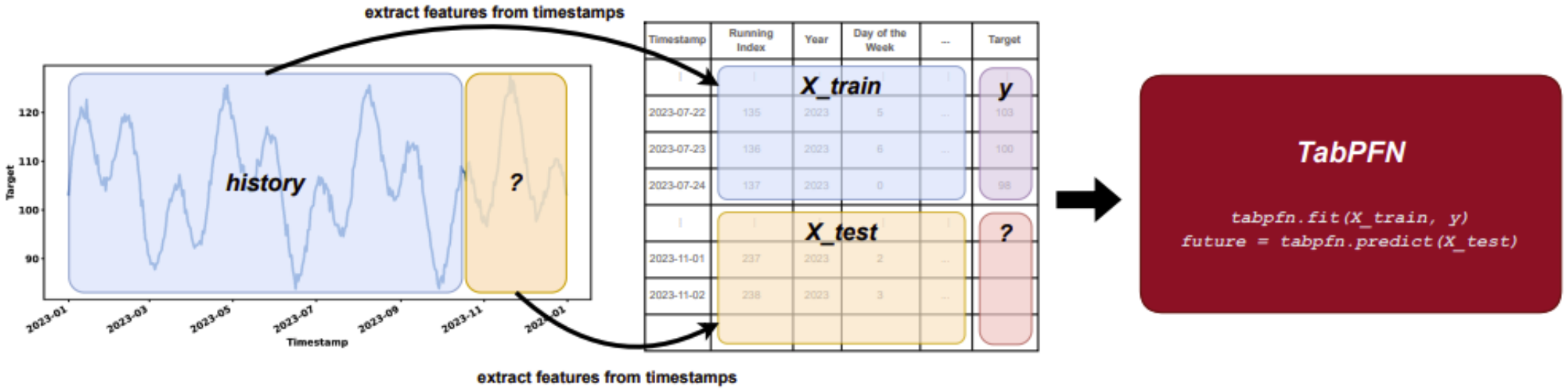


Figure 1: Overview of TabPFN-TS. Given a time series, we derive features from the timestamps to form both `X_train` and `X_test`. The target values of the history are used as `y_train`. These three variables are then used by TabPFN to predict the target values of the future timestamps.

Shi Bin Hoo et al (2025), The Tabular Foundation Model TabPFN Outperforms Specialized Time Series Forecasting Models Based on Simple Features

TabPFN-TS

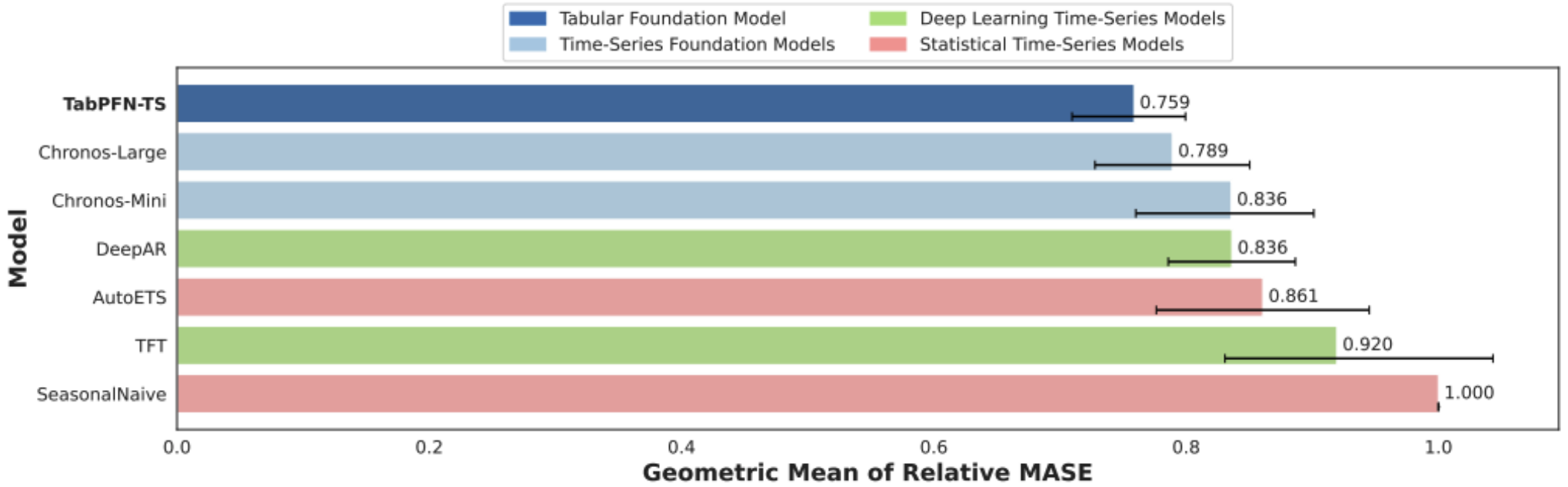


Figure 2: Forecasting performance of various models on 24 datasets. MAE scores are normalized using the scores of Seasonal Naive to compute Relative MASE, then aggregated via geometric mean over the datasets. 95% confidence interval is included. Lower is better.

Shi Bin Hoo et al (2025), The Tabular Foundation Model TabPFN Outperforms Specialized Time Series Forecasting Models Based on Simple Features

TabPFN-2.5

PRIOR

TabPFN-2.5: Advancing the State of the Art in Tabular Foundation Models

Prior Labs Team[1]
[1]The list of contributors can be found in the appendix.

The first tabular foundation model, TabPFN, and its successor TabPFNv2 have impacted tabular AI substantially, with dozens of methods building on it and hundreds of applications across different use cases.
This report introduces TabPFN-2.5, the next generation of our tabular foundation model, scaling to 20× data cells compared to TabPFNv2. On industry standard benchmarks with up to 50,000 data points and 2,000 features, TabPFN-2.5 substantially outperforms tuned tree-based models and matches the accuracy of AutoGluon 1.4, a complex four-hour tuned ensemble that even includes the previous TabPFNv2.
For production use cases, we introduce a new distillation engine that converts TabPFN-2.5 into a compact MLP or tree ensemble, preserving most of its accuracy while delivering orders-of-magnitude lower latency and plug-and-play deployment.
This new release will immediately strengthen the performance of the many applications and methods already built on the TabPFN ecosystem.

Date: November 6, 2025
Website: https://priorlabs.ai/
Docs: https://docs.priorlabs.ai/overview
PyPI: `pip install tabpfn`
License: `TABPFN-2.5 License v1.0` (see Section 6 for details)
Contact: hello@priorlabs.ai

TabPFN-2.5

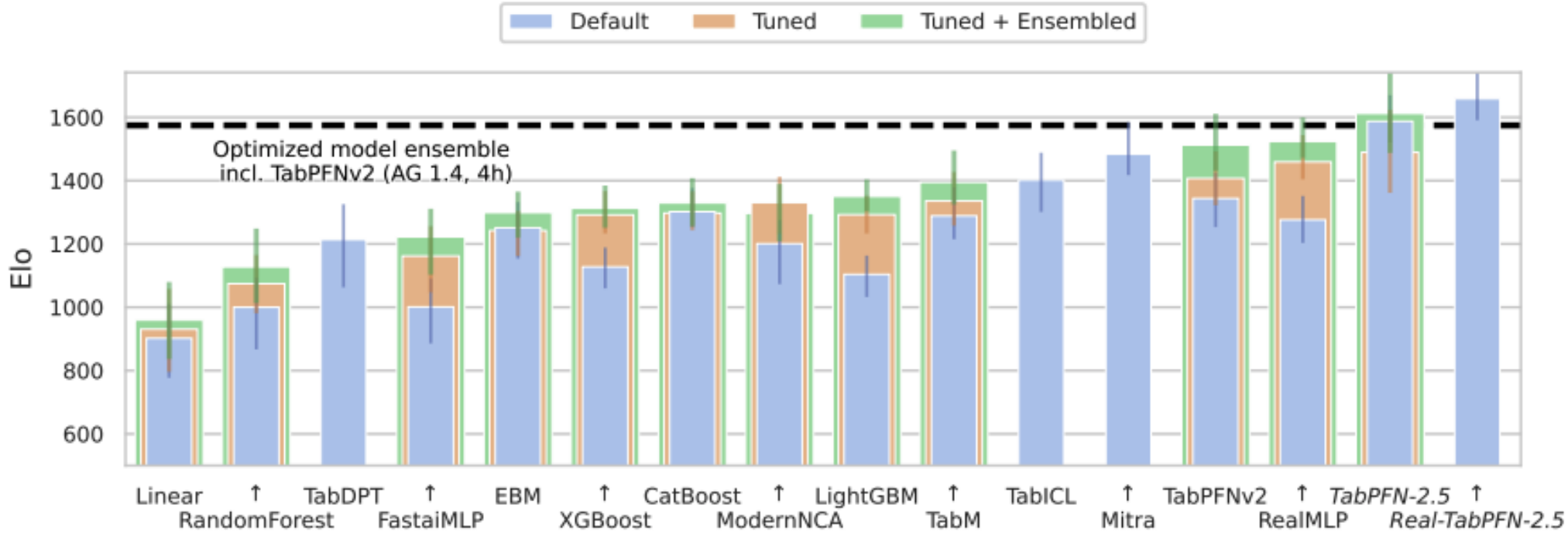


Figure 1: TabPFN-2.5 performance on the standard TabArena-lite benchmark, TabPFNv2 classification subset. TabPFN-2.5 outperforms any other model in a forward pass, and marks a strong leap from TabPFNv2. When fine-tuned on real data, Real-TabPFN-2.5 shows even stronger performance. The horizontal dotted line stands for AutoGluon 1.4 extreme mode tuned for 4 hours, an ensemble of models including TabPFNv2 [1].

TabPFN Capacity Evolution

Version	Max Training Rows	Max Features	Approx. Total Data Cells	Key Advances
TabPFN (v1, 2022, 2023)	**1 000**	**100**	$\sim 1 \times 10^5$	Original model trained on small synthetic tasks (classification only). Focus on few-shot tabular reasoning.
TabPFN v2 (Nature 2025)	**10 000**	**500**	$\sim 5 \times 10^6$	Major scale-up via efficient attention kernels, caching, and mixed-precision training. Introduced regression, causal, and multimodal extensions.
TabPFN 2.5 (Tech Report 2025)	**50 000**	**2 000**	$\sim 1 \times 10^8$ ($\approx 20\times$ v2)	Optimized for large-batch inference; supports 20× more data cells, improved distillation, and hybrid ICL + fine-tuning modes.

References

- Samuel Muller, Noah Hollmann, Sebastian Pineda, Josif Grabocka, Frank Hutter (2022), Transformers can do Bayesian Inference
- Noah Hollmann, Samuel Müller, Katharina Eggensperger, Frank Hutter (2023), TabPFN: A transformer that solves small Tabular classification problems in a second
- Noah Hollmann, Samuel Muller, Lennart Purucker, Arjun Krishnakumar, Max Korfer, Shi Bin Hoo, Robin Tibor Schirrmeister, Frank Hutter (2025), Accurate predictions on small data with a tabular foundation model
- Shi Bin Hoo, Samuel Muller, David Salinas, Frank Hutter (2025), The Tabular Foundation Model TabPFN Outperforms Specialized Time Series Forecasting Models Based on Simple Features
- Prior Labs Team (2025), TabPFN-2.5: Advancing the start of the Art in Tabular Foundation Models


```
pip install
tabpfn
```