TabM: Advancing Tabular Deep Learning with Parameter-Efficient Ensembling

Anonymous authors
Paper under double-blind review

Abstract

Deep learning architectures for supervised learning on tabular data range from simple multilayer perceptrons (MLP) to sophisticated Transformers and retrieval-augmented methods. This study highlights a major, yet so far overlooked opportunity for substantially improving tabular MLPs; namely, parameter-efficient ensembling – a paradigm for imitating an ensemble of models with just one model. We start by describing TabM – a simple model based on MLP and BatchEnsemble (an existing technique), improved with our custom modifications. Then, we perform a large scale evaluation of tabular DL architectures on public benchmarks in terms of both task performance and efficiency, which renders the landscape of tabular DL in a new light. In particular, we find that TabM outperforms prior tabular DL models, while the complexity of attention- and retrieval-based methods does not pay off. Lastly, we conduct a detailed empirical analysis, that sheds some light on the high performance of TabM. For example, we show that parameter-efficient ensembling is a highly effective way to reduce overfitting and improve optimization dynamics of tabular MLPs. Overall, our work brings an impactful technique to tabular DL, analyses its behaviour, and advances the performance-efficiency tradeoff with TabM – a simple and powerful baseline for researchers and practitioners.

1 Introduction

Supervised learning on tabular data is a popular practical ML scenario in a wide range of industrial applications. Among classic non-deep-learning methods, the state-of-the-art solution for such tasks is gradient-boosted decision trees (GBDT) (Prokhorenkova et al., 2018; Chen & Guestrin, 2016; Ke et al., 2017). Deep learning (DL) models for tabular data, in turn, are reportedly improving, and the most recent works claim to perform on par or even outperform GBDT on academic benchmarks (Hollmann et al., 2023; Chen et al., 2023b;a; Gorishniy et al., 2024).

However, from the practical perspective, it is unclear if tabular DL offers any obvious go-to baselines beyond simple architectures in the spirit of a multilayer perceptron (MLP). *First*, the scale and consistency of performance improvements of new methods w.r.t. simple MLP-like baselines are not always explicitly analyzed in the literature. Thus, one has to infer those statistics from numerous per-dataset performance scores, which makes it hard to reason about the progress. At the same time, due to the extreme diversity of tabular datasets, consistency is an especially valuable and hard-to-achieve property for a hypothetical go-to baseline. *Second*, efficiency-related properties, such as training time, and especially inference throughput, sometimes receive less attention. While methods are usually equally affordable on small-to-medium datasets (e.g. $<$100K objects), their applicability to larger datasets remains uncertain. In this work, we revisit existing tabular DL methods, and find that non-MLP models do not yet offer a convincing replacement for MLP-like models.

At the same time, we identify a previously overlooked path towards more powerful, reliable and reasonably efficient tabular DL models. Our story starts with an observation that BatchEnsemble (Wen et al., 2020) – a technique that allows one model to efficiently imitate an ensemble of models – is a highly effective modification for tabular MLPs. Then, we analyze and improve BatchEnsemble-based MLPs, which results in our model TabM. Drawing an informal parallel with GBDT, TabM can also be viewed as a simple base model (MLP) combined with an ensembling-like technique, providing high performance, simple implementation and ease of use, all at once.

Given the massive positive impact of a long-existing method (BatchEnsemble) on a long-existing baseline (MLP), we suggest that the lack of a powerful ensemble-like tabular architecture has been a prominent gap in tabular DL for too long time. Our work closes this gap and offers a new powerful and practical baseline to practitioners and researchers.

Main contributions. We summarize our main contributions as follows:

1. We present TabM – a simple deep learning architecture for supervised learning on tabular data. TabM is a combination of MLP, BatchEnsemble and custom modifications (e.g. an improved initialization strategy). TabM easily competes with GBDT models and outperforms prior tabular DL models, while being more efficient than attention- and retrieval-based DL models.
2. We provide a fresh perspective on tabular DL models in a large scale evaluation along four dimensions: task performance, performance consistency, training time and inference throughput. One of our findings is that MLP-like models, including TabM, hit an appealing performance-efficiency tradeoff, which is rather not the case for attention- and retrieval-based models.
3. We conduct a detailed empirical analysis that gives an intuition on the effectiveness of TabM compared to plain MLPs. In particular, we observe that TabM exhibits significantly improved training dynamics, including reduced overfitting and smaller variance of the stochastic gradients.

2 RELATED WORK

Decision-tree-based models. Gradient-boosted decision trees (GBDT) (Prokhorenkova et al., 2018; Chen & Guestrin, 2016; Ke et al., 2017) is a powerful baseline for tabular tasks. GBDT is a classic machine learning model, while our model TabM is a deep learning model.

Tabular deep learning architectures. A large number of deep learning architectures for tabular data has been proposed over the recent years. That includes attention-based architectures (Song et al., 2019; Gorishniy et al., 2021; Somepalli et al., 2021; Kossen et al., 2021; Yan et al., 2023), retrieval-augmented architectures (Somepalli et al., 2021; Kossen et al., 2021; Gorishniy et al., 2024; Ye et al., 2024), MLP-like models (Gorishniy et al., 2021; Klambauer et al., 2017; Wang et al., 2020) and others (Arik & Pfister, 2020; Popov et al., 2020; Chen et al., 2023b; Marton et al., 2024; Hollmann et al., 2023). Compared to prior work, the key difference of our model TabM is its computation flow, where one TabM imitates an ensemble of MLPs by producing multiple independently trained predictions. Prior attempts to bring ensemble-like elements to tabular DL (Badirli et al., 2020; Popov et al., 2020) were not found promising (Gorishniy et al., 2021).

Also, being a simple feed-forward MLP-based model, TabM is significantly more efficient than some of the prior work. For example, compared to attention-based models, TabM does not suffer from quadratic computational complexity w.r.t. the dataset dimensions. Compared to retrieval-based models, TabM is easily applicable to large datasets.

Improving tabular MLP-like models. Multiple recent studies achieved competitive performance with MLP-like architectures on tabular tasks by applying architectural modifications (Gorishniy et al., 2022), regularizations (Kadra et al., 2021; Jeffares et al., 2023a; Holzmüller et al., 2024), pretraining (Bahri et al., 2021; Rubachev et al., 2022). Thus, it seems that tabular MLPs have good potential, but one has to deal with overfitting and optimization issues to reveal that potential. In line with prior studies, our work also renders MLP as a capable tabular backbone and effectively addresses the aforementioned issues by applying the parameter-efficient "ensembling" method BatchEnsemble (Wen et al., 2020). Our approach is orthogonal to (and compatible with) training techniques like regularization or pretraining, and with any architectural advancements in MLPs and feature embeddings (Gorishniy et al., 2022).

Deep ensembles. In this paper, by a deep ensemble, we imply multiple DL models of the same architecture trained independently (Jeffares et al., 2023b) for the same task under different random seeds (i.e. with different initializations, training batch sequences, etc.). The prediction of a deep ensemble is the average prediction of its members. Deep ensembles often significantly outperform single DL models of the same architecture (Fort et al., 2020), and can excel in other tasks like uncertainty estimation or out-of-distribution detection (Lakshminarayanan et al., 2017). It was observed that individual members of deep ensembles can learn to extract diverse information from the input, and the power of deep ensembles depends on this diversity (Allen-Zhu & Li, 2023). The main drawback of deep ensembles is the cost and inconvenience of training and using multiple models.

Parameter-efficient deep "ensembles". To achieve the performance of deep ensembles at a lower cost, multiple studies proposed architectures that imitate ensembles by producing multiple independently trained predictions (Lee et al., 2015; Zhang et al., 2020; Wen et al., 2020; Havasi et al., 2021; Turkoglu et al., 2022) (there are also non-architectural approaches to efficient ensembling, e.g. FGE (Garipov et al., 2018), but they are less relevant to our work). Despite being *single* architectures, such approaches are sometimes informally called "(parameter-efficient) ensembles". Usually, in such ensemble-like models, each of the predictions relies on a large amount of weights shared for all predictions, and a small amount of prediction-specific weights. In our work, by applying and customizing BatchEnsemble (Wen et al., 2020), we highlight parameter-efficient ensembling as an impactful paradigm for tabular DL, and perform the original analysis on its influence on tabular MLPs (section 5).

3 TabM

In this section, we present TabM – a **Tab**ular model that makes **M**ultiple predictions.

3.1 Preliminaries

Notation. We consider classification and regression tasks on tabular data. x and y denote the features and a label, respectively, of one object from a given dataset. A machine learning model takes x as input and produces $\hat{y}$ as a prediction of y. $N \in \mathbb{N}$ and $d \in \mathbb{N}$ respectively denote the "depth" (e.g. the number of blocks) and "width" (e.g. the size of the latent representation) of a given neural network. $d_y \in \mathbb{N}$ is the output representation size (e.g. $d_y = 1$ for regression tasks, and d_y equals the number of classes for classification tasks). $\mathcal{L}$ is the loss function used for training a neural network.

Datasets. Our benchmark consists of 50 publicly available datasets used in prior work, including Grinsztajn et al. (2022); Gorishniy et al. (2024); Rubachev et al. (2024). The main properties of our benchmark are summarized in Table 1, and more details are provided in Appendix C.

Table 1: The overview of our benchmark. The "Split type" property is explained in the text.

#Datasets	Train Size				#Features				Task type		Split type	
	Min.	Q50	Mean	Max.	Min.	Q50	Mean	Max.	#Regr.	#Classif.	Random	Domain-aware
50	1787	10K	71K	723K	3	17	100	986	32	18	41	9

In particular, we pay extra attention to datasets with what we call "domain-aware" splits, including the eight datasets from Rubachev et al. (2024) and the Microsoft dataset (Qin & Liu, 2013). For these datasets, their original real world splits are available, for example, time-aware splits as in Rubachev et al. (2024). Such datasets were shown to be challenging for some methods, because they naturally exhibit a certain degree of distribution shift between training and test parts (Rubachev et al., 2024). The random splits of the remaining 41 datasets are inherited from prior work.

Experiment setup. We use the experiment setup from Gorishniy et al. (2024) and describe it in detail in subsection E.2. In a nutshell, for each model on each dataset, the pipeline is as follows. First, the model undergoes hyperparameter tuning on the *validation* set. Then, the tuned model is trained from scratch under multiple random seeds. The final score of the model on the dataset is defined as the *test* metric averaged over the random seeds.

Metrics. We use RMSE (the root mean square error) for regression tasks, and accuracy or ROC-AUC for classification tasks depending on the dataset source. Additional details are provided in subsection E.3.

Also, throughout the paper, we often use the relative performance of models w.r.t. MLP as the key metric. This metric gives a unified perspective on all tasks and allows reasoning about the scale and consistency of improvements w.r.t. to a simple baseline (MLP). Formally, on a given dataset, the metric is defined as $\left(\frac{\text{score}}{\text{baseline}} - 1\right) \cdot 100\%$, where "score" is the metric of a given model, and "baseline" is the metric of MLP. In this computation, for regression tasks, we convert the raw metrics from RMSE to R^2 to better align the scales of classification and regression metrics.

3.2 A QUICK INTRODUCTION TO BATCHENSEMBLE

BatchEnsemble (Wen et al., 2020) plays an important role in our story, so we quickly describe it in this section.

Let's consider a linear layer $\texttt{Linear}(x) = Wx + b$, where $x \in \mathbb{R}^d$, $W \in \mathbb{R}^{d \times d}$ and $b \in \mathbb{R}^d$ (we keep the input and output dimensions equal for simplicity, but they can be different). In a traditional deep ensemble, each ensemble member has an independent weight matrix W_i for this linear layer. By contrast, in BatchEnsemble, the weight matrix W_i of each member is obtained with the elementwise product of a shared full-rank matrix and a non-shared rank-one matrix: $W_i = W \odot r_i s_i^T$, where W is shared between all ensemble members, and $r_i, s_i \in \mathbb{R}^d$ are not shared. r_i and s_i are randomly initialized with ± 1 to ensure diversity of the k linear layers. This weight sharing strategy can be applied to one or more linear layers of the original neural network f. All other layers, including the remaining linear layers, are fully shared between all ensemble members.

The described parametrization allows to pack all ensemble members in *one* model that simultaneously applies all (now implicit) submodels in parallel, without explicitly instantiating the W_i matrices of individual members. This can be achieved by replacing one or more linear layers of the original neural network f with their BatchEnsemble versions (e.g. see the lower left part of Figure 1). Formally:

$$\text{Linear}_{\text{BE}}(X) = ((X \odot R)W) \odot S + B \tag{1}$$

where $X \in \mathbb{R}^{k \times d}$ represents k representations of the same input object (one per submodel), and $R, S, B \in \mathbb{R}^d$ represent the non-shared weights (r_i, s_i, b_i) of the submodels.

Terminology. In this paper, we call r_i, s_i, b_i, R, S and B *adapters*.

Overhead to the model size. Adding a new ensemble member means only adding one row to each of R, S and B of each $\text{Linear}_{\text{BE}}$, which gives $3d$ new parameters per layer. For typical values of d, this is a negligible overhead to the original layer size $d^2 + d$.

Overhead to the runtime. Thanks to the modern hardware and the parallel execution of the k forward passes, the runtime overhead of BatchEnsemble can be (significantly) lower than $\times k$ (Wen et al., 2020). In short, if the original workload underutilizes the hardware, there are more chances to pay less than $\times k$ overhead. This property is crucial for the efficiency of our model TabM.

3.3 TABM & TABM$_{\text{MINI}}$: BETTER MLPS WITH CUSTOMIZED BATCHENSEMBLE

In this section, we describe our models TabM and TabM$_{\text{mini}}$. In short, the models are based on a multilayer perceptron (MLP) and BatchEnsemble (Wen et al., 2020), with certain technical tweaks. In subsection A.1, we explain that (1) we choose specifically BatchEnsemble as the efficient ensembling method because of its performance and ease of use, while (2) using MLP as the backbone is crucial because of its excellent efficiency.

TabM$_{\text{naive}}$. We start by naively applying BatchEnsemble (Wen et al., 2020) to all linear layers of a vanilla MLP, with a minor difference that we use fully non-shared prediction heads. This gives us TabM$_{\text{naive}}$– the preliminary suboptimal version of TabM. In fact, the architecture (but not the initialization) of TabM$_{\text{naive}}$ is already the same as in TabM, so the lower left part of Figure 1 describes TabM$_{\text{naive}}$ as well. Throughout the paper, we always use $k = 32$, and then analyse this hyperparameter in subsection 5.3. The performance of TabM$_{\text{naive}}$ is reported in Figure 2, and it immediately shows the great potential of BatchEnsemble. For example, TabM$_{\text{naive}}$ is clearly superior to FT-Transformer (Gorishniy et al., 2021) – a popular attention-based baseline. This motivates further exploration.

TabM$_{\text{mini}}$. By construction, the just described TabM$_{\text{naive}}$ has $3N$ submodel-specific adapters (R, S and B in each of the linear layers, see Figure 1). A simple experiment reveals that, among the $3N$ adapters, exactly one of them plays a special role, namely, the first adapter (R) of the very first linear layer. To illustrate that, we first remove only this one adapter from TabM$_{\text{naive}}$ and keep the remaining $3N - 1$ adapters untouched, which gives us TabM$_{\text{bad}}$ with worse performance, as shown in Figure 2. Then, we do the opposite: we remove the $3N - 1$ adapters and keep the very first one, which essentially means having one adapter followed by MLP fully shared between all submodels. This gives us TabM$_{\text{mini}}$ – the minimal version of TabM, illustrated in Figure 1, where we call the described approach "MiniEnsemble". Perhaps, surprisingly, but Figure 2 shows that TabM$_{\text{mini}}$ performs better than TabM$_{\text{naive}}$, despite the $3N - 1$ pruned adapters.

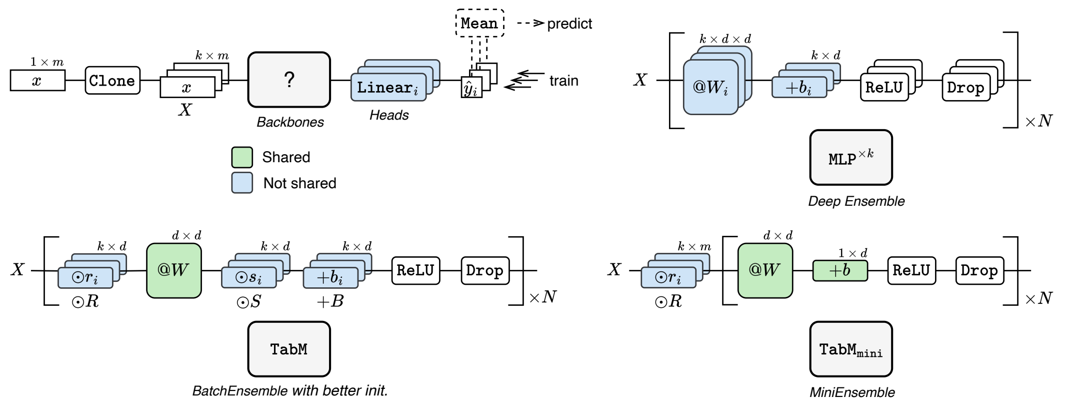

Figure 1: *(Upper left)* A template for implementing an ensemble of k MLPs. The remaining parts of the figure are three different parametrizations of the k MLP backbones. In all cases, each of the k MLP backbones independently processes its own copy of the input object. *(Upper right)* MLP$^{\times k}$ is a traditional deep ensemble of k fully independent MLPs. *(Lower left)* TabM is obtained by injecting three non-shared adapters R, S, B in each of the N linear layers of *one* MLP. *(Lower right)* TabM$_{\text{mini}}$ is obtained by keeping only the very first adapter R of TabM and removing the remaining $3N - 1$ adapters. Thus, TabM$_{\text{mini}}$ applies the same shared MLP to k object representations, with only two non-shared elements ensuring diversity of predictions: the randomly initialized multiplicative adapter R and the k prediction heads. *(Details)* Input transformations such as one-hot-encoding, feature embeddings (Gorishniy et al., 2022) and others are omitted for simplicity. In practice, they are applied (and the result is flattened) before the `Clone` module. `Drop` denotes dropout.

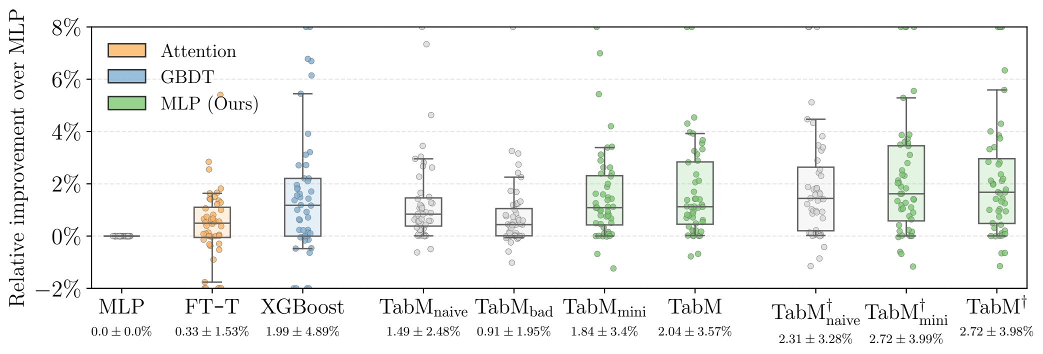

Figure 2: The performance on the 50 datasets from Table 1. For a given model, one dot on a jitter plot describes the performance score on one dataset. The numbers under the model names are the mean and standard deviations over the corresponding jitter plots. The box plots describe the percentiles of the jitter plot: boxes describe the 25th, 50th and 75th percentiles, and whiskers describe the 10th and 90th percentiles. Outliers are clipped. For each model, hyperparameters are tuned.

TabM. The just obtained results motivate the next simple experiment: we restore all $3N - 1$ pruned adapters for TabM$_{\text{mini}}$, but in an "incremental" way. Namely, we initialize all multiplicative adapters (R and S), except for the very first one, deterministically with 1 (instead of random ± 1 as recommended in the original BatchEnsemble). This gives us TabM, illustrated in Figure 1. As such, at initialization, TabM is equivalent to TabM$_{\text{mini}}$, but the deterministically initialized adapters are free to add more expressivity during training. Figure 2 shows that TabM is the best variation so far.

TabM$_{\text{mini}}$† & TabM†. Non-linear feature embeddings (Gorishniy et al., 2022) are known to boost the performance of many tabular models, especially of MLPs. We denote TabM$_{\text{mini}}$ and TabM with non-linear feature embeddings as TabM$_{\text{mini}}$† and TabM†, respectively. By default, we recommend using specifically the piecewise-linear embeddings (Gorishniy et al., 2022). In subsection A.5, we clarify additional implementation details, such as slightly different initialization. Figure 2 shows that, TabM$_{\text{mini}}$† is competitive with TabM†, so we will be using TabM$_{\text{mini}}$† by default for simplicity.

Intuition. To give additional intuition on TabM, we make the following observations:

- Setting $k = 1$ makes TabM identical to one plain MLP.
- Increasing k by one adds a negligible number of new parameters to TabM.
- For Transformer-like or Mixer-like (Tolstikhin et al., 2021) models, the shape of the latent representation is $m \times d$, where m is the number of features and d is the models' width; the m embeddings are repeatedly mixed with each other through attention or mixing layers. For TabM, the shape is only $k \times d$, and the k embeddings are never mixed.
- The story behind TabM$_{\text{mini}}$ shows that it is critical to create the k different object representations *before* the tabular features are mixed with each other in the first linear layer.

Hyperparameters. Compared to MLP, the only new hyperparameter of TabM is k – the number of implicit submodels. We heuristically set $k = 32$ and do not tune this value. We analyze the influence of k in subsection 5.3.

Limitations and practical considerations are commented in subsection A.6.

Next steps. The performance of TabM in Figure 2 renders it as a highly promising model. This motivates a full-fledged empirical comparison against prior tabular models (section 4) and detailed analysis of TabM's behaviour (section 5).

4 Evaluating tabular deep learning architectures

In this section, we perform a large scale empirical comparison of tabular models, including TabM introduced in section 3.

4.1 Baselines

In the main text, we use the most established and/or competitive methods, including: MLP (the classic multilayer perceptron), FT-Transformer denoted as "FT-T" (the attention-based model from Gorishniy et al. (2021)), SAINT (the attention- and retrieval- based model from Somepalli et al. (2021)), T2G-Former denoted as "T2G" (the attention-based model from Yan et al. (2023)), ExcelFormer denoted as "Excel" (the attention-based model from Chen et al. (2023a)), TabR (the retrieval-based model from Gorishniy et al. (2024)), ModernNCA denoted as "MNCA" (the retrieval-based model from Ye et al. (2024)) and three GBDT implementations: XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017) and CatBoost (Prokhorenkova et al., 2018). MLP†, TabR† and MNCA† denote the corresponding models with non-linear feature embeddings (Gorishniy et al., 2022). In fact, some other baselines, such as Excel (Chen et al., 2023a)), already use custom non-linear feature embeddings.

We present results for more baselines in Appendix F. In Appendix E, we provide implementation details for all methods.

4.2 Task performance

We evaluate all models following the protocol announced in subsection 3.1, and report the results in Figure 3 (see also the critical difference diagram in Figure 10). We make the following observations:

1. The performance ranks render TabM as the top-tier model along with GBDT models. Among other methods, only the most expensive variations of TabR and ModernNCA show some promise.
2. The middle and right parts of Figure 3 provide a fresh perspective on performance scores. TabM continues showing itself as a solid state-of-the-art model. At the same time, the task performance of some methods turns out to be no better or even worse than that of MLP on a non-negligible number of datasets, especially in the case of domain-aware splits (right). As such, given their complexity, it may be hard to position them as reliable go-to baselines.
3. One important characteristic of a model is the *weakest* part of its performance profile (e.g. the 10th or 25th percentiles in the middle plot), since it shows how reliable the model is on "inconvenient" datasets. From that perspective, MLP† seems to be a decent practical option between the plain MLP and TabM, especially given its simplicity and efficiency compared to retrieval-based alternatives, such as TabR and ModernNCA.

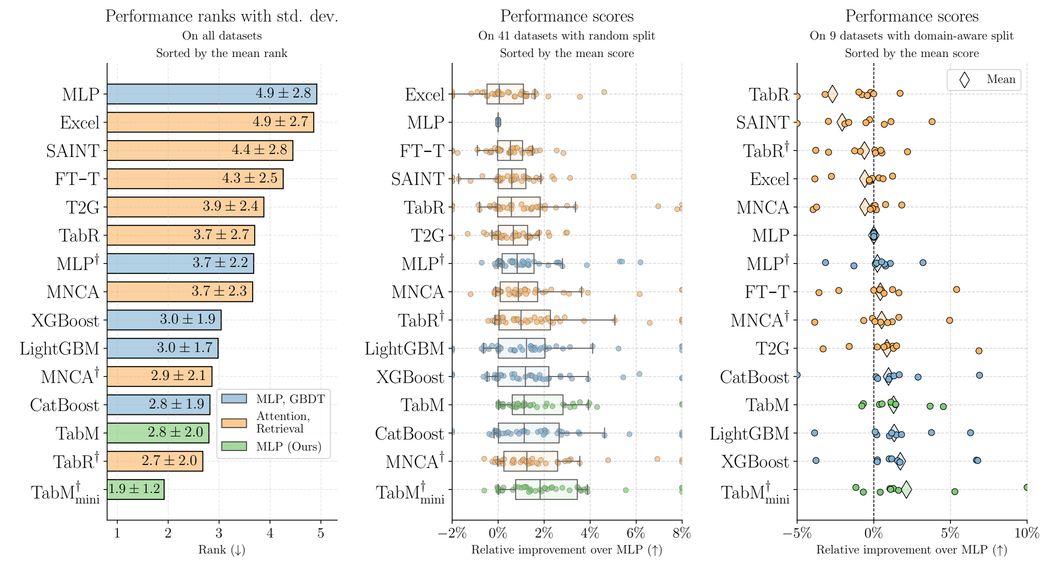

Figure 3: The task performance of tabular models on the datasets from Table 1. *(left)* The average performance ranks over all datasets summarize the head-to-head comparison between models on all datasets. *(middle, right)* The relative performance w.r.t. the plain multilayer perceptron (MLP) allows reasoning about the scale and consistency of performance improvements over this simple baseline. The meaning of the jitter plots and box plots is the same as in Figure 2. Outliers are clipped. The separation in random and domain-aware dataset splits is explained in subsection 3.1.

The main takeaway: TabM confidently demonstrates the best performance among tabular DL models and can serve as a reliable go-to DL baseline. The same cannot be said about attention- and retrieval- based models. MLP-like models remain simple and consistent tabular DL baselines.

4.3 Efficiency

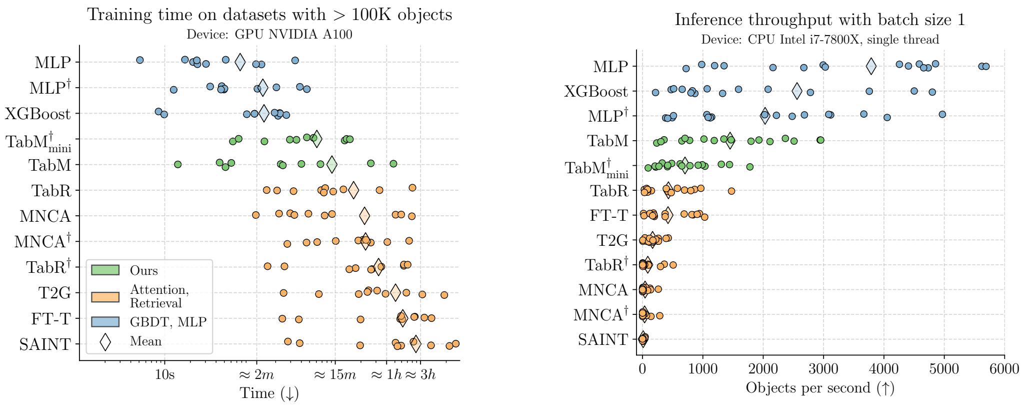

Figure 4: *(left)* Training times of the models from Figure 3 averaged over five random seeds. *(right)* Inference throughput of the models from Figure 3.

Now, we evaluate tabular models in terms of training and inference efficiency, which becomes a serious reality check for some of the methods. We benchmark exactly those hyperparameter configurations of models that are presented in Figure 3 (see subsection B.2 for the motivation).

Training time. We focus on training times on larger datasets, because on small datasets, all methods effectively become almost equally affordable regardless of the formal relative difference between methods. Nevertheless, in Figure 11, we provide measurements on small datasets as well. The left side of Figure 4 reveals that TabM offers practical training times. By contrast, the long training time of attention- and retrieval-based models becomes one more limitation of these methods.

Inference throughput. The right side of Figure 4 tells basically the same story as the left side.

Parameter count. Most tabular networks are overall compact. This, in particular, applies to TabM, since it adds a litte number of parameters compared to MLP. We report model sizes in Table 4.

The main takeaway. Simple MLPs and XGBoost are the fastest models, with TabM being the runner-up with still practical characteristics. The picture is significantly less positive for other methods, because their complexity actually converts to serious performance overhead.

4.4 Applying tabular models to large datasets

In this section, we quickly assess the applicability of several tabular DL to large datasets, without a strong focus on the task performance. Among the baselines, we use one attention-based model (FT-Transformer, (Gorishniy et al., 2021)), and one retrieval-based model (TabR (Gorishniy et al., 2024)). The results are reported in Table 2. As expected, attention- and retrieval-based models struggle on large datasets, yielding extremely long training times, or being simply inapplicable without additional effort. Implementation details are provided in subsection E.4.

5 Analysis

5.1 Practical properties of TabM

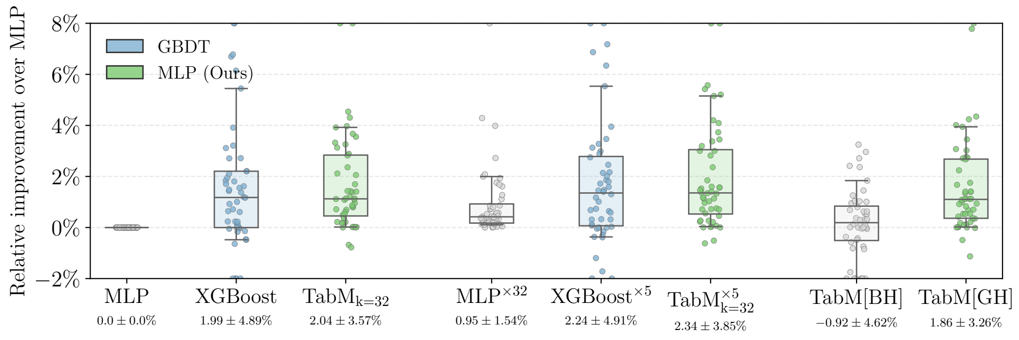

Figure 5: The performance on the 50 datasets from Table 1. The notation is the same as in Figure 2. The Model$^{\times K}$ denotes an ensemble of K models.

Here, we conduct experiments directly motivated by the architectural nature of TabM.

Ensembles. The first natural question to ask is how TabM compares to the traditional deep ensemble of MLPs. The results reported in Figure 5 are intriguing: $\mathrm{TabM}_{k=32}$ – one model imitating an ensemble of 32 MLPs – performs better than the full-fledged ensemble of 32 MLPs. We analyze this phenomenon in subsection A.2. The figure also shows that TabM, treated as one model, can itself benefit from traditional ensembling.

Diversity of the k predictions. The diversity-related properties of efficient ensembles are well-studied in original papers. For that reason, we only perform a minimal experiment to check if the k predictions of TabM are diverse. To that end, after the training, we choose the best prediction head (out of the k heads) on the validation set, and report its test performance in Figure 5 under the name TabM[BH] ("best head"). Interestingly, the best prediction head of TabM performs no better than the plain MLP. Thus, the k predictions of TabM must exhibit non-negligible diversity to compensate for poor individual performance.

Selecting submodels after training. The design of TabM allows selecting only a subset of sub-models after training according to any criteria, simply by removing extra prediction heads and the corresponding rows of the adapter matrices. To showcase these mechanics, using the validation set, we greedily construct the best subset of submodels of TabM after the training, and evaluate its test performance (see subsection E.5 for details). On average, this procedure results in 7.1 ± 5.6 submodels out of the initial $k = 32$, which can result in faster inference. The performance reported in Figure 5 under the name TabM[GH] ("greedy heads") illustrates the competitive performance of the "pruned" TabM.

5.2 Optimization properties of TabM

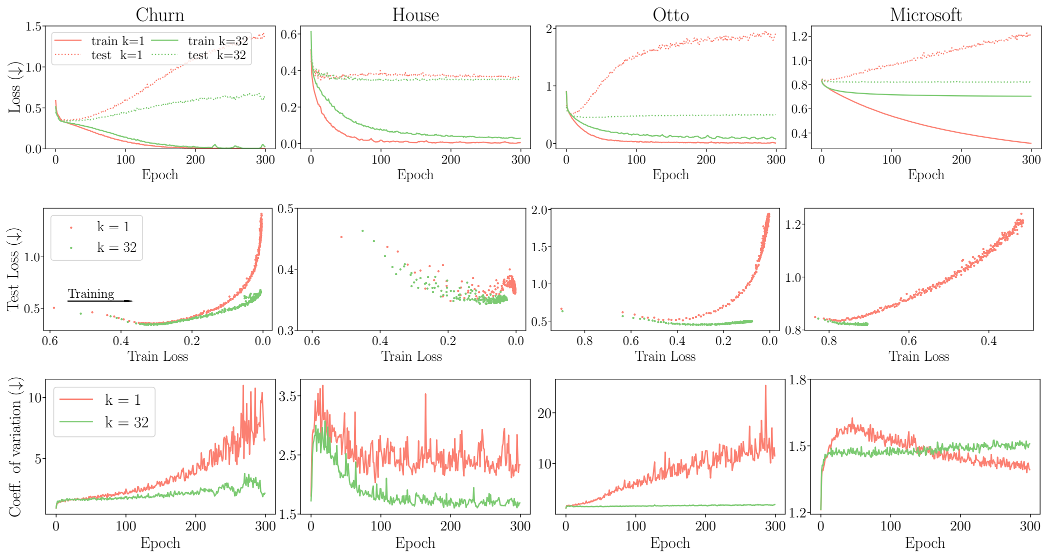

Figure 6: Training TabM$_{\text{mini}}$ with $k = 32$ and $k = 1$ (MLP) for 300 epochs with all regularizations turned off as explained in subsection 5.2. *(First row)* The training curves. *(Second row)* Same as the first row, but in the train-test coordinates: each dot corresponds to some epoch from the first row, and generally, the training progress happens from left to right. This allows reasoning about overfitting by comparing test loss values for a given train loss value. *(Third row)* The coefficient of variance, also recorded during the same run as the training curves.

Now, we aim to develop a better intuition on TabM's behaviour and its strong performance from the perspective of optimization. For simplicity, we analyze TabM$_{\text{mini}}$. Recall that TabM$_{\text{mini}}$ with $k = 1$ essentially equals one plain MLP of the same depth and width, and there is a large performance gap between MLP and TabM$_{\text{mini}}$ with $k = 32$. Then, the question is what exactly changes during the transition from $k = 1$ (MLP) to $k = 32$ (TabM$_{\text{mini}}$).

Experiment setup. Given the goal of this section and the posed question, we intentionally simplify the experiment setup to exclude side-effects coming from other places than from the transition between $k = 1$ and $k = 32$. We use the same depth 3 and width 512 for TabM$_{\text{mini}}$ and MLP. We turn off all regularizations (dropout, weight decay, gradient clipping), and, on each dataset, we tune the learning rate on the validation split separately for TabM$_{\text{mini}}$ and MLP. We consider four diverse datasets from our benchmarks (two classification and two regression tasks of different sizes). We turn off early stopping, train TabM$_{\text{mini}}$ and MLP for 300 epochs, record various optimization-related metrics and report them in Figure 6. Based on that, we make the following observations.

TabM$_{\text{mini}}$ exhibits reduced overfitting, as indicated by the second row of Figure 6.

TabM$_{\text{mini}}$ has lower variance of the stochastic gradients, as indicated by the third row of Figure 6 (lower coefficient of variation corresponds to relatively lower variance). In a nutshell, it means that the stochastic optimization process of TabM$_{\text{mini}}$ is more "stable", in the sense that randomly sampled training batches induce more accurate estimates of the full gradient (the gradient averaged over all training objects) for TabM$_{\text{mini}}$ than for MLP. See details in subsection E.6.

The latter result may help in understanding the intriguing superiority of TabM$_{k=32}$ over the ensemble of $k = 32$ MLPs observed in Figure 5. Recall that, for TabM$_{\text{mini}}$, the gradient induced by one training object is the average of k gradients coming from the k predictions. Perhaps, this "gradient ensembling" results in so good and stable gradients that their optimization power cannot be recovered with traditional ensemble of any size. At the same time, we are not aware of similar results for BatchEnsemble (Wen et al., 2020). Then, there is a chance that our result is specific for tabular MLPs, for example, because of poor optimization properties of simple MLPs and generally challenging optimization on real world tabular data.

5.3 How does the performance of TabM depend on k?

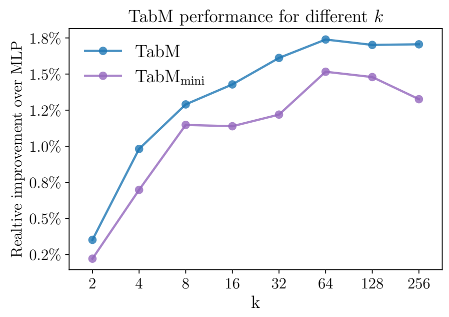

Figure 7: The average performance of TabM and TabM$_{\text{mini}}$ over 9 datasets from Table 5 with different values of the hyperparameter k.

Table 2: The performance and training time of multiple models on two large regression datasets from Rubachev et al. (2024): Weather (13M objects and 103 features) and Maps-Routing (6.5M objects and 986 features). Sorted by the average training time.

	Weather		Maps Routing	
	RMSE ↓	Time	RMSE ↓	Time
XGBoost	1.423	**10m**	0.1601	28m
MLP	1.484	30m	0.1592	**10m**
TabM$^{\dagger}_{\text{mini}}$	**1.410**	1.5h	**0.1583**	3h
FT-T	1.444	4.3h	0.1594	29h
TabR	OOM	N/A	OOM	N/A

Here, we explore the dependency of TabM on the number of implicit submodels k. We use TabM with the number of layers 3 and the width 512, tune the learning rate for each k, and report the performance in Figure 7. The figure indicates that $k = 32$ used throughout the paper was slightly suboptimal, though we consider it as a reasonable default value with a good balance between performance and efficiency. Also, too large values of k can be detrimental, as can be observed for TabM$_{\text{mini}}$. Perhaps, this happens because a larger number of submodels may require a larger model width to accommodate all submodels. At the same time, this effect is less pronounced for TabM. Perhaps, the larger number of submodel adapters in TabM compared to TabM$_{\text{mini}}$ allows fitting more submodels in one backbone. The implementation details are available in subsection E.7.

6 Conclusion & Future work

In this work, we have demonstrated that tabular multilayer perceptrons (MLPs) greatly benefit from BatchEnsemble – a parameter-efficient ensembling method. Based on this insight, we have designed TabM – a simple MLP-based model with state-of-the-art performance. In a large scale comparison with many tabular DL models, we have demonstrated that TabM is ready to serve as a new powerful and efficient tabular DL baseline. Finally, we have analyzed the key properties of TabM and provided intuition on its high performance.

One potential direction for future work is to use the multiple predictions of TabM for uncertainty estimation and out-of-distribution (OOD) detection on tabular data. This is motivated by the strong performance of (efficient) deep ensembles on those tasks in other domains (Lakshminarayanan et al., 2017). Another idea, directly inspired by our study (and in particular by subsection 5.2), is to bring the power of (parameter-)efficient ensembles to other (non-tabular) domains with optimization-related challenges and, ideally, lightweight models that will remain efficient even with a large number of implicit ensemble members.

Reproducibility statement. We provide all details about implementation and experiment setup in Appendix E. The proposed model is thoroughly described in section 3. Also, the source code is shared as supplementary material in a ZIP-archive. The `exp/` directory in the source code contains configuration files (`.toml` files) and report files (`.json` files) that, together, contain all information about experiments (hyperparameters, metrics, hardware, training time, etc.).

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, 2019. 20

Zeyuan Allen-Zhu and Yuanzhi Li. Towards understanding ensemble, knowledge distillation and self-distillation in deep learning. In *ICLR*, 2023. 2

Sercan O. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv*, 1908.07442v5, 2020. 2

Sarkhan Badirli, Xuanqing Liu, Zhengming Xing, Avradeep Bhowmik, Khoa Doan, and Sathiya S. Keerthi. Gradient boosting neural networks: Grownet. *arXiv*, 2002.07971v2, 2020. 2

Dara Bahri, Heinrich Jiang, Yi Tay, and Donald Metzler. Scarf: Self-supervised contrastive learning using random feature corruption. In *ICLR*, 2021. 2

Jintai Chen, Jiahuan Yan, Danny Ziyi Chen, and Jian Wu. Excelformer: A neural network surpassing gbdts on tabular data. *arXiv*, 2301.02819v1, 2023a. 1, 6, 20, 24, 25

Kuan-Yu Chen, Ping-Han Chiang, Hsin-Rung Chou, Ting-Wei Chen, and Tien-Hao Chang. Trompt: Towards a better deep neural network for tabular data. In *ICML*, 2023b. 1, 2, 20

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, 2016. 1, 2, 6, 20

Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. Deep ensembles: A loss landscape perspective. *arXiv*, 1912.02757v2, 2020. 2

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018. 3

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *NeurIPS*, 2021. 2, 4, 6, 8, 19, 20, 23, 26

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In *NeurIPS*, 2022. 2, 5, 6, 14, 20, 22, 23, 26

Yury Gorishniy, Ivan Rubachev, Nikolay Kartashev, Daniil Shlenskii, Akim Kotelnikov, and Artem Babenko. Tabr: Tabular deep learning meets nearest neighbors. In *ICLR*, 2024. 1, 2, 3, 6, 8, 18, 19, 20, 21, 23, 24, 26

Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *NeurIPS, the "Datasets and Benchmarks" track*, 2022. 3, 18, 19, 24

Marton Havasi, Rodolphe Jenatton, Stanislav Fort, Jeremiah Zhe Liu, Jasper Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran. Training independent subnetworks for robust prediction. In *ICLR*, 2021. 3, 14

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *ICLR*, 2023. 1, 2, 20

David Holzmüller, Léo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. *arXiv*, 2407.04491v1, 2024. 2

Alan Jeffares, Tennison Liu, Jonathan Crabbé, Fergus Imrie, and Mihaela van der Schaar. Tangos: Regularizing tabular neural networks through gradient orthogonalization and specialization. In *ICLR*, 2023a. 2

Alan Jeffares, Tennison Liu, Jonathan Crabbé, and Mihaela van der Schaar. Joint training of deep ensembles fails due to learner collusion. In *NeurIPS*, 2023b. 2

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. In *NeurIPS*, 2021. 2

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017. 1, 2, 6, 20

Myung Jun Kim, Léo Grinsztajn, and Gaël Varoquaux. Carte: pretraining and transfer for tabular learning. *arXiv*, abs/2402.16785v1, 2024. 17

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *NIPS*, 2017. 2, 20

Jannik Kossen, Neil Band, Clare Lyle, Aidan N. Gomez, Tom Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In *NeurIPS*, 2021. 2

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, 2017. 2, 10, 14

Olivier Laurent, Adrien Lafage, Enzo Tartaglione, Geoffrey Daniel, Jean-Marc Martinez, Andrei Bursuc, and Gianni Franchi. Packed ensembles for efficient uncertainty estimation. In *ICLR*, 2023. 15

Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra. Why M heads are better than one: Training a diverse ensemble of deep networks. *arXiv*, abs/1511.06314, 2015. 3, 14, 29

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 20

Sascha Marton, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Grande: Gradient-based decision tree ensembles for tabular data. In *ICLR*, 2024. 2

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 20

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *ICLR*, 2020. 2

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In *NeurIPS*, 2018. 1, 2, 6, 20

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *arXiv*, 1306.2597v1, 2013. 3

Ivan Rubachev, Artem Alekberov, Yury Gorishniy, and Artem Babenko. Revisiting pretraining objectives for tabular deep learning. *arXiv*, 2207.03208v1, 2022. 2

Ivan Rubachev, Nikolay Kartashev, Yury Gorishniy, and Artem Babenko. Tabred: A benchmark of tabular machine learning in-the-wild. *arXiv preprint arXiv:2406.19380*, 2024. 3, 10, 18, 19, 20, 21, 23, 25, 26

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. SAINT: improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv*, 2106.01342v1, 2021. 2, 6, 20, 25

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, 2019. 2, 20, 26

Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021. 6, 20

Mehmet Ozgur Turkoglu, Alexander Becker, Hüseyin Anil Gündüz, Mina Rezaei, Bernd Bischl, Rodrigo Caye Daudt, Stefano D'Aronco, Jan D. Wegner, and Konrad Schindler. Film-ensemble: Probabilistic deep learning via feature-wise linear modulation. In *NeurIPS 2022*, 2022. 3, 14

Ruoxi Wang, Rakesh Shivanna, Derek Z. Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. *arXiv*, 2008.13535v2, 2020. 2, 20

Yeming Wen, Dustin Tran, and Jimmy Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. In *ICLR*, 2020. 1, 2, 3, 4, 10, 14, 16

Jiahuan Yan, Jintai Chen, Yixuan Wu, Danny Z. Chen, and Jian Wu. T2G-FORMER: organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *AAAI*, 2023. 2, 6, 20, 24

Han-Jia Ye, Huai-Hong Yin, and De-Chuan Zhan. Modern neighborhood components analysis: A deep tabular baseline two decades later. *arXiv*, 2407.03257v1, 2024. 2, 6, 20, 23

Shaofeng Zhang, Meng Liu, and Junchi Yan. The diversified ensemble neural network. In *NeurIPS*, 2020. 3

A Additional analysis and discussion on TabM

A.1 Motivation

Why BatchEnsemble? Among relatively ease-to-use "efficient ensembling" methods, beyond BatchEnsemble, there are examples such as dropout ensembles (Lakshminarayanan et al., 2017), naive multi-head architectures, TreeNet (Lee et al., 2015). However, in the literature, they were consistently outperformed by more advanced methods, including BatchEnsemble (Wen et al., 2020), MIMO (Havasi et al., 2021), FiLM-Ensemble (Turkoglu et al., 2022).

Among advanced methods, in turn, BatchEnsemble is arguably one of the simplest and most flexible methods. For example, FiLM-Ensemble (Turkoglu et al., 2022) requires normalization layers to be presented in the original architecture, which is not always the case for tabular MLPs. MIMO (Havasi et al., 2021), in turn, imposes additional limitations compared to BatchEnsemble. *First*, it requires *concatenating* (not *stacking*, as with BatchEnsemble) all k input representations, which increases the input size of the first linear layer. With the relatively high number of submodels $k = 32$ used in our paper, this can be an issue on datasets with a large number of features, and especially when feature embeddings (Gorishniy et al., 2022) are used. For example, for $k = 32$, the number of features $m = 1000$ and the feature embedding size $l = 32$, the input size approaches one million resulting in an extremely large first linear layer of MLP. *Second*, with BatchEnsemble, it is easy to explicitly materialize, analyze and prune individual submodels. By contrast, in MIMO, all submodels are implicitly entangled within one MLP, and there is no easy way to access individual submodels.

Why MLPs? Indeed, despite the applicability of BatchEnsemble (Wen et al., 2020) to almost any architecture, we focus specifically on MLPs. The key reason is *efficiency*. *First,* to achieve high performance, throughout the paper, we use the relatively large number of submodels $k = 32$. However, the desired less-than-$\times k$ runtime overhead of BatchEnsemble typically happens only when the original model underutilizes the power of parallel computations of a given hardware. This will not be the case for attention-based models on datasets with a large number of features, as well as for retrieval-based models on datasets with a large number of objects. *Second,* as we show in subsection 4.3, attention- and retrieval-based models are already slow as-is. By contrast, MLPs are exceptionally efficient, to the extent that slowing them down even by an order of magnitude will still result in practical models.

A.2 Why TabM outperforms a full-fledged deep ensemble?

As shown in subsection 5.1, TabM is superior to MLP$^{\times k}$ (the traditional deep ensemble of k MLPs). In fact, the same is true for other variations of TabM, including TabM$_{\text{mini}}$ and even TabM$_{\text{naive}}$. This is intriguing, because TabM is simply an efficient parametrization of MLP$^{\times k}$. In this section, we analyze the potential reasons behind the difference in performance. In particular, we show that weight sharing between the implicit submodels is a major source of TabM's power. subsection A.3 provides more insights on the regularization power of the weight sharing.

For simplicity, we consider TabM$_{\text{mini}}$. To understand why TabM$_{\text{mini}}$ is superior to MLP$^{\times k}$, we start from MLP$^{\times k}$ and gradually modify it until we get TabM$_{\text{mini}}$, measuring the performance after each modification. We report the results in Table 3 and explain it in detail below.

Table 3: Comparing four implementations of the ensemble of k MLPs across all 50 datasets, with separate hyperparameter tuning for each implementation. See the text after the table for a detailed explanation of the models.

Model	Ensemble-aware training	Adapter	Weight-sharing	Ranks (↓)	Relative improvement over MLP (↑)	#Parameters overhead over MLP (↓)
MLP$^{\times 32}$	✗	✗	✗	2.02 ± 1.09	$0.96\% \pm 1.55\%$	$\times 32 \pm 0$
MLP$^{\times 32 \mid \text{EA}}$	✓	✗	✗	1.87 ± 0.72	$1.10\% \pm 1.84\%$	$\times 71 \pm 197$
TabM$_{\text{mini}}^{\text{k=1} \mid \times 32 \mid \text{EA}}$	✓	✓	✗	1.80 ± 0.72	$1.19\% \pm 2.08\%$	$\times 41 \pm 81$
TabM$_{\text{mini}}$	✓	✓	✓	1.26 ± 0.53	$1.87\% \pm 3.45\%$	$\times 4 \pm 5$

Table 3 tells the following story:

1. MLP$^{\times k}$. We start from the vanilla deep ensemble MLP$^{\times k}$ consisting of $k = 32$ MLPs trained *independently* under different random seeds. Hyperparameter tuning is performed to maximize *individual* performance of MLP, not of the final ensemble.
2. MLP$^{\times k | EA}$. The first thing we add is the **E**nsemble-**A**ware training. In fact, this is how TabM is trained out-of-the-box. Namely, now, the training (1) is *parallel*, i.e. each training step happens simultaneously for all k MLPs; (2) uses *ensemble-based early stopping*, i.e. the early stopping (and, as a consequence, the hyperparameter tuning) now optimizes the collective mean prediction of the MLPs; (3) is performed on the same batches for all models. Technically, the whole ensemble is efficiently implemented as one model following Packed-Ensembles (Laurent et al., 2023), which gives significantly better training efficiency compared to the sequential training in MLP$^{\times k}$. The results show that the new training protocol tend to improve the performance to some extent.
3. TabM$_{\text{mini}}^{k=1 | \times k | EA}$. Second, we remove the architectural difference and augment each MLP with a multiplicative adapter in the beginning. In other words, the base model changes from MLP to TabM$_{\text{mini}}^{k=1}$. The changes in performance seem to be less significant than from the previous step. Interestingly, the most affected metric is the model size.
4. TabM$_{\text{mini}}$. Finally, we share all weights except for the adapters and prediction heads, which gives us TabM$_{\text{mini}}$. The results indicate the high importance of weight sharing. Since the weight sharing limits the flexibility of an ensemble, one potential interpretations is that the weight sharing is a regularization. subsection A.3 provides more insights on the weight sharing.

A.3 Diversity of the k gradients

Recall that TabM receives k gradients per object on each training step, due to the k independently trained predictions. Due to the weight sharing between the submodels of TabM, these k gradients are mostly applied to the same weights (the exceptions are only the non-shared weights, i.e. the adapters and the predition heads). In this section, we show that these k gradients are "diverse" in a sense that, on average, they barely agree with each other on the optimal direction in the weight space. Intuitively, this can be viewed as a regularization, which is line with the story in subsection A.2.

We consider TabM$_{\text{mini}}$ in the simplified experiment setup as in subsection 5.2. The green line in Figure 8 shows that the mean pairwise cosine similarity between the k individual gradients remains close to zero during training. This may explain the higher training loss of TabM$_{\text{mini}}^{k=32}$ compared to TabM$_{\text{mini}}^{k=1}$ observed in the first row in Figure 6: perhaps, the weight sharing combined with the diverse gradients prevents TabM$_{\text{mini}}$ from (over)optimizing for the training task, and thus serves as a form of a regularization.

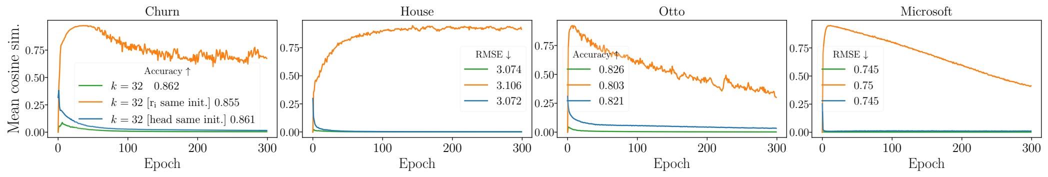

Figure 8: The mean pairwise cosine similarity between the k individual gradients of TabM$_{\text{mini}}^{k=32}$ with the default initialization (green) and two suboptimal initializations described in subsection A.3. Formally: $\frac{2}{n \cdot k(k-1)} \sum_{l,i,j(i<j)} \cos\left(g_i^l, g_j^l\right)$, where g_i^l is the gradient of the i-th submodel induced by the l-th of the $n = 1000$ training objects (the objects are selected randomly once before the training; the prediction heads are excluded when computing the cosine similarity). The legends report the test scores if early stopping was used.

In the same Figure 8, we also run an ablation study on the two sources of submodel diversity in TabM$_{\text{mini}}$: the random initializations in the adapter R and in the k prediction heads. When all rows of R (i.e. r_i in terms of Figure 1) receive the same initialization, while the k prediction heads are initialized completely randomly (the orange line), the submodel gradients are correlated, and the task performance is poor. By contrast, the issue is less pronounced when the k prediction heads receive the same initialization, and the initialization of R is completely random (the blue line), though it also can hurt the performance. Thus, the first adapter seems to be a more impactful source of gradient diversity. Overall, we see the gradient diversity as an experimental metric requiring more exploration.

A.4 The overhead of BatchEnsemble

This section aims to give some intuition on the actual efficiency overhead w.r.t. one model caused by applying BatchEnsemble (Wen et al., 2020) to this model. The section relies on the notation and story in subsection 3.2 and Figure 1.

Overhead to the model size. The whole overhead is concentrated in the $R, S, B \in \mathbb{R}^{k \times d}$ matrices in the modified linear layers, because all other layers are fully shared between the implicit ensemble members. As can be observed in Figure 1, there are only $3d$ additional trainable parameters per member compared to the original linear layer. This is negligible compared to $d^2 + d$ parameters per member in full-fledged deep ensembles. For example, for BatchEnsemble with $k = 32$ and $d = 512$, the overhead is less than $\times 2$ w.r.t. the size of the original linear layer.

Runtime overhead. One could rightfully ask how BatchEnsemble is more runtime-efficient than deep ensemble, if there are still k full-fledged forward passes happening under the hood during the one forward pass of f_{BE}. The answer is "because of the modern hardware": it turns out that, depending on the original architecture f and the batch size, the actual runtime overhead of f_{BE} w.r.t. f can be significantly lower than $\times k$. The more underutilized a GPU, the less the overhead. For example, here are some numbers obtained for the overhead of $\mathrm{Linear}_{\mathrm{BE}}$ compared to `Linear` on one NVIDIA A100:

- For $k = 32$, $d = 256$ and batch size 256, the overhead to the inference throughput is $\times 2.6$.
- For $k = 32$, $d = 512$ and batch size 512, the overhead to the inference throughput is $\times 6.8$.
- For $k = 32$, $d = 1024$ and batch size 1, the overhead to the inference throughput is $\times 1.3$.

A.5 TabM with feature embeddings

Here, we provide additional implementation details for $\mathrm{TabM_{mini}}^{\dagger}$ and $\mathrm{TabM}^{\dagger}$ introduced in subsection 3.3.

We note that feature embeddings are applied, and the result is flattened, before the `Repeat` module in terms of Figure 1. In other words, there are no changes in usage compared to plain MLPs. For $\mathrm{TabM_{mini}}^{\dagger}$ and $\mathrm{TabM}^{\dagger}$, we initialize the first multiplicative adapter R of the first linear layer from the standard normal distribution $\mathcal{N}(0, 1)$.

The remaining details are insignificant and are best understood from the source code.

A.6 Limitations and practical considerations

TabM does introduce any new limitations compared to BatchEnsemble (Wen et al., 2020). Nevertheless, we note the following:

- Arguably the key limitation is that BatchEnsemble-like techniques are not "local", but instead affect the whole model starting from the first modified layer. Namely, when the computation flow hits the first modified layer, the k prediction branches are created, and the rest of the network will have to make k times more computations. This can be easily affordable for small models, but may be less affordable for heavy base models.
- For ensemble-like models, such as TabM, the notion of "the final object embedding" changes: now, it is not a single vector, but a set of k vectors. This can be important for scenarios when TabM is used for solving more than one task, in particular, when it is pretrained as a generic feature extractor and then reused for other tasks. The main practical guideline is that the k prediction branches should *never* interact with each other (e.g. through attention, pooling, etc.) and should always be trained separately.

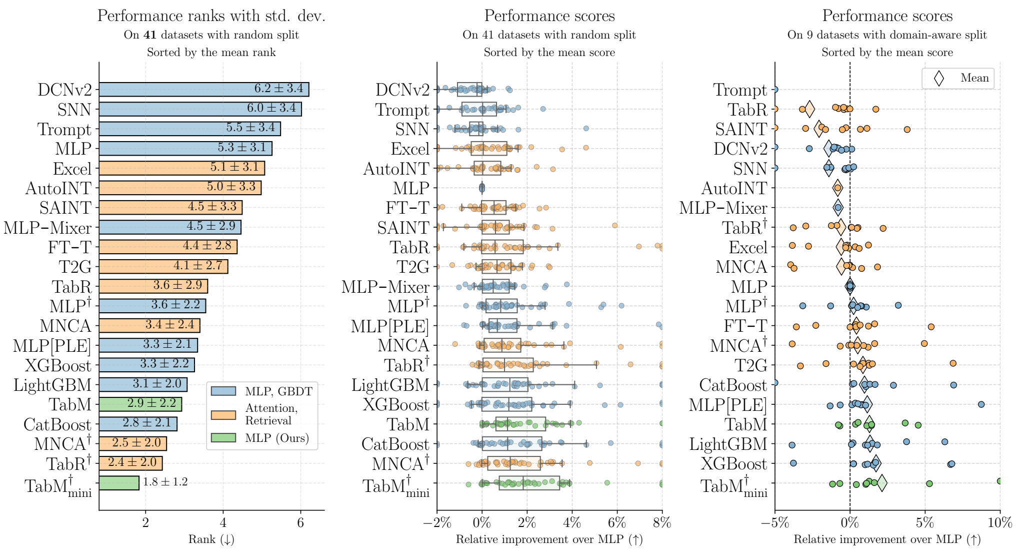

Figure 9: An extended comparison of tabular models as in Figure 3. Note that here ranks (*left*) are computed on 41 datasets (not on 50) since some models have not been evaluated on 8 datasets from Table 6.

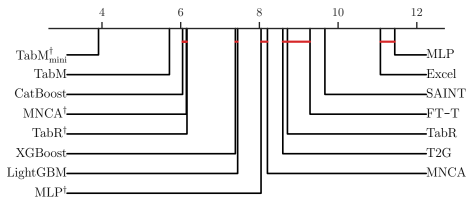

Figure 10: Critical difference diagram (CDD). The computation method is taken from the Kim et al. (2024).

B Extended results

B.1 Evaluation

B.2 Efficiency

This section completes subsection 4.3.

Motivation for the benchmark setup. Benchmarking efficiency properly is hard, and comparing models under all possible kinds of budget (task performance, the number of parameters, training time, etc.) on all possible hardware (GPU, CPU, etc.) with all possible batch sizes is rather infeasible. As such, we set a narrow goal of *providing a high-level intuition on the efficiency under trasparent setting*. Thus, benchmarking the transparently obtained tuned hyperparameter configurations works well for our goal. Yet, this choice also has a limitation: the hyperparameter tuning process is not aware of efficiency budget, so it can prefer much heavier confugirations even if they lead to tiny performance improvements, which will negatively affect efficiency without a good reason.

Motivation for the two setups for measuring inference throughput.

- The setup in the right side of Figure 4 simulates the online per-object predictions.
- The setup in the right side of Figure 11 simulates the offline batched computations.

Additional results.

Figure 11 completes Figure 4.

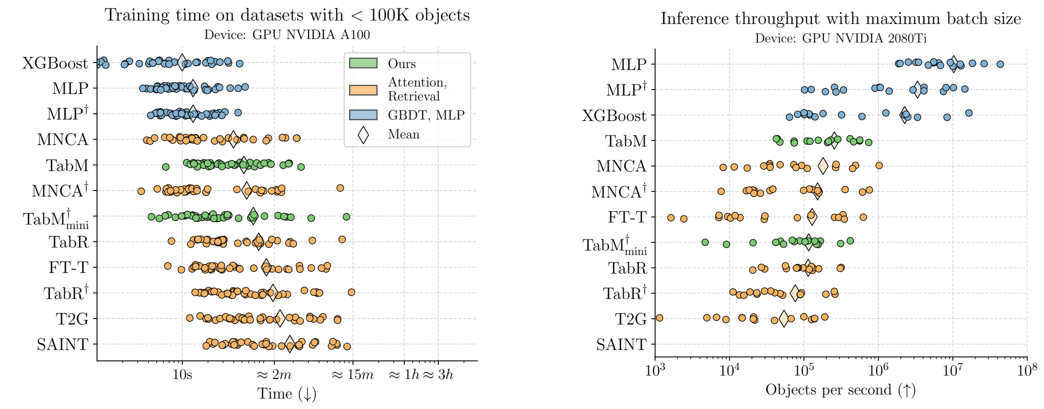

Figure 11: (*Left*) Training time on datasets with less than 100K objects. (*Right*) Inference throughput on GPU with maximum batch size.

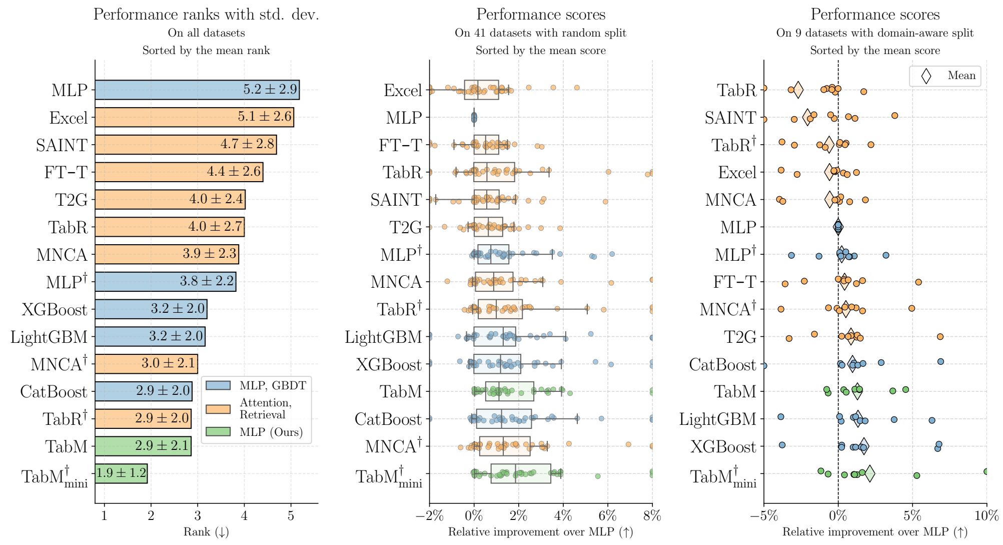

Figure 12: Same as Figure 3, but ROC-AUC is used as the metric for all classification datasets.

Table 4: Average number of parameters with std. dev. for 7 different tuned models across all 50 datasets.

TabM	MLP	FT-T	T2G	TabR	ModernNCA	SAINT
$1.4M \pm 1.3M$	$1.0M \pm 1.0M$	$1.2M \pm 1.2M$	$2.1M \pm 1.6M$	$858K \pm 1.4M$	$1.0M \pm 1.1M$	$175.4M \pm 565.4M$

C Datasets

We use some datasets from Gorishniy et al. (2024) with minor differences commented below and 8 datasets from (Rubachev et al., 2024). In total, we use 50 datasets: 32 datasets from Grinsztajn et al. (2022) (also used in Gorishniy et al. (2024)), 10 dataset from Gorishniy et al. (2024), 8 datasets from Rubachev et al. (2024). There are 9 dataset with domain-aware split in total: 8 from Rubachev et al. (2024) and Microsoft from Table 5.

Originally, (Gorishniy et al., 2024) uses 47 distinct datasets across all experiments (including datasets from Table 5). We use 42 datasets out of these 47 datasets. We did not include the following 5 datasets:

Table 5: Properties of those datasets (10 datasets out of 50 datasets used in our study) that are not part of Grinsztajn et al. (2022) or Rubachev et al. (2024) benchmarks. "# Num", "# Bin", and "# Cat" denote the number of numerical, binary, and categorical features, respectively. The table is taken from (Gorishniy et al., 2024).

Name	# Train	# Validation	# Test	# Num	# Bin	# Cat	Task type	Batch size
Churn Modelling	6400	1600	2000	10	3	1	Binclass	128
California Housing	13209	3303	4128	8	0	0	Regression	256
House 16H	14581	3646	4557	16	0	0	Regression	256
Adult	26048	6513	16281	6	1	8	Binclass	256
Diamond	34521	8631	10788	6	0	3	Regression	512
Otto Group Products	39601	9901	12376	93	0	0	Multiclass	512
Higgs Small	62751	15688	19610	28	0	0	Binclass	512
Black Friday	106764	26692	33365	4	1	4	Regression	512
Covertype	371847	92962	116203	54	44	0	Multiclass	1024
Microsoft	723412	235259	241521	131	5	0	Regression	1024

Table 6: Properties of 8 datasets from TabReD (Rubachev et al., 2024) benchmark. "# Num", "# Bin", and "# Cat" denote the number of numerical, binary, and categorical features, respectively.

Name	# Train	# Validation	# Test	# Num	# Bin	# Cat	Task type	Batch size
Sberbank Housing	18847	4827	4647	365	17	10	Regression	256
Ecom Offers	109341	24261	26455	113	6	0	Binclass	1024
Maps Routing	160019	59975	59951	984	0	2	Regression	1024
Homesite Insurance	224320	20138	16295	253	23	23	Binclass	1024
Cooking Time	227087	51251	41648	186	3	3	Regression	1024
Homecredit Default	267645	58018	56001	612	2	82	Binclass	1024
Delivery ETA	279415	34174	36927	221	1	1	Regression	1024
Weather	106764	42359	40840	100	3	0	Regression	1024

- electricity from (Grinsztajn et al., 2022). The dataset is a time series forecasting problem transformed to a regression task (each window of a certain size is treated as a tabular object with the label being the next measurement after the window). However, the windows are shuffled and split randomly into train, validation and test, which means that algorithms are partly trained on future data, which is an unrealistic setup.
- yprop_4_1 from (Grinsztajn et al., 2022). This dataset is not informative for our purposes: in our experiments, all algorithms show the same performance regardless of the type of the algorithm and hyperparameter tuning, which is in line with the results from Gorishniy et al. (2024) (see their Appendix.E in Gorishniy et al. (2024)).
- rl from (Grinsztajn et al., 2022). On this dataset, we observed abnormal results, and since this is an anonymous dataset, we removed it to avoid confusion.
- compass from (Grinsztajn et al., 2022). There is a leak in this dataset. The rows of the dataset are the results of the different assessments, plus some features related to the person who took the assessments. But it is possible that a person completed more than one assessment (on the same day) which results in more than one row for that person. Due to the random split one person can be included in both train and test splits. The correct way would be to split data by person id. Importantly, this does not affect any conclusions.
- The weather forecasting dataset. This dataset was split without taking time into account, which means that algorithms are partly trained on future data, which is an unrealistic setup for weather prediction. There is a Weather dataset from Rubachev et al. (2024) that we use.

D BASELINES

D.1 MAIN BASELINES

In section 4, we used the following models as baselines:

- MLP (Gorishniy et al., 2021)

- FT-Transformer (Gorishniy et al., 2021)
- Excelformer (Chen et al., 2023a)
- T2G-Former (Yan et al., 2023)
- SAINT (Somepalli et al., 2021)
- MLP† (with periodic embeddings from Gorishniy et al. (2022))
- CatBoost (Prokhorenkova et al., 2018)
- XGBoost (Chen & Guestrin, 2016)
- LightGBM (Ke et al., 2017)
- TabR (Gorishniy et al., 2024)
- TabR† (with numerical embeddings Gorishniy et al. (2024))
- ModernNCA (Ye et al., 2024)
- ModernNCA† (Ye et al., 2024) (with numerical embeddings Gorishniy et al. (2024))

D.2 Additional baselines

In fact, we evaluated even more baselines and their results are available in Appendix F.

- SNN (Klambauer et al., 2017)
- DCNv2 (Wang et al., 2020)
- MLP[PLE] (with piecewise-linear embeddings from Gorishniy et al. (2022))
- AutoInt (Song et al., 2019)
- TabPFN (Hollmann et al., 2023) (not applicable to regressions)
- Trompt (Chen et al., 2023b) (our reimplementation, since there is no official implementation)
- MLP-Mixer (our heuristic adaptation of Tolstikhin et al. (2021) for tabular data)

For SNN and DCNv2, our observations were in line with prior studies (Gorishniy et al., 2021), and they did not affect our story. For Trompt, we did not manage to get competitive results. TabPFN specializes on small datasets, and indeed, it was not competitive on our benchmark.

E Implementation details

E.1 Hardware

Most of the experiments were conducted on a single NVIDIA A100 GPU. In rare exceptions, we used a machine with a single NVIDIA 2080 Ti GPU and Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz.

E.2 Experiment setup

We mostly follow the experiment setup from Gorishniy et al. (2024). As such, some of the text below is copied from (Gorishniy et al., 2024).

Data preprocessing. For each dataset, for all DL-based solutions, the same preprocessing was used for fair comparison. For numerical features, by default, we used a slightly modified version of the quantile normalization from the Scikit-learn package (Pedregosa et al., 2011) (see the source code), with rare exceptions when it turned out to be detrimental (for such datasets, we used the standard normalization or no normalization). For categorical features, we used one-hot encoding. Binary features (i.e. the ones that take only two distinct values) are mapped to $\{0, 1\}$ without any further preprocessing. We completely follow Rubachev et al. (2024) on Table 6 datasets.

Training neural networks. For DL-based algorithms, we minimize cross-entropy for classification problems and mean squared error for regression problems. We use the AdamW optimizer (Loshchilov & Hutter, 2019). We do not apply learning rate schedules. We do not use data augmentations. We apply global gradient clipping to 1.0. For each dataset, we used a predefined dataset-specific batch size. We continue training until there are `patience` consecutive epochs without improvements on the validation set; we set `patience` $= 16$ for the DL models.

Hyperparameter tuning. In most cases, hyperparameter tuning is performed with the TPE sampler (typically, 50-100 iterations) from the Optuna package (Akiba et al., 2019). Hyperparameter tuning spaces for most models are provided in individual sections below (example for TabM: subsection E.8). We follow Rubachev et al. (2024) and use 25 iterations on some datasets from Table 6.

Evaluation. On a given dataset, for a given model, the tuned hyperparameters are evaluated under multiple (in most cases, 15) random seeds. The mean test metric and its standard deviation over these random seeds are then used to compare algorithms as described in subsection E.3.

E.3 Metrics

We use Root Mean Squared Error for regression tasks, ROC-AUC for classification datasets from Table 6 (following Rubachev et al. (2024)), and accuracy for the rest of datasets (following Gorishniy et al. (2024)). We also tried computing ROC-AUC for all classification datasets, but did not observe any significant changes (see Figure 12), so we stuck to prior work. By default, the mean score and its standard deviation are obtained by training a given model with tuned hyperparameters from scratch on a given dataset under 15 different random seeds. We use the test splits to compare the performance of different models.

How we compute relative improvements. We compute performance improvement of `model` relative to `baseline` as follows: $(\texttt{mean\_model\_score}/\texttt{mean\_baseline\_score} - 1) \cdot 100\%$. `mean_score` is R2-score for regressions and Accuracy (ROC-AUC for Table 6) for classifications. Note, that here we use R2-score for regression tasks, so the regression and classification metrics become more comparable.

How we compute ranks. Our method of computing ranks used in Figure 3 does not count small improvements as wins, hence the reduced range of ranks compared to other studies. Intuitively, our ranks can be considered as “tiers”.

Recall that, on a given dataset, a given model A has its mean score A_{mean} and the standard deviation if its score A_{std} (obtained after the evaluation under multiple random seeds on the dataset). Assuming the higher score the better, we define that the model A is better than the model B if: $\text{A}_{\text{mean}} - \text{A}_{\text{std}} > \text{B}_{\text{mean}}$. In other words, a model is considered better if it has a better mean score and the margin is larger than the standard deviation.

On a given dataset, when there are many models, we sort them in descending score order. Starting from the best model (with rank equal to 1) we iterate over models and assign first rank to all models that are no worse than the best model according to the above rule. The first model in descending order that is worse than the best model is assigned rank 2 and becomes a new reference model. We continue the process until all models are ranked. Ranks are computed independently for each dataset.

E.4 Implementation details of subsection 4.4

In this section, we used a full train split of Weather and Maps Routing datasets from Rubachev et al. (2024) while validation and test splits remained the same size as in Table 6. So, the results are comparable with the results from Appendix F. We took tuned configurations of the models from section 4 and trained these models on large datasets (1 seed was used for FT-T and 3 seeds for the other models). DL models were trained using Automatic Mixed Precision for the speed of experiments.

E.5 Implementation details of subsection 5.1

TabM[GH]. Here, we clarify implementation details for TabM[GH] announced in subsection 5.1. TabM[GH] is obtained from a trained TabM by greedily selecting submodels from TabM starting from the best one and stopping when two conditions are simultaneously true for the first time: (1) adding any new submodel does not improve the validation metrics; (2) the current validation metric is already better than that of the model with all heads. To clarify, during the greedy selection, the i-th submodel is considered to be better than the j-th submodel if adding the i-th submodel to the aggregated prediction leads to better validation metrics (i.e. it is *not* the same as adding the submodel in the order of their individual validation metrics).

E.6 Implementation details of subsection 5.2

Coefficient of variation. Let p be the number of model parameters, n is the size of train set and $g_i \in \mathbb{R}^p$ is the model gradient induced by training sampling $i \sim \text{U}[1, n]$.

The coefficient of variation is calculated as follows:

$$\texttt{Coeff} := \frac{\sqrt{\operatorname{Tr}(\operatorname{Cov}(g_i))}}{\mathbb{E}_{i\sim \mathrm{U}[1,n]}[\|g_i\|]} = \frac{\sqrt{\mathbb{E}_{i\sim \mathrm{U}[1,n]}[\|g_i\|^2] - \|\mathbb{E}_{i\sim \mathrm{U}[1,n]} g_i\|^2}}{\mathbb{E}_{i\sim \mathrm{U}[1,n]}[\|g_i\|]}$$

Essentially, we mostly interested in the standard deviation of a gradient (numerator), but the division by the mean norm makes results for different architectures ($k = 32$ and $k = 1$) comparable. On each epoch, we used the same 5000 objects from train set in order to estimate the coefficient (these 5000 objects were randomly selected before the training). Also, a script that calculates coefficient of variation during training is included in the source code (`bin/model_analysis.py`).

E.7 Implementation details of subsection 5.3

Figure 7 shows mean percentage improvement (see subsection E.3) over MLP across 9 datasets from Table 5 (without Covertype). We have used a TabM$_{\text{mini}}$ with 3 hidden layers of the width $d = 512$, the dropout rate 0.1 and tuned learning rate for different k. The score on each dataset is averaged over 5 seeds.

E.8 TabM

Here we provide hyperparameter tuning spaces for TabM and TabM$_{\text{mini}}$.

Table 7: The hyperparameter tuning space for TabM. Here, (B) = {Covertype, Microsoft, Table 6} and (A) contains all other datasets.

Parameter	Distribution or Value
k	32
# layers	$\mathrm{UniformInt}[1, 5]$
Width (hidden size)	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[1e\text{-}4, 5e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50

Table 8: The hyperparameter tuning space for TabM$_{\text{mini}}$ that uses PiecewiseLinearEncoding embeddings from Gorishniy et al. (2022). Here, (B) = {Covertype, Microsoft, Table 6} and (A) contains all other datasets.

Parameter	Distribution or Value
k	32
# layers	$\mathrm{UniformInt}[1, 4]$
Width (hidden size)	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.5]\}$
# PLE bins	$\mathrm{UniformInt}[8, 32]$
Learning rate	$\mathrm{LogUniform}[5e\text{-}5, 3e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50

E.9 TabR

Since we completely follow training and evaluation protocols from Gorishniy et al. (2024) and TabR was proposed in Gorishniy et al. (2024), we simply reuse results for TabR. More details can be found in Appendix.D from Gorishniy et al. (2024). For TabR† on Table 6 we have used 25 tuning iterations and the same tuning space as for TabR from Rubachev et al. (2024), we also followed Gorishniy et al. (2024) and used periodic embeddings on small datasets (Sberbank Housing and Ecom Offers) and Linear embeddings for the other datasets.

E.10 FT-Transformer

We used the implementation from the "`rtdl_revisiting_models`" Python package (version 0.0.2). The results on datasets from Table 6 were copied from Rubachev et al. (2024).

Table 9: The hyperparameter tuning space for FT-Transformer Gorishniy et al. (2021). Here, (B) = {Covertype, Microsoft} and (A) contains all other datasets (except Table 6).

Parameter	Distribution or Value
# blocks	$\mathrm{UniformInt}[1, 4]$
d_{token}	$\mathrm{UniformInt}[16, 384]$
Attention dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
FFN hidden dimension expansion rate	$\mathrm{Uniform}[2/3, 8/3]$
FFN dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
Residual dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.2]\}$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50

E.11 ModernNCA

We adapted an official implementation of Ye et al. (2024). We have used periodic embeddings Gorishniy et al. (2022) for ModernNCA† and no embeddings for ModernNCA.

Table 10: The hyperparameter tuning space for ModernNCA. Here, (C) = {Table 6}, (B) = {Covertype, Microsoft} and (A) contains all other datasets.

Parameter	Distribution
# blocks	$\mathrm{UniformInt}[0, 2]$
d_{block}	$\mathrm{UniformInt}[64, 1024]$
dim	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
Sample rate	$\mathrm{Uniform}[0.05, 0.6]$
Learning rate	$\mathrm{LogUniform}[1e\text{-}5, 1e\text{-}1]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}6, 1e\text{-}3]\}$
# Tuning iterations	(A) 100 (B, C) 50

Table 11: The hyperparameter tuning space for ModernNCA†. Here, (C) = {Table 6}, (B) = {Covertype, Microsoft} and (A) contains all other datasets.

Parameter	Distribution
# blocks	$\mathrm{UniformInt}[0, 2]$
d_{block}	$\mathrm{UniformInt}[64, 1024]$
dim	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
Sample rate	$\mathrm{Uniform}[0.05, 0.6]$
Learning rate	$\mathrm{LogUniform}[1e\text{-}5, 1e\text{-}1]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}6, 1e\text{-}3]\}$
n_frequencies	$\mathrm{UniformInt}[16, 96]$
d_embedding	$\mathrm{UniformInt}[16, 32]$
frequency_init_scale	$\mathrm{LogUniform}[0.01, 10]$
# Tuning iterations	(A) 100 (B, C) 50

E.12 T2G-Former

We adapted the implementation and hyperparameters of Yan et al. (2023) from the official repository[1].

Table 12: The hyperparameter tuning space for T2G-Former Yan et al. (2023). Here, (C) = {all from Table 6}, (B) = {Covertype, Microsoft} and (A) contains all other datasets. Also, we used 50 tuning iterations for some datasets from Grinsztajn et al. (2022).

Parameter	Distribution or Value
# blocks	(A) $\mathrm{UniformInt}[3, 4]$ (B, C) $\mathrm{UniformInt}[1, 3]$
d_{token}	$\mathrm{UniformInt}[64, 512]$
Attention dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
FFN hidden dimension expansion rate	(A, B) $\mathrm{Uniform}[2/3, 8/3]$ (C) $4/3$
FFN dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
Residual dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.2]\}$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Col. Learning rate	$\mathrm{LogUniform}[5e\text{-}3, 5e\text{-}2]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}6, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50 (C) 25

E.13 SAINT

We completely adapted hyperparameters and protocol from Gorishniy et al. (2024) to evaluate SAINT on Grinsztajn et al. (2022) benchmark. Results on datasets from Table 5 were directly taken from Gorishniy et al. (2024). Additional details can be found in Appendix.D from Gorishniy et al. (2024). We have used a default configuration on big datasets due to very high cost of tuning (see Table 13).

E.14 ExcelFormer

We adapted the implementation and hyperparameters of Chen et al. (2023a) from the official repository[2]. Importantly, we did not use MixUp technique from the paper in our expirements.

[1]https://github.com/jyansir/t2g-former
[2]https://github.com/WhatAShot/ExcelFormer

Table 13: The default hyperparameters for SAINT (Somepalli et al., 2021) on datasets from Rubachev et al. (2024).

Parameter	Value
depth	2
d_{token}	32
n_{heads}	4
d_{head}	8
Attention dropout rate	0.1
FFN hidden dimension expansion rate	1
FFN dropout rate	0.8
Learning rate	$1e$-4
Weight decay	$1e$-2

Table 14: The hyperparameter tuning space for Excelformer Chen et al. (2023a). Here, (D) = {Homecredit, Maps Routing}, (C) = {Table 6 w/o (D)}, (B) = {Covertype, Microsoft} and (A) contains all other datasets.

Parameter	Distribution or Value
# blocks	(A, B) $\mathrm{UniformInt}[2, 5]$ (C) $\mathrm{UniformInt}[2, 4]$ (D) $\mathrm{UniformInt}[1, 3]$
d_{token}	(A, B) $\{32, 64, 128, 256\}$ (C) $\{16, 32, 64\}$ (D) $\{4, 8, 16, 32\}$
n_{heads}	(A,B) $\{4, 8, 16, 32\}$ (C) $\{4, 8, 16\}$ (D) 4
Attention dropout rate	0.3
FFN dropout rate	0.0
Residual dropout rate	$\mathrm{Uniform}[0.0, 0.5]$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50 (C, D) 25

E.15 MLP

We used the implementation from the "`rtdl_revisiting_models`" Python package (version 0.0.2) and "`rtdl_num_embeddings`" Python package (version 0.0.10).

Table 15: The hyperparameter tuning space for MLP.

Parameter	Distribution
# layers	$\mathrm{UniformInt}[1, 6]$
Width (hidden size)	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	100

Table 16: The hyperparameter tuning space for MLP† that uses periodic embeddings from Gorishniy et al. (2022) (more precisely, the lite version of the PLR embeddings from the "`rtdl_num_embeddings`" package).

Parameter	Distribution
# layers	$\mathrm{UniformInt}[1, 6]$
Width (hidden size)	$\mathrm{UniformInt}[64, 1024]$
Dropout rate	$\{0.0, \mathrm{Uniform}[0.0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
n_frequencies	$\mathrm{UniformInt}[16, 96]$
d_embedding	$\mathrm{UniformInt}[16, 64]$
frequency_init_scale	$\mathrm{LogUniform}[0.01, 100]$
# Tuning iterations	100

E.16 CatBoost, XGBoost and LightGBM

We found the hyperparameter tuning protocol in Gorishniy et al. (2024) to give strong and representative results for GBDT models. Since our setup is directly taken from Gorishniy et al. (2024), we simply reused their results for GBDTs. Importantly, in a series of preliminary experiments, we confirmed that those results are reproducible in our instance of their setup. The details can be found in Appendix.D from Gorishniy et al. (2024). Results on datasets from Table 6 were copied from the paper (Rubachev et al., 2024).

E.17 AutoInt

We used an implementation from Gorishniy et al. (2021) which is an adapted official implementation[3].

Table 17: The hyperparameter tuning space for AutoInt (Song et al., 2019). Here, (B) = {Covertype, Microsoft} and (A) contains all other datasets.

Parameter	Distribution
# blocks	$\mathrm{UniformInt}[1, 6]$
d_{token}	$\mathrm{UniformInt}[8, 64]$
n_{heads}	2
Attention dropout rate	$\{0, \mathrm{Uniform}[0.0, 0.5]\}$
Embedding dropout rate	$\{0, \mathrm{Uniform}[0.0, 0.5]\}$
Learning rate	$\mathrm{LogUniform}[3e\text{-}5, 1e\text{-}3]$
Weight decay	$\{0, \mathrm{LogUniform}[1e\text{-}4, 1e\text{-}1]\}$
# Tuning iterations	(A) 100 (B) 50

E.17.1 TabPFN

Since TabPFN accepts only less than 10K training samples we use different subsamples of size 10K for different random seeds. Also, TabPFN is not applicable to regressions and datasets with more than 100 features.

[3]https://github.com/shichence/AutoInt

F Per-dataset results with standard deviations

Table 18: Extended results for the datasets from Table 5. Results are grouped by datasets.

churn ↑

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.8567 ± 0.0020	0.8570 ± 0.0017
SNN	0.8506 ± 0.0051	0.8533 ± 0.0033
MLP	0.8553 ± 0.0029	0.8582 ± 0.0008
Trompt	0.8557 ± 0.0077	–
TabPFN	0.8624 ± 0.0008	–
Excel	0.8614 ± 0.0025	0.8653 ± 0.0037
AutoINT	0.8607 ± 0.0047	0.8622 ± 0.0003
SAINT	0.8603 ± 0.0029	0.8628 ± 0.0008
FT−T	0.8593 ± 0.0028	0.8598 ± 0.0025
MLP−Mixer	0.8592 ± 0.0036	0.8630 ± 0.0005
T2G	0.8610 ± 0.0018	0.8613 ± 0.0013
TabR	0.8599 ± 0.0025	0.8620 ± 0.0023
MNCA	0.8595 ± 0.0028	0.8615 ± 0.0013
MLP†	0.8624 ± 0.0010	0.8638 ± 0.0012
MLP[PLE]	0.8580 ± 0.0028	0.8605 ± 0.0018
XGBoost	0.8605 ± 0.0022	0.8608 ± 0.0013
LightGBM	0.8600 ± 0.0008	0.8600 ± 0.0000
CatBoost	0.8582 ± 0.0017	0.8588 ± 0.0008
MNCA†	0.8606 ± 0.0032	0.8607 ± 0.0008
TabM	0.8613 ± 0.0025	0.8615 ± 0.0005
TabR†	0.8625 ± 0.0021	0.8645 ± 0.0013
$\text{TabM}^{\dagger}_{\text{mini}}$	0.8608 ± 0.0019	0.8592 ± 0.0003

california ↓

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.4971 ± 0.0122	0.4779 ± 0.0022
SNN	0.5033 ± 0.0075	0.4933 ± 0.0035
MLP	0.4948 ± 0.0058	0.4880 ± 0.0022
Trompt	0.4650 ± 0.0072	–
TabPFN	–	–
Excel	0.4553 ± 0.0043	0.4348 ± 0.0009
AutoINT	0.4682 ± 0.0063	0.4490 ± 0.0028
SAINT	0.4680 ± 0.0048	0.4575 ± 0.0014
FT−T	0.4635 ± 0.0048	0.4515 ± 0.0016
MLP−Mixer	0.4746 ± 0.0056	0.4509 ± 0.0029
T2G	0.4616 ± 0.0070	0.4439 ± 0.0026
TabR	0.4030 ± 0.0023	0.3964 ± 0.0013
MNCA	0.4239 ± 0.0012	0.4231 ± 0.0005
MLP†	0.4652 ± 0.0045	0.4549 ± 0.0006
MLP[PLE]	0.4530 ± 0.0029	0.4491 ± 0.0010
XGBoost	0.4327 ± 0.0016	0.4316 ± 0.0007
LightGBM	0.4352 ± 0.0019	0.4339 ± 0.0008
CatBoost	0.4294 ± 0.0012	0.4265 ± 0.0003
MNCA†	0.4142 ± 0.0031	0.4071 ± 0.0029
TabM	0.4509 ± 0.0032	0.4490 ± 0.0018
TabR†	0.3998 ± 0.0033	0.3914 ± 0.0020
$\text{TabM}^{\dagger}_{\text{mini}}$	0.4314 ± 0.0036	0.4261 ± 0.0019

house ↓

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	3.3327 ± 0.0878	3.1303 ± 0.0410
SNN	3.2176 ± 0.0376	3.1320 ± 0.0155
MLP	3.1117 ± 0.0294	3.0706 ± 0.0140
Trompt	3.2215 ± 0.0629	–
TabPFN	–	–
Excel	3.2432 ± 0.0511	3.1053 ± 0.0160
AutoINT	3.2157 ± 0.0436	3.1261 ± 0.0095
SAINT	3.2424 ± 0.0595	3.1067 ± 0.0253
FT−T	3.1823 ± 0.0460	3.0974 ± 0.0334
MLP−Mixer	3.1871 ± 0.0519	3.0184 ± 0.0086
T2G	3.1524 ± 0.0291	3.0918 ± 0.0073
TabR	3.0667 ± 0.0403	2.9958 ± 0.0270
MNCA	3.0884 ± 0.0286	3.0538 ± 0.0072
MLP†	3.0633 ± 0.0248	3.0170 ± 0.0070
MLP[PLE]	3.0999 ± 0.0351	3.0401 ± 0.0071
XGBoost	3.1773 ± 0.0102	3.1644 ± 0.0068
LightGBM	3.1774 ± 0.0087	3.1672 ± 0.0050
CatBoost	3.1172 ± 0.0125	3.1058 ± 0.0022
MNCA†	3.0704 ± 0.0388	3.0149 ± 0.0308
TabM	3.0002 ± 0.0182	2.9796 ± 0.0024
TabR†	3.1048 ± 0.0410	3.0246 ± 0.0101
$\text{TabM}^{\dagger}_{\text{mini}}$	2.9902 ± 0.0271	2.9648 ± 0.0035

adult ↑

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.8582 ± 0.0011	0.8593 ± 0.0002
SNN	0.8582 ± 0.0009	0.8603 ± 0.0012
MLP	0.8540 ± 0.0018	0.8559 ± 0.0011
Trompt	0.8566 ± 0.0020	–
TabPFN	–	–
Excel	0.8623 ± 0.0028	0.8645 ± 0.0004
AutoINT	0.8592 ± 0.0016	0.8612 ± 0.0004
SAINT	0.8601 ± 0.0019	0.8618 ± 0.0001
FT−T	0.8588 ± 0.0015	0.8608 ± 0.0011
MLP−Mixer	0.8598 ± 0.0013	0.8617 ± 0.0002
T2G	0.8603 ± 0.0011	0.8621 ± 0.0001
TabR	0.8646 ± 0.0022	0.8680 ± 0.0019
MNCA	0.8677 ± 0.0018	0.8696 ± 0.0003
MLP†	0.8693 ± 0.0007	0.8702 ± 0.0006
MLP[PLE]	0.8603 ± 0.0009	0.8616 ± 0.0006
XGBoost	0.8720 ± 0.0006	0.8723 ± 0.0002
LightGBM	0.8713 ± 0.0007	0.8721 ± 0.0004
CatBoost	0.8714 ± 0.0012	0.8723 ± 0.0007
MNCA†	0.8717 ± 0.0008	0.8742 ± 0.0006
TabM	0.8582 ± 0.0011	0.8588 ± 0.0003
TabR†	0.8699 ± 0.0011	0.8722 ± 0.0005
$\text{TabM}^{\dagger}_{\text{mini}}$	0.8679 ± 0.0017	0.8690 ± 0.0005

diamond ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.1420 ± 0.0032	0.1374 ± 0.0020
SNN	0.1473 ± 0.0057	0.1424 ± 0.0008
MLP	0.1404 ± 0.0012	0.1362 ± 0.0003
Trompt	0.1394 ± 0.0014	–
TabPFN	–	–
Excel	0.1765 ± 0.0024	0.1713 ± 0.0003
AutoINT	0.1392 ± 0.0014	0.1361 ± 0.0004
SAINT	0.1369 ± 0.0019	0.1343 ± 0.0011
FT−T	0.1376 ± 0.0013	0.1360 ± 0.0002
MLP−Mixer	0.1400 ± 0.0025	0.1378 ± 0.0008
T2G	0.1375 ± 0.0011	0.1349 ± 0.0007
TabR	0.1327 ± 0.0010	0.1311 ± 0.0005
MNCA	0.1370 ± 0.0018	0.1348 ± 0.0005
MLP†	0.1342 ± 0.0008	0.1325 ± 0.0004
MLP[PLE]	0.1323 ± 0.0010	0.1301 ± 0.0005
XGBoost	0.1368 ± 0.0004	0.1363 ± 0.0001
LightGBM	0.1359 ± 0.0002	0.1358 ± 0.0001
CatBoost	0.1335 ± 0.0006	0.1327 ± 0.0004
MNCA†	0.1327 ± 0.0012	0.1315 ± 0.0006
TabM	0.1342 ± 0.0017	0.1327 ± 0.0004
TabR†	0.1333 ± 0.0013	0.1312 ± 0.0005
TabM$^\dagger_{\text{mini}}$	0.1320 ± 0.0010	0.1307 ± 0.0005

otto ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8064 ± 0.0021	0.8208 ± 0.0023
SNN	0.8087 ± 0.0020	0.8156 ± 0.0013
MLP	0.8175 ± 0.0022	0.8222 ± 0.0007
Trompt	0.7875 ± 0.0087	–
TabPFN	0.7408 ± 0.0028	–
Excel	0.8101 ± 0.0033	0.8225 ± 0.0004
AutoINT	0.8050 ± 0.0034	0.8111 ± 0.0020
SAINT	0.8119 ± 0.0018	0.8193 ± 0.0024
FT−T	0.8133 ± 0.0033	0.8221 ± 0.0013
MLP−Mixer	0.8092 ± 0.0040	0.8136 ± 0.0010
T2G	0.8161 ± 0.0026	0.8268 ± 0.0024
TabR	0.8179 ± 0.0022	0.8236 ± 0.0009
MNCA	0.8275 ± 0.0012	0.8313 ± 0.0006
MLP†	0.8190 ± 0.0021	0.8271 ± 0.0015
MLP[PLE]	0.8205 ± 0.0021	0.8290 ± 0.0006
XGBoost	0.8297 ± 0.0011	0.8316 ± 0.0008
LightGBM	0.8302 ± 0.0009	0.8316 ± 0.0013
CatBoost	0.8250 ± 0.0013	0.8268 ± 0.0002
MNCA†	0.8265 ± 0.0015	0.8304 ± 0.0006
TabM	0.8268 ± 0.0014	0.8300 ± 0.0007
TabR†	0.8246 ± 0.0018	0.8309 ± 0.0014
TabM$^\dagger_{\text{mini}}$	0.8342 ± 0.0014	0.8365 ± 0.0005

higgs-small ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7164 ± 0.0030	0.7237 ± 0.0011
SNN	0.7142 ± 0.0024	0.7171 ± 0.0020
MLP	0.7180 ± 0.0027	0.7192 ± 0.0005
Trompt	0.7223 ± 0.0035	–
TabPFN	0.6727 ± 0.0034	–
Excel	0.7262 ± 0.0022	0.7324 ± 0.0005
AutoINT	0.7240 ± 0.0028	0.7287 ± 0.0008
SAINT	0.7236 ± 0.0019	0.7295 ± 0.0011
FT−T	0.7281 ± 0.0016	0.7334 ± 0.0013
MLP−Mixer	0.7248 ± 0.0023	0.7334 ± 0.0007
T2G	0.7340 ± 0.0029	0.7381 ± 0.0017
TabR	0.7223 ± 0.0010	0.7257 ± 0.0008
MNCA	0.7263 ± 0.0023	0.7292 ± 0.0006
MLP†	0.7260 ± 0.0017	0.7304 ± 0.0008
MLP[PLE]	0.7210 ± 0.0016	0.7252 ± 0.0005
XGBoost	0.7246 ± 0.0015	0.7264 ± 0.0013
LightGBM	0.7256 ± 0.0009	0.7263 ± 0.0007
CatBoost	0.7260 ± 0.0011	0.7273 ± 0.0010
MNCA†	0.7300 ± 0.0020	0.7348 ± 0.0008
TabM	0.7383 ± 0.0028	0.7409 ± 0.0010
TabR†	0.7294 ± 0.0014	0.7326 ± 0.0005
TabM$^\dagger_{\text{mini}}$	0.7348 ± 0.0017	0.7379 ± 0.0006

black-friday ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.6968 ± 0.0013	0.6936 ± 0.0007
SNN	0.6996 ± 0.0013	0.6978 ± 0.0004
MLP	0.6955 ± 0.0004	0.6942 ± 0.0002
Trompt	0.6988 ± 0.0010	–
TabPFN	–	–
Excel	0.6948 ± 0.0010	0.6901 ± 0.0006
AutoINT	0.6994 ± 0.0082	0.6927 ± 0.0021
SAINT	0.6934 ± 0.0009	0.6879 ± 0.0006
FT−T	0.6987 ± 0.0192	0.6879 ± 0.0023
MLP−Mixer	0.6905 ± 0.0021	0.6851 ± 0.0011
T2G	0.6904 ± 0.0086	0.6843 ± 0.0018
TabR	0.6899 ± 0.0004	0.6883 ± 0.0002
MNCA	0.6893 ± 0.0004	0.6883 ± 0.0000
MLP†	0.6849 ± 0.0006	0.6824 ± 0.0002
MLP[PLE]	0.6836 ± 0.0006	0.6812 ± 0.0002
XGBoost	0.6806 ± 0.0001	0.6805 ± 0.0000
LightGBM	0.6799 ± 0.0003	0.6795 ± 0.0001
CatBoost	0.6822 ± 0.0003	0.6813 ± 0.0002
MNCA†	0.6885 ± 0.0007	0.6863 ± 0.0003
TabM	0.6875 ± 0.0015	0.6866 ± 0.0003
TabR†	0.6761 ± 0.0009	0.6735 ± 0.0006
TabM$^\dagger_{\text{mini}}$	0.6807 ± 0.0013	0.6783 ± 0.0009

covtype ↑		
Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.9622 ± 0.0019	0.9673 ± 0.0011
SNN	0.9636 ± 0.0010	0.9677 ± 0.0002
MLP	0.9630 ± 0.0012	0.9664 ± 0.0004
Trompt	–	–
TabPFN	–	–
Excel	0.9606 ± 0.0018	$0.9670 \pm nan$
AutoINT	–	–
SAINT	0.9669 ± 0.0010	–
FT-T	0.9698 ± 0.0008	0.9731 ± 0.0006
MLP-Mixer	–	–
T2G	0.9666 ± 0.0009	0.9706 ± 0.0005
TabR	0.9737 ± 0.0005	0.9745 ± 0.0006
MNCA	0.9724 ± 0.0003	0.9729 ± 0.0001
MLP†	0.9690 ± 0.0008	0.9721 ± 0.0006
MLP[PLE]	0.9697 ± 0.0008	0.9721 ± 0.0005
XGBoost	0.9710 ± 0.0002	0.9713 ± 0.0000
LightGBM	–	–
CatBoost	0.9671 ± 0.0003	–
MNCA†	0.9747 ± 0.0002	0.9747 ± 0.0002
TabM	0.9712 ± 0.0008	0.9729 ± 0.0003
TabR†	0.9752 ± 0.0003	0.9759 ± 0.0003
TabM$^\dagger_{\text{mini}}$	0.9740 ± 0.0006	0.9754 ± 0.0001

microsoft ↓		
Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7499 ± 0.0003	0.7477 ± 0.0001
SNN	0.7488 ± 0.0004	0.7470 ± 0.0001
MLP	0.7475 ± 0.0003	0.7460 ± 0.0003
Trompt	0.7551 ± 0.0007	–
TabPFN	–	–
Excel	0.7477 ± 0.0006	0.7438 ± 0.0004
AutoINT	0.7482 ± 0.0005	0.7455 ± 0.0002
SAINT	0.7625 ± 0.0066	–
FT-T	0.7460 ± 0.0007	0.7422 ± 0.0004
MLP-Mixer	0.7482 ± 0.0008	0.7436 ± 0.0001
T2G	0.7461 ± 0.0006	0.7429 ± 0.0003
TabR	0.7503 ± 0.0006	0.7485 ± 0.0002
MNCA	0.7458 ± 0.0003	0.7448 ± 0.0002
MLP†	0.7446 ± 0.0002	0.7434 ± 0.0002
MLP[PLE]	0.7465 ± 0.0005	0.7448 ± 0.0001
XGBoost	0.7413 ± 0.0001	0.7410 ± 0.0000
LightGBM	0.7417 ± 0.0001	0.7413 ± 0.0000
CatBoost	0.7412 ± 0.0001	0.7406 ± 0.0000
MNCA†	0.7460 ± 0.0008	0.7435 ± 0.0004
TabM	0.7434 ± 0.0003	0.7424 ± 0.0001
TabR†	0.7501 ± 0.0005	–
TabM$^\dagger_{\text{mini}}$	0.7427 ± 0.0002	0.7416 ± 0.0002

Table 19: Extended results for the datasets from Lee et al. (2015). Results are grouped by datasets.

wine ↑		
Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7492 ± 0.0147	0.7764 ± 0.0095
SNN	0.7818 ± 0.0143	0.7994 ± 0.0097
MLP	0.7778 ± 0.0153	0.7907 ± 0.0117
Trompt	0.7665 ± 0.0117	–
TabPFN	0.7908 ± 0.0063	–
Excel	0.7619 ± 0.0144	0.7731 ± 0.0106
AutoINT	0.7745 ± 0.0144	0.7909 ± 0.0160
SAINT	0.7684 ± 0.0144	0.7821 ± 0.0105
FT-T	0.7755 ± 0.0133	0.7894 ± 0.0083
MLP-Mixer	0.7769 ± 0.0149	0.7950 ± 0.0087
T2G	0.7738 ± 0.0126	0.7894 ± 0.0149
TabR	0.7936 ± 0.0114	0.8055 ± 0.0057
MNCA	0.7911 ± 0.0135	0.8005 ± 0.0121
MLP†	0.7803 ± 0.0157	0.7964 ± 0.0146
MLP[PLE]	0.7814 ± 0.0132	0.7919 ± 0.0098
XGBoost	0.7949 ± 0.0178	0.8010 ± 0.0186
LightGBM	0.7890 ± 0.0160	0.7929 ± 0.0106
CatBoost	0.7994 ± 0.0131	0.8057 ± 0.0098
MNCA†	0.7867 ± 0.0113	0.7953 ± 0.0114
TabM	0.7961 ± 0.0136	0.8011 ± 0.0084
TabR†	0.7804 ± 0.0148	0.7945 ± 0.0118
TabM$^\dagger_{\text{mini}}$	0.7886 ± 0.0167	0.7963 ± 0.0113

phoneme ↑		
Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8342 ± 0.0151	0.8543 ± 0.0118
SNN	0.8596 ± 0.0124	0.8687 ± 0.0080
MLP	0.8525 ± 0.0126	0.8635 ± 0.0099
Trompt	0.8528 ± 0.0150	–
TabPFN	0.8684 ± 0.0050	–
Excel	0.8537 ± 0.0118	0.8685 ± 0.0074
AutoINT	0.8623 ± 0.0138	0.8754 ± 0.0095
SAINT	0.8657 ± 0.0130	0.8799 ± 0.0080
FT-T	0.8667 ± 0.0127	0.8795 ± 0.0093
MLP-Mixer	0.8629 ± 0.0123	0.8757 ± 0.0095
T2G	0.8630 ± 0.0146	0.8736 ± 0.0119
TabR	0.8781 ± 0.0096	0.8840 ± 0.0054
MNCA	0.8835 ± 0.0079	0.8861 ± 0.0057
MLP†	0.8742 ± 0.0120	0.8861 ± 0.0071
MLP[PLE]	0.8647 ± 0.0098	0.8761 ± 0.0076
XGBoost	0.8682 ± 0.0174	0.8771 ± 0.0156
LightGBM	0.8702 ± 0.0129	0.8733 ± 0.0126
CatBoost	0.8827 ± 0.0117	0.8897 ± 0.0055
MNCA†	0.8828 ± 0.0082	0.8925 ± 0.0056
TabM	0.8701 ± 0.0167	0.8766 ± 0.0128
TabR†	0.8772 ± 0.0087	0.8849 ± 0.0073
TabM$^\dagger_{\text{mini}}$	0.8790 ± 0.0098	0.8885 ± 0.0056

analcatdata_supreme ↓

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.0811 ± 0.0137	0.0759 ± 0.0086
SNN	0.0826 ± 0.0096	0.0779 ± 0.0098
MLP	0.0782 ± 0.0081	0.0766 ± 0.0090
Trompt	0.0770 ± 0.0086	–
TabPFN	–	–
Excel	0.0791 ± 0.0091	0.0773 ± 0.0090
AutoINT	0.0783 ± 0.0078	0.0768 ± 0.0083
SAINT	0.0773 ± 0.0078	0.0759 ± 0.0076
FT−T	0.0787 ± 0.0086	0.0775 ± 0.0091
MLP−Mixer	0.0770 ± 0.0082	0.0759 ± 0.0081
T2G	0.0778 ± 0.0077	0.0766 ± 0.0077
TabR	0.0803 ± 0.0066	0.0759 ± 0.0046
MNCA	0.0809 ± 0.0072	0.0784 ± 0.0062
MLP†	0.0798 ± 0.0088	0.0769 ± 0.0092
MLP[PLE]	0.0774 ± 0.0064	0.0759 ± 0.0063
XGBoost	0.0801 ± 0.0126	0.0774 ± 0.0107
LightGBM	0.0778 ± 0.0115	0.0767 ± 0.0110
CatBoost	0.0780 ± 0.0067	0.0734 ± 0.0022
MNCA†	0.0825 ± 0.0090	0.0793 ± 0.0072
TabM	0.0777 ± 0.0099	0.0769 ± 0.0105
TabR†	0.0807 ± 0.0088	0.0754 ± 0.0046
TabM$^\dagger_\text{mini}$	0.0790 ± 0.0079	0.0770 ± 0.0086

KDDCup09_upselling ↑

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.7850 ± 0.0161	0.7884 ± 0.0135
SNN	0.7884 ± 0.0122	0.7940 ± 0.0116
MLP	0.7759 ± 0.0137	0.7806 ± 0.0125
Trompt	0.7812 ± 0.0144	–
TabPFN	–	–
Excel	0.7929 ± 0.0085	0.7991 ± 0.0108
AutoINT	0.8004 ± 0.0075	0.8037 ± 0.0063
SAINT	0.7942 ± 0.0112	0.7993 ± 0.0081
FT−T	0.7957 ± 0.0127	0.7960 ± 0.0139
MLP−Mixer	0.7979 ± 0.0105	0.8010 ± 0.0094
T2G	0.7988 ± 0.0107	0.8015 ± 0.0083
TabR	0.7838 ± 0.0136	0.7859 ± 0.0167
MNCA	0.7939 ± 0.0097	0.7989 ± 0.0115
MLP†	0.7962 ± 0.0093	0.7995 ± 0.0105
MLP[PLE]	0.7925 ± 0.0123	0.7963 ± 0.0089
XGBoost	0.7930 ± 0.0108	0.7950 ± 0.0102
LightGBM	0.7932 ± 0.0119	0.7969 ± 0.0115
CatBoost	0.7992 ± 0.0117	0.8010 ± 0.0121
MNCA†	0.7960 ± 0.0131	0.8008 ± 0.0110
TabM	0.8002 ± 0.0103	0.8021 ± 0.0074
TabR†	0.7908 ± 0.0123	0.8028 ± 0.0084
TabM$^\dagger_\text{mini}$	0.8031 ± 0.0133	0.8039 ± 0.0114

kdd_ipums_la_97-small ↑

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.8770 ± 0.0072	0.8824 ± 0.0068
SNN	0.8722 ± 0.0093	0.8733 ± 0.0083
MLP	0.8828 ± 0.0061	0.8845 ± 0.0055
Trompt	0.8798 ± 0.0111	–
TabPFN	0.8578 ± 0.0046	–
Excel	0.8814 ± 0.0061	0.8822 ± 0.0052
AutoINT	0.8808 ± 0.0083	0.8830 ± 0.0081
SAINT	0.8837 ± 0.0055	0.8839 ± 0.0049
FT−T	0.8795 ± 0.0077	0.8792 ± 0.0062
MLP−Mixer	0.8762 ± 0.0100	0.8770 ± 0.0088
T2G	0.8842 ± 0.0056	0.8847 ± 0.0057
TabR	0.8798 ± 0.0081	0.8819 ± 0.0078
MNCA	0.8819 ± 0.0054	0.8832 ± 0.0048
MLP†	0.8765 ± 0.0108	0.8765 ± 0.0108
MLP[PLE]	0.8757 ± 0.0101	0.8756 ± 0.0104
XGBoost	0.8825 ± 0.0089	0.8835 ± 0.0085
LightGBM	0.8792 ± 0.0075	0.8802 ± 0.0067
CatBoost	0.8793 ± 0.0088	0.8803 ± 0.0100
MNCA†	0.8837 ± 0.0062	0.8860 ± 0.0059
TabM	0.8845 ± 0.0063	0.8848 ± 0.0070
TabR†	0.8831 ± 0.0050	0.8839 ± 0.0052
TabM$^\dagger_\text{mini}$	0.8775 ± 0.0094	0.8780 ± 0.0099

wine_quality ↓

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.7010 ± 0.0171	0.6699 ± 0.0139
SNN	0.6604 ± 0.0174	0.6245 ± 0.0140
MLP	0.6707 ± 0.0178	0.6530 ± 0.0152
Trompt	0.6871 ± 0.0104	–
TabPFN	–	–
Excel	0.6877 ± 0.0160	0.6656 ± 0.0142
AutoINT	0.6840 ± 0.0126	0.6478 ± 0.0146
SAINT	0.6797 ± 0.0161	0.6604 ± 0.0307
FT−T	0.6787 ± 0.0149	0.6564 ± 0.0250
MLP−Mixer	0.6672 ± 0.0263	0.6294 ± 0.0200
T2G	0.6802 ± 0.0162	0.6592 ± 0.0222
TabR	0.6315 ± 0.0097	0.6197 ± 0.0096
MNCA	0.6154 ± 0.0083	0.6058 ± 0.0149
MLP†	0.6569 ± 0.0167	0.6328 ± 0.0155
MLP[PLE]	0.6721 ± 0.0180	0.6463 ± 0.0262
XGBoost	0.6039 ± 0.0134	0.6025 ± 0.0139
LightGBM	0.6135 ± 0.0138	0.6122 ± 0.0144
CatBoost	0.6088 ± 0.0132	0.6060 ± 0.0137
MNCA†	0.6099 ± 0.0144	0.6028 ± 0.0157
TabM	0.6169 ± 0.0123	0.6131 ± 0.0126
TabR†	0.6412 ± 0.0105	0.6202 ± 0.0066
TabM$^\dagger_\text{mini}$	0.6255 ± 0.0146	0.6194 ± 0.0150

isolet ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	2.2449 ± 0.1579	2.0176 ± 0.0770
SNN	2.4269 ± 0.2382	2.1142 ± 0.1262
MLP	2.2744 ± 0.2203	2.0018 ± 0.1111
Trompt	2.7814 ± 0.0885	–
TabPFN	–	–
Excel	2.8877 ± 0.1027	2.6073 ± 0.0731
AutoINT	2.6130 ± 0.1658	2.3308 ± 0.1088
SAINT	–	–
FT−T	2.4879 ± 0.2524	2.1501 ± 0.1506
MLP−Mixer	2.3344 ± 0.2073	2.0915 ± 0.1159
T2G	2.2700 ± 0.2384	1.9258 ± 0.1408
TabR	1.9760 ± 0.1738	1.7627 ± 0.1520
MNCA	1.7905 ± 0.1594	1.6205 ± 0.1676
MLP†	2.2719 ± 0.1006	2.1026 ± 0.1088
MLP[PLE]	2.0979 ± 0.1779	1.9283 ± 0.1334
XGBoost	2.7567 ± 0.0470	2.7294 ± 0.0366
LightGBM	2.7005 ± 0.0296	2.6903 ± 0.0290
CatBoost	2.8852 ± 0.0225	2.8480 ± 0.0020
MNCA†	1.8912 ± 0.1851	1.7147 ± 0.1348
TabM	1.8831 ± 0.1194	1.8578 ± 0.1088
TabR†	1.9919 ± 0.1813	1.7483 ± 0.1434
TabM$^\dagger_{\text{mini}}$	1.8378 ± 0.0803	1.8126 ± 0.0692

cpu_act ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	2.7868 ± 0.1999	2.4884 ± 0.0327
SNN	2.5811 ± 0.1480	2.3863 ± 0.0324
MLP	2.6814 ± 0.2291	2.4953 ± 0.1150
Trompt	2.3872 ± 0.2610	–
TabPFN	–	–
Excel	2.3507 ± 0.2540	2.2001 ± 0.1574
AutoINT	2.2537 ± 0.0536	2.1708 ± 0.0349
SAINT	2.2781 ± 0.0630	2.2032 ± 0.0310
FT−T	2.2394 ± 0.0508	2.1494 ± 0.0268
MLP−Mixer	2.3079 ± 0.0829	2.1831 ± 0.0470
T2G	2.2100 ± 0.0404	2.1280 ± 0.0300
TabR	2.2980 ± 0.0529	2.2228 ± 0.0501
MNCA	2.2603 ± 0.0479	2.2339 ± 0.0508
MLP†	2.2730 ± 0.0457	2.1899 ± 0.0419
MLP[PLE]	2.3309 ± 0.0719	2.2516 ± 0.0574
XGBoost	2.5237 ± 0.3530	2.4723 ± 0.3789
LightGBM	2.2223 ± 0.0894	2.2067 ± 0.0916
CatBoost	2.1239 ± 0.0489	2.1092 ± 0.0499
MNCA†	2.2105 ± 0.0483	2.1396 ± 0.0474
TabM	2.1940 ± 0.0523	2.1677 ± 0.0487
TabR†	2.1278 ± 0.0783	2.0631 ± 0.0502
TabM$^\dagger_{\text{mini}}$	2.1572 ± 0.0376	2.1222 ± 0.0358

visualizing_soil ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.3547 ± 0.2726	0.2549 ± 0.1517
SNN	0.3642 ± 0.2350	0.1599 ± 0.0406
MLP	0.1461 ± 0.0152	0.1338 ± 0.0073
Trompt	0.1289 ± 0.0023	–
TabPFN	–	–
Excel	0.1528 ± 0.0200	0.1276 ± 0.0044
AutoINT	0.1598 ± 0.0724	0.1357 ± 0.0655
SAINT	0.1368 ± 0.0155	0.1235 ± 0.0051
FT−T	0.1443 ± 0.0235	0.1250 ± 0.0104
MLP−Mixer	0.1431 ± 0.0472	0.1323 ± 0.0420
T2G	0.2878 ± 0.2651	0.2706 ± 0.2550
TabR	0.3979 ± 0.3523	0.3869 ± 0.3746
MNCA	0.3642 ± 0.3482	0.3626 ± 0.3660
MLP†	0.1601 ± 0.0785	0.1396 ± 0.0630
MLP[PLE]	0.1063 ± 0.0239	0.0973 ± 0.0180
XGBoost	0.1765 ± 0.0707	0.1539 ± 0.0539
LightGBM	0.0616 ± 0.0159	0.0616 ± 0.0167
CatBoost	0.0554 ± 0.0063	0.0468 ± 0.0059
MNCA†	0.2367 ± 0.3529	0.2290 ± 0.2782
TabM	0.1242 ± 0.0188	0.1171 ± 0.0118
TabR†	0.2268 ± 0.2641	0.2021 ± 0.1468
TabM$^\dagger_{\text{mini}}$	0.1060 ± 0.0243	0.1043 ± 0.0234

sulfur ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0247 ± 0.0050	0.0208 ± 0.0050
SNN	0.0209 ± 0.0034	0.0194 ± 0.0038
MLP	0.0217 ± 0.0024	0.0204 ± 0.0028
Trompt	0.0252 ± 0.0048	–
TabPFN	–	–
Excel	0.0259 ± 0.0053	0.0251 ± 0.0051
AutoINT	0.0206 ± 0.0035	0.0192 ± 0.0034
SAINT	0.0199 ± 0.0028	0.0178 ± 0.0022
FT−T	0.0215 ± 0.0042	0.0201 ± 0.0037
MLP−Mixer	0.0199 ± 0.0034	0.0184 ± 0.0032
T2G	0.0218 ± 0.0031	0.0200 ± 0.0025
TabR	0.0222 ± 0.0022	0.0208 ± 0.0021
MNCA	0.0198 ± 0.0030	0.0189 ± 0.0020
MLP†	0.0192 ± 0.0032	0.0181 ± 0.0028
MLP[PLE]	0.0197 ± 0.0026	0.0187 ± 0.0029
XGBoost	0.0202 ± 0.0019	0.0200 ± 0.0017
LightGBM	0.0203 ± 0.0020	0.0200 ± 0.0015
CatBoost	0.0189 ± 0.0022	0.0185 ± 0.0022
MNCA†	0.0198 ± 0.0029	0.0185 ± 0.0032
TabM	0.0192 ± 0.0035	0.0184 ± 0.0030
TabR†	0.0217 ± 0.0031	0.0201 ± 0.0028
TabM$^\dagger_{\text{mini}}$	0.0197 ± 0.0042	0.0192 ± 0.0045

bank-marketing ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7859 ± 0.0068	0.7917 ± 0.0078
SNN	0.7836 ± 0.0074	0.7882 ± 0.0054
MLP	0.7860 ± 0.0057	0.7887 ± 0.0052
Trompt	0.7922 ± 0.0114	–
TabPFN	0.7894 ± 0.0091	–
Excel	0.7957 ± 0.0092	0.7998 ± 0.0088
AutoINT	0.7917 ± 0.0071	0.7956 ± 0.0058
SAINT	0.7953 ± 0.0058	0.7974 ± 0.0050
FT−T	0.7918 ± 0.0076	0.7951 ± 0.0071
MLP−Mixer	0.7954 ± 0.0059	0.8001 ± 0.0048
T2G	0.7930 ± 0.0064	0.7957 ± 0.0037
TabR	0.7995 ± 0.0054	0.8015 ± 0.0037
MNCA	0.7961 ± 0.0065	0.8003 ± 0.0077
MLP†	0.7947 ± 0.0101	0.7977 ± 0.0117
MLP[PLE]	0.7981 ± 0.0065	0.8008 ± 0.0057
XGBoost	0.8013 ± 0.0081	0.8030 ± 0.0076
LightGBM	0.8006 ± 0.0078	0.8013 ± 0.0072
CatBoost	0.8026 ± 0.0068	0.8056 ± 0.0082
MNCA†	0.7977 ± 0.0081	0.8010 ± 0.0084
TabM	0.7908 ± 0.0068	0.7915 ± 0.0068
TabR†	0.8023 ± 0.0088	0.8037 ± 0.0096
TabM$^\dagger_\text{mini}$	0.8003 ± 0.0087	0.8017 ± 0.0087

Brazilian_houses ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0477 ± 0.0172	0.0427 ± 0.0207
SNN	0.0630 ± 0.0162	0.0556 ± 0.0175
MLP	0.0473 ± 0.0179	0.0440 ± 0.0207
Trompt	0.0428 ± 0.0295	–
TabPFN	–	–
Excel	0.0457 ± 0.0167	0.0424 ± 0.0186
AutoINT	0.0470 ± 0.0192	0.0437 ± 0.0217
SAINT	0.0479 ± 0.0205	0.0426 ± 0.0236
FT−T	0.0438 ± 0.0181	0.0412 ± 0.0204
MLP−Mixer	0.0513 ± 0.0234	0.0484 ± 0.0262
T2G	0.0465 ± 0.0167	0.0432 ± 0.0188
TabR	0.0490 ± 0.0152	0.0454 ± 0.0170
MNCA	0.0527 ± 0.0157	0.0509 ± 0.0180
MLP†	0.0426 ± 0.0180	0.0397 ± 0.0206
MLP[PLE]	0.0421 ± 0.0209	0.0409 ± 0.0226
XGBoost	0.0541 ± 0.0270	0.0535 ± 0.0287
LightGBM	0.0603 ± 0.0249	0.0589 ± 0.0271
CatBoost	0.0468 ± 0.0312	0.0456 ± 0.0332
MNCA†	0.0553 ± 0.0192	0.0511 ± 0.0191
TabM	0.0443 ± 0.0213	0.0431 ± 0.0233
TabR†	0.0451 ± 0.0163	0.0413 ± 0.0174
TabM$^\dagger_\text{mini}$	0.0460 ± 0.0206	0.0439 ± 0.0228

MagicTelescope ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8432 ± 0.0074	0.8490 ± 0.0046
SNN	0.8536 ± 0.0052	0.8567 ± 0.0047
MLP	0.8539 ± 0.0060	0.8566 ± 0.0061
Trompt	0.8484 ± 0.0058	–
TabPFN	0.8579 ± 0.0064	–
Excel	0.8498 ± 0.0078	0.8571 ± 0.0057
AutoINT	0.8522 ± 0.0056	0.8560 ± 0.0034
SAINT	0.8595 ± 0.0060	0.8632 ± 0.0061
FT−T	0.8588 ± 0.0046	0.8643 ± 0.0037
MLP−Mixer	0.8571 ± 0.0080	0.8624 ± 0.0044
T2G	0.8563 ± 0.0054	0.8617 ± 0.0037
TabR	0.8682 ± 0.0058	0.8729 ± 0.0038
MNCA	0.8602 ± 0.0061	0.8628 ± 0.0041
MLP†	0.8591 ± 0.0061	0.8626 ± 0.0044
MLP[PLE]	0.8593 ± 0.0054	0.8621 ± 0.0037
XGBoost	0.8550 ± 0.0094	0.8589 ± 0.0110
LightGBM	0.8547 ± 0.0085	0.8556 ± 0.0086
CatBoost	0.8586 ± 0.0070	0.8588 ± 0.0077
MNCA†	0.8622 ± 0.0085	0.8681 ± 0.0064
TabM	0.8607 ± 0.0058	0.8622 ± 0.0050
TabR†	0.8641 ± 0.0052	0.8680 ± 0.0020
TabM$^\dagger_\text{mini}$	0.8616 ± 0.0080	0.8646 ± 0.0075

Ailerons ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0002 ± 0.0000	0.0002 ± 0.0000
SNN	0.0002 ± 0.0000	0.0002 ± 0.0000
MLP	0.0002 ± 0.0000	0.0002 ± 0.0000
Trompt	0.0002 ± 0.0000	–
TabPFN	–	–
Excel	0.0002 ± 0.0000	0.0002 ± 0.0000
AutoINT	0.0002 ± 0.0000	0.0002 ± 0.0000
SAINT	0.0002 ± 0.0000	0.0002 ± 0.0000
FT−T	0.0002 ± 0.0000	0.0002 ± 0.0000
MLP−Mixer	0.0002 ± 0.0000	0.0002 ± 0.0000
T2G	0.0002 ± 0.0000	0.0002 ± 0.0000
TabR	0.0002 ± 0.0000	0.0002 ± 0.0000
MNCA	0.0002 ± 0.0000	0.0002 ± 0.0000
MLP†	0.0002 ± 0.0000	0.0002 ± 0.0000
MLP[PLE]	0.0002 ± 0.0000	0.0002 ± 0.0000
XGBoost	0.0002 ± 0.0000	0.0002 ± 0.0000
LightGBM	0.0002 ± 0.0000	0.0002 ± 0.0000
CatBoost	0.0002 ± 0.0000	0.0002 ± 0.0000
MNCA†	0.0002 ± 0.0000	0.0002 ± 0.0000
TabM	0.0002 ± 0.0000	0.0002 ± 0.0000
TabR†	0.0002 ± 0.0000	0.0002 ± 0.0000
TabM$^\dagger_\text{mini}$	0.0002 ± 0.0000	0.0002 ± 0.0000

MiamiHousing2016 ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.1683 ± 0.0099	0.1575 ± 0.0047
SNN	0.1618 ± 0.0029	0.1557 ± 0.0021
MLP	0.1614 ± 0.0033	0.1574 ± 0.0043
Trompt	0.1536 ± 0.0045	–
TabPFN	–	–
Excel	0.1520 ± 0.0033	0.1441 ± 0.0024
AutoINT	0.1537 ± 0.0035	0.1478 ± 0.0027
SAINT	0.1507 ± 0.0022	0.1471 ± 0.0023
FT-T	0.1514 ± 0.0029	0.1462 ± 0.0031
MLP-Mixer	0.1527 ± 0.0037	0.1479 ± 0.0033
T2G	0.1511 ± 0.0022	0.1470 ± 0.0021
TabR	0.1417 ± 0.0025	0.1390 ± 0.0020
MNCA	0.1503 ± 0.0040	0.1477 ± 0.0032
MLP†	0.1514 ± 0.0025	0.1479 ± 0.0017
MLP[PLE]	0.1461 ± 0.0015	0.1433 ± 0.0022
XGBoost	0.1440 ± 0.0029	0.1434 ± 0.0029
LightGBM	0.1461 ± 0.0025	0.1455 ± 0.0030
CatBoost	0.1417 ± 0.0021	0.1408 ± 0.0026
MNCA†	0.1475 ± 0.0031	0.1438 ± 0.0024
TabM	0.1483 ± 0.0030	0.1465 ± 0.0029
TabR†	0.1392 ± 0.0023	0.1364 ± 0.0021
TabM$^{\dagger}_{\text{mini}}$	0.1407 ± 0.0016	0.1387 ± 0.0008

OnlineNewsPopularity ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8714 ± 0.0013	0.8648 ± 0.0004
SNN	0.8692 ± 0.0015	0.8665 ± 0.0005
MLP	0.8643 ± 0.0007	0.8632 ± 0.0005
Trompt	$0.8671 \pm nan$	–
TabPFN	–	–
Excel	0.8609 ± 0.0018	0.8562 ± 0.0010
AutoINT	0.8636 ± 0.0022	0.8596 ± 0.0008
SAINT	0.8600 ± 0.0007	0.8582 ± 0.0003
FT-T	0.8629 ± 0.0019	0.8603 ± 0.0000
MLP-Mixer	0.8615 ± 0.0008	0.8598 ± 0.0004
T2G	0.8637 ± 0.0015	0.8587 ± 0.0013
TabR	0.8677 ± 0.0013	0.8633 ± 0.0009
MNCA	0.8651 ± 0.0003	0.8650 ± 0.0002
MLP†	0.8604 ± 0.0009	0.8591 ± 0.0004
MLP[PLE]	0.8585 ± 0.0003	0.8581 ± 0.0001
XGBoost	0.8545 ± 0.0002	0.8543 ± 0.0000
LightGBM	0.8546 ± 0.0002	0.8544 ± 0.0000
CatBoost	0.8532 ± 0.0003	0.8527 ± 0.0001
MNCA†	0.8647 ± 0.0010	0.8624 ± 0.0006
TabM	0.8584 ± 0.0003	0.8581 ± 0.0001
TabR†	0.8624 ± 0.0011	0.8589 ± 0.0003
TabM$^{\dagger}_{\text{mini}}$	0.8560 ± 0.0015	0.8532 ± 0.0008

Bike_Sharing_Demand ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	45.2596 ± 0.9906	43.2049 ± 0.3088
SNN	48.0917 ± 1.1852	44.6840 ± 1.0755
MLP	45.0186 ± 0.7700	43.2726 ± 0.5498
Trompt	44.8162 ± 0.1210	–
TabPFN	–	–
Excel	43.5201 ± 1.0363	40.9541 ± 0.3714
AutoINT	43.5852 ± 0.7439	41.6339 ± 0.2132
SAINT	42.7850 ± 0.4637	41.8555 ± 0.4083
FT-T	43.2031 ± 0.4889	41.1763 ± 0.3443
MLP-Mixer	43.1481 ± 0.6971	40.8738 ± 0.3218
T2G	42.8300 ± 0.6775	41.1650 ± 0.3333
TabR	43.6370 ± 0.6814	42.3390 ± 0.4146
MNCA	44.8100 ± 0.5191	44.4483 ± 0.4231
MLP†	43.1846 ± 1.1145	41.3309 ± 0.2381
MLP[PLE]	42.5106 ± 0.4022	41.4351 ± 0.1280
XGBoost	42.7657 ± 0.1260	42.6060 ± 0.0391
LightGBM	42.5028 ± 0.1896	42.3416 ± 0.1492
CatBoost	40.9275 ± 0.2316	40.5515 ± 0.0898
MNCA†	42.6308 ± 0.8834	41.6584 ± 0.5771
TabM	42.1081 ± 0.5016	41.3316 ± 0.3496
TabR†	42.6486 ± 0.9394	41.2265 ± 0.6146
TabM$^{\dagger}_{\text{mini}}$	41.3374 ± 0.6326	40.4473 ± 0.5201

credit ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7703 ± 0.0034	0.7746 ± 0.0026
SNN	0.7712 ± 0.0045	0.7716 ± 0.0059
MLP	0.7735 ± 0.0042	0.7729 ± 0.0047
Trompt	0.7731 ± 0.0050	–
TabPFN	0.7636 ± 0.0045	–
Excel	0.7726 ± 0.0043	0.7745 ± 0.0054
AutoINT	0.7737 ± 0.0050	0.7765 ± 0.0058
SAINT	0.7739 ± 0.0052	0.7749 ± 0.0066
FT-T	0.7745 ± 0.0041	0.7767 ± 0.0040
MLP-Mixer	0.7748 ± 0.0038	0.7768 ± 0.0059
T2G	0.7747 ± 0.0047	0.7767 ± 0.0043
TabR	0.7730 ± 0.0043	0.7740 ± 0.0040
MNCA	0.7739 ± 0.0032	0.7757 ± 0.0026
MLP†	0.7749 ± 0.0055	0.7767 ± 0.0075
MLP[PLE]	0.7758 ± 0.0040	0.7772 ± 0.0055
XGBoost	0.7698 ± 0.0027	0.7706 ± 0.0029
LightGBM	0.7686 ± 0.0028	0.7726 ± 0.0034
CatBoost	0.7734 ± 0.0035	0.7752 ± 0.0038
MNCA†	0.7734 ± 0.0045	0.7754 ± 0.0040
TabM	0.7751 ± 0.0042	0.7755 ± 0.0049
TabR†	0.7723 ± 0.0037	0.7750 ± 0.0029
TabM$^{\dagger}_{\text{mini}}$	0.7748 ± 0.0026	0.7757 ± 0.0036

elevators ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0019 ± 0.0000	0.0019 ± 0.0000
SNN	0.0020 ± 0.0001	0.0019 ± 0.0000
MLP	0.0020 ± 0.0001	0.0019 ± 0.0000
Trompt	0.0019 ± 0.0001	–
TabPFN	–	–
Excel	0.0019 ± 0.0000	0.0018 ± 0.0000
AutoINT	0.0019 ± 0.0000	0.0018 ± 0.0000
SAINT	0.0018 ± 0.0000	0.0018 ± 0.0000
FT−T	0.0019 ± 0.0000	0.0018 ± 0.0000
MLP−Mixer	0.0019 ± 0.0000	0.0018 ± 0.0000
T2G	0.0019 ± 0.0000	0.0018 ± 0.0000
TabR	0.0049 ± 0.0000	0.0049 ± 0.0000
MNCA	0.0019 ± 0.0000	0.0019 ± 0.0000
$\text{MLP}^{\dagger}$	0.0019 ± 0.0000	0.0018 ± 0.0000
MLP[PLE]	0.0018 ± 0.0000	0.0018 ± 0.0000
XGBoost	0.0020 ± 0.0000	0.0020 ± 0.0000
LightGBM	0.0020 ± 0.0000	0.0020 ± 0.0000
CatBoost	0.0020 ± 0.0000	0.0019 ± 0.0000
$\text{MNCA}^{\dagger}$	0.0018 ± 0.0000	0.0018 ± 0.0000
TabM	0.0019 ± 0.0000	0.0018 ± 0.0000
$\text{TabR}^{\dagger}$	0.0019 ± 0.0001	0.0018 ± 0.0001
$\text{TabM}^{\dagger}_{\text{mini}}$	0.0018 ± 0.0000	0.0018 ± 0.0000

fifa ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8046 ± 0.0135	0.7993 ± 0.0129
SNN	0.8074 ± 0.0140	0.8031 ± 0.0147
MLP	0.8038 ± 0.0124	0.8011 ± 0.0143
Trompt	0.7920 ± 0.0157	–
TabPFN	–	–
Excel	0.7915 ± 0.0119	0.7869 ± 0.0139
AutoINT	0.7923 ± 0.0128	0.7886 ± 0.0127
SAINT	0.7901 ± 0.0118	0.7851 ± 0.0119
FT−T	0.7928 ± 0.0132	0.7888 ± 0.0130
MLP−Mixer	0.7936 ± 0.0119	0.7903 ± 0.0133
T2G	0.7944 ± 0.0134	0.7920 ± 0.0141
TabR	0.7902 ± 0.0119	0.7863 ± 0.0120
MNCA	0.7967 ± 0.0138	0.7933 ± 0.0145
$\text{MLP}^{\dagger}$	0.7940 ± 0.0118	0.7898 ± 0.0141
MLP[PLE]	0.7806 ± 0.0104	0.7800 ± 0.0114
XGBoost	0.7800 ± 0.0108	0.7795 ± 0.0114
LightGBM	0.7806 ± 0.0120	0.7787 ± 0.0122
CatBoost	0.7835 ± 0.0116	0.7817 ± 0.0114
$\text{MNCA}^{\dagger}$	0.7909 ± 0.0107	0.7866 ± 0.0106
TabM	0.7974 ± 0.0144	0.7954 ± 0.0160
$\text{TabR}^{\dagger}$	0.7914 ± 0.0136	0.7865 ± 0.0159
$\text{TabM}^{\dagger}_{\text{mini}}$	0.7783 ± 0.0114	0.7768 ± 0.0123

house_sales ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.1862 ± 0.0032	0.1778 ± 0.0015
SNN	0.1800 ± 0.0008	0.1770 ± 0.0004
MLP	0.1790 ± 0.0009	0.1763 ± 0.0003
Trompt	$0.1716 \pm nan$	–
TabPFN	–	–
Excel	0.1718 ± 0.0009	0.1667 ± 0.0002
AutoINT	0.1700 ± 0.0014	0.1670 ± 0.0008
SAINT	0.1713 ± 0.0015	0.1685 ± 0.0005
FT−T	0.1690 ± 0.0010	0.1659 ± 0.0004
MLP−Mixer	0.1704 ± 0.0007	0.1690 ± 0.0005
T2G	0.1693 ± 0.0011	0.1664 ± 0.0003
TabR	0.1689 ± 0.0009	0.1657 ± 0.0003
MNCA	0.1737 ± 0.0013	0.1714 ± 0.0005
$\text{MLP}^{\dagger}$	0.1699 ± 0.0008	0.1687 ± 0.0007
MLP[PLE]	0.1687 ± 0.0004	0.1681 ± 0.0001
XGBoost	0.1694 ± 0.0003	0.1689 ± 0.0001
LightGBM	0.1692 ± 0.0004	0.1686 ± 0.0001
CatBoost	0.1669 ± 0.0001	0.1667 ± 0.0000
$\text{MNCA}^{\dagger}$	0.1694 ± 0.0007	0.1670 ± 0.0003
TabM	0.1692 ± 0.0011	0.1680 ± 0.0005
$\text{TabR}^{\dagger}$	0.1636 ± 0.0009	0.1606 ± 0.0002
$\text{TabM}^{\dagger}_{\text{mini}}$	0.1656 ± 0.0005	0.1647 ± 0.0002

medical_charges ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0818 ± 0.0003	0.0815 ± 0.0001
SNN	0.0827 ± 0.0006	0.0817 ± 0.0001
MLP	0.0816 ± 0.0001	0.0814 ± 0.0000
Trompt	$0.0814 \pm nan$	–
TabPFN	–	–
Excel	0.0817 ± 0.0003	0.0813 ± 0.0000
AutoINT	0.0822 ± 0.0007	0.0814 ± 0.0001
SAINT	0.0814 ± 0.0002	0.0812 ± 0.0001
FT−T	0.0814 ± 0.0002	0.0812 ± 0.0000
MLP−Mixer	0.0814 ± 0.0002	0.0811 ± 0.0000
T2G	0.0815 ± 0.0004	0.0812 ± 0.0001
TabR	0.0815 ± 0.0002	0.0812 ± 0.0000
MNCA	0.0811 ± 0.0001	0.0810 ± 0.0000
$\text{MLP}^{\dagger}$	0.0812 ± 0.0002	0.0810 ± 0.0000
MLP[PLE]	0.0812 ± 0.0000	0.0811 ± 0.0000
XGBoost	0.0825 ± 0.0001	0.0825 ± 0.0000
LightGBM	0.0820 ± 0.0000	0.0820 ± 0.0000
CatBoost	0.0816 ± 0.0000	0.0815 ± 0.0000
$\text{MNCA}^{\dagger}$	0.0809 ± 0.0000	0.0808 ± 0.0000
TabM	0.0813 ± 0.0001	0.0812 ± 0.0000
$\text{TabR}^{\dagger}$	0.0811 ± 0.0001	0.0810 ± 0.0000
$\text{TabM}^{\dagger}_{\text{mini}}$	0.0812 ± 0.0001	0.0812 ± 0.0000

pol $\downarrow$

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	6.5374 ± 0.9479	5.1814 ± 0.7775
SNN	6.1816 ± 0.7366	5.5959 ± 0.8243
MLP	5.5244 ± 0.5768	4.9945 ± 0.5923
Trompt	3.2484 ± 0.4095	–
TabPFN	–	–
Excel	3.0397 ± 0.2359	2.5502 ± 0.0939
AutoINT	3.3295 ± 0.3379	2.7999 ± 0.1776
SAINT	2.7203 ± 0.1858	2.4507 ± 0.1153
FT−T	2.6974 ± 0.1666	2.3718 ± 0.0724
MLP−Mixer	3.2011 ± 0.2921	2.8698 ± 0.2577
T2G	2.9195 ± 0.1601	2.5973 ± 0.0890
TabR	6.0708 ± 0.5368	5.5578 ± 0.4036
MNCA	5.7878 ± 0.4884	5.3773 ± 0.5463
MLP†	2.8239 ± 0.2173	2.5266 ± 0.0605
MLP[PLE]	2.4958 ± 0.1292	2.3651 ± 0.1223
XGBoost	4.2963 ± 0.0644	4.2548 ± 0.0488
LightGBM	4.2320 ± 0.3369	4.1880 ± 0.3110
CatBoost	3.6320 ± 0.1006	3.5505 ± 0.0896
MNCA†	2.9083 ± 0.1364	2.6717 ± 0.0530
TabM	3.3595 ± 0.4017	3.2130 ± 0.3979
TabR†	2.5770 ± 0.1689	2.3258 ± 0.0577
TabM$^\dagger_{\mathrm{mini}}$	2.4893 ± 0.1620	2.4175 ± 0.1124

superconduct $\downarrow$

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	10.8108 ± 0.0957	10.4342 ± 0.0179
SNN	10.8562 ± 0.1300	10.3342 ± 0.0509
MLP	10.8740 ± 0.0868	10.4118 ± 0.0429
Trompt	$11.2139 \pm nan$	–
TabPFN	–	–
Excel	11.1609 ± 0.1977	10.4870 ± 0.0699
AutoINT	11.0019 ± 0.1391	10.4469 ± 0.0521
SAINT	10.7807 ± 0.1074	10.4652 ± 0.0267
FT−T	10.8256 ± 0.1692	10.3391 ± 0.0794
MLP−Mixer	10.7502 ± 0.0800	10.3281 ± 0.0450
T2G	10.8731 ± 0.1527	10.3313 ± 0.0739
TabR	10.8842 ± 0.1073	10.4800 ± 0.0280
MNCA	10.4419 ± 0.0640	10.2926 ± 0.0261
MLP†	10.5058 ± 0.0758	10.2322 ± 0.0463
MLP[PLE]	10.7220 ± 0.0757	10.3758 ± 0.0606
XGBoost	10.1610 ± 0.0201	10.1413 ± 0.0025
LightGBM	10.1634 ± 0.0118	10.1552 ± 0.0050
CatBoost	10.2422 ± 0.0222	10.2116 ± 0.0058
MNCA†	10.5651 ± 0.0616	10.3155 ± 0.0253
TabM	10.3379 ± 0.0338	10.1943 ± 0.0291
TabR†	10.3835 ± 0.0562	10.1366 ± 0.0232
TabM$^\dagger_{\mathrm{mini}}$	10.2083 ± 0.0591	10.0737 ± 0.0222

jannis $\uparrow$

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.7712 ± 0.0029	0.7825 ± 0.0009
SNN	0.7818 ± 0.0025	0.7859 ± 0.0011
MLP	0.7840 ± 0.0018	0.7872 ± 0.0007
Trompt	$0.7948 \pm nan$	–
TabPFN	0.7419 ± 0.0018	–
Excel	0.7965 ± 0.0026	0.8034 ± 0.0014
AutoINT	0.7933 ± 0.0018	0.7983 ± 0.0013
SAINT	0.7971 ± 0.0028	0.8033 ± 0.0008
FT−T	0.7940 ± 0.0028	0.7998 ± 0.0006
MLP−Mixer	0.7927 ± 0.0025	0.8019 ± 0.0012
T2G	0.8011 ± 0.0029	0.8057 ± 0.0005
TabR	0.7983 ± 0.0022	0.8023 ± 0.0018
MNCA	0.7993 ± 0.0019	0.8042 ± 0.0013
MLP†	0.7923 ± 0.0018	0.7945 ± 0.0010
MLP[PLE]	0.7891 ± 0.0013	0.7900 ± 0.0006
XGBoost	0.7967 ± 0.0019	0.7998 ± 0.0007
LightGBM	0.7956 ± 0.0017	0.7968 ± 0.0005
CatBoost	0.7985 ± 0.0018	0.8009 ± 0.0012
MNCA†	0.8068 ± 0.0021	0.8128 ± 0.0007
TabM	0.8066 ± 0.0015	0.8075 ± 0.0004
TabR†	0.8051 ± 0.0023	0.8114 ± 0.0013
TabM$^\dagger_{\mathrm{mini}}$	0.8059 ± 0.0018	0.8085 ± 0.0006

MiniBooNE $\uparrow$

Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.9433 ± 0.0011	0.9470 ± 0.0010
SNN	0.9476 ± 0.0013	0.9491 ± 0.0010
MLP	0.9480 ± 0.0007	0.9498 ± 0.0001
Trompt	$0.9393 \pm nan$	–
TabPFN	0.9266 ± 0.0012	–
Excel	0.9436 ± 0.0017	0.9460 ± 0.0008
AutoINT	0.9447 ± 0.0014	0.9473 ± 0.0010
SAINT	0.9471 ± 0.0009	0.9485 ± 0.0002
FT−T	0.9467 ± 0.0014	0.9486 ± 0.0010
MLP−Mixer	0.9446 ± 0.0014	0.9483 ± 0.0002
T2G	0.9474 ± 0.0010	0.9504 ± 0.0005
TabR	0.9487 ± 0.0008	0.9500 ± 0.0002
MNCA	0.9488 ± 0.0010	0.9505 ± 0.0001
MLP†	0.9466 ± 0.0009	0.9478 ± 0.0004
MLP[PLE]	0.9482 ± 0.0008	0.9492 ± 0.0001
XGBoost	0.9436 ± 0.0006	0.9452 ± 0.0003
LightGBM	0.9422 ± 0.0009	0.9427 ± 0.0003
CatBoost	0.9453 ± 0.0008	0.9459 ± 0.0005
MNCA†	0.9493 ± 0.0012	0.9501 ± 0.0008
TabM	0.9500 ± 0.0005	0.9505 ± 0.0002
TabR†	0.9475 ± 0.0007	0.9489 ± 0.0002
TabM$^\dagger_{\mathrm{mini}}$	0.9497 ± 0.0006	0.9508 ± 0.0003

SGEMM_GPU_kernel_performance ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.0161 ± 0.0005	0.0157 ± 0.0002
SNN	0.0191 ± 0.0008	0.0169 ± 0.0001
MLP	0.0165 ± 0.0003	0.0160 ± 0.0001
Trompt	$0.0165 \pm nan$	–
TabPFN	–	–
Excel	0.0168 ± 0.0007	0.0158 ± 0.0002
AutoINT	0.0165 ± 0.0004	0.0160 ± 0.0003
SAINT	0.0158 ± 0.0002	0.0155 ± 0.0001
FT−T	0.0167 ± 0.0007	0.0159 ± 0.0004
MLP−Mixer	0.0164 ± 0.0004	0.0158 ± 0.0002
T2G	0.0165 ± 0.0006	0.0156 ± 0.0002
TabR	0.0174 ± 0.0014	0.0161 ± 0.0005
MNCA	0.0147 ± 0.0000	0.0146 ± 0.0000
MLP†	0.0160 ± 0.0003	0.0156 ± 0.0000
MLP[PLE]	0.0156 ± 0.0000	0.0154 ± 0.0000
XGBoost	0.0167 ± 0.0000	0.0167 ± 0.0000
LightGBM	0.0168 ± 0.0000	0.0168 ± 0.0000
CatBoost	0.0168 ± 0.0000	0.0166 ± 0.0000
MNCA†	0.0146 ± 0.0002	0.0145 ± 0.0000
TabM	0.0158 ± 0.0004	0.0155 ± 0.0001
TabR†	0.0154 ± 0.0005	0.0150 ± 0.0002
TabM$^\dagger_{\text{mini}}$	0.0156 ± 0.0003	0.0154 ± 0.0001

nyc-taxi-green-dec-2016 ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.3919 ± 0.0009	0.3889 ± 0.0003
SNN	0.3933 ± 0.0013	0.3899 ± 0.0004
MLP	0.3951 ± 0.0009	0.3921 ± 0.0003
Trompt	$0.4574 \pm nan$	–
TabPFN	–	–
Excel	0.3968 ± 0.0026	0.3894 ± 0.0003
AutoINT	0.4084 ± 0.0256	0.3967 ± 0.0059
SAINT	0.3905 ± 0.0013	0.3876 ± 0.0002
FT−T	0.3937 ± 0.0064	0.3889 ± 0.0018
MLP−Mixer	0.3914 ± 0.0026	0.3861 ± 0.0013
T2G	0.3907 ± 0.0029	0.3860 ± 0.0011
TabR	0.3577 ± 0.0222	0.3380 ± 0.0027
MNCA	0.3728 ± 0.0012	0.3720 ± 0.0010
MLP†	0.3812 ± 0.0018	0.3761 ± 0.0016
MLP[PLE]	0.3680 ± 0.0006	0.3653 ± 0.0005
XGBoost	0.3792 ± 0.0002	0.3787 ± 0.0000
LightGBM	0.3688 ± 0.0002	0.3684 ± 0.0000
CatBoost	0.3647 ± 0.0005	0.3632 ± 0.0003
MNCA†	0.3536 ± 0.0052	0.3407 ± 0.0009
TabM	0.3866 ± 0.0006	0.3855 ± 0.0003
TabR†	0.3725 ± 0.0091	0.3497 ± 0.0031
TabM$^\dagger_{\text{mini}}$	0.3527 ± 0.0112	0.3478 ± 0.0009

particulate-matter-ukair-2017 ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.3759 ± 0.0012	0.3738 ± 0.0004
SNN	0.3790 ± 0.0007	0.3744 ± 0.0002
MLP	0.3759 ± 0.0004	0.3729 ± 0.0003
Trompt	$0.3724 \pm nan$	–
TabPFN	–	–
Excel	0.3706 ± 0.0011	0.3660 ± 0.0007
AutoINT	0.3723 ± 0.0011	0.3692 ± 0.0010
SAINT	0.3704 ± 0.0014	0.3672 ± 0.0009
FT−T	0.3735 ± 0.0012	0.3686 ± 0.0004
MLP−Mixer	0.3741 ± 0.0010	0.3698 ± 0.0004
T2G	0.3682 ± 0.0021	0.3635 ± 0.0006
TabR	0.3613 ± 0.0005	0.3590 ± 0.0002
MNCA	0.3670 ± 0.0004	0.3649 ± 0.0002
MLP†	0.3665 ± 0.0008	0.3642 ± 0.0003
MLP[PLE]	0.3649 ± 0.0011	0.3637 ± 0.0008
XGBoost	0.3641 ± 0.0001	0.3640 ± 0.0000
LightGBM	0.3637 ± 0.0001	0.3635 ± 0.0000
CatBoost	0.3647 ± 0.0004	0.3637 ± 0.0002
MNCA†	0.3646 ± 0.0001	0.3643 ± 0.0000
TabM	0.3686 ± 0.0006	0.3679 ± 0.0003
TabR†	0.3596 ± 0.0004	0.3579 ± 0.0002
TabM$^\dagger_{\text{mini}}$	0.3603 ± 0.0005	0.3589 ± 0.0003

road-safety ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.7781 ± 0.0014	0.7823 ± 0.0012
SNN	0.7847 ± 0.0010	0.7865 ± 0.0002
MLP	0.7857 ± 0.0019	0.7873 ± 0.0004
Trompt	$0.7823 \pm nan$	–
TabPFN	0.7338 ± 0.0032	–
Excel	0.7861 ± 0.0034	0.7902 ± 0.0009
AutoINT	0.7826 ± 0.0030	0.7883 ± 0.0013
SAINT	0.7584 ± 0.0584	0.7846 ± 0.0021
FT−T	0.7907 ± 0.0012	0.7943 ± 0.0007
MLP−Mixer	0.7878 ± 0.0032	0.7919 ± 0.0015
T2G	0.7917 ± 0.0026	0.7958 ± 0.0009
TabR	0.8403 ± 0.0014	0.8441 ± 0.0005
MNCA	0.8080 ± 0.0013	0.8121 ± 0.0006
MLP†	0.7867 ± 0.0018	0.7903 ± 0.0002
MLP[PLE]	0.7899 ± 0.0009	0.7935 ± 0.0003
XGBoost	0.8101 ± 0.0017	0.8129 ± 0.0004
LightGBM	0.7982 ± 0.0012	0.7996 ± 0.0005
CatBoost	0.8012 ± 0.0009	0.8022 ± 0.0002
MNCA†	0.8232 ± 0.0017	0.8287 ± 0.0008
TabM	0.7946 ± 0.0013	0.7961 ± 0.0005
TabR†	0.8374 ± 0.0013	0.8430 ± 0.0002
TabM$^\dagger_{\text{mini}}$	0.8015 ± 0.0034	0.8060 ± 0.0015

year ↓		
Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	9.2761 ± 0.0401	9.0640 ± 0.0156
SNN	9.0054 ± 0.0256	8.9351 ± 0.0073
MLP	8.9628 ± 0.0232	8.8931 ± 0.0066
Trompt	$9.1554 \pm nan$	–
TabPFN	–	–
Excel	9.0452 ± 0.0224	8.9612 ± 0.0129
AutoINT	9.0430 ± 0.0280	8.9619 ± 0.0092
SAINT	9.0248 ± 0.0225	8.9548 ± 0.0102
FT−T	9.0005 ± 0.0215	8.9360 ± 0.0013
MLP−Mixer	8.9589 ± 0.0182	8.9086 ± 0.0177
T2G	8.9762 ± 0.0160	8.8993 ± 0.0013
TabR	9.0069 ± 0.0152	8.9132 ± 0.0088
MNCA	8.9476 ± 0.0152	8.8977 ± 0.0037
MLP†	8.9355 ± 0.0103	8.9063 ± 0.0030
MLP[PLE]	8.9379 ± 0.0206	8.8753 ± 0.0038
XGBoost	9.0307 ± 0.0028	9.0245 ± 0.0015
LightGBM	9.0200 ± 0.0025	9.0128 ± 0.0015
CatBoost	9.0370 ± 0.0073	9.0054 ± 0.0028
MNCA†	8.8973 ± 0.0082	8.8550 ± 0.0031
TabM	8.8701 ± 0.0110	8.8517 ± 0.0022
TabR†	8.9721 ± 0.0105	8.9172 ± 0.0029
TabM$^\dagger_{\text{mini}}$	8.8825 ± 0.0087	8.8560 ± 0.0015

Table 20: Extended results for datasets from Table 6. Results are grouped by datasets.

sberbank-housing ↓		
Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.2616 ± 0.0049	0.2506 ± 0.0015
SNN	0.2671 ± 0.0140	0.2555 ± 0.0033
MLP	0.2529 ± 0.0078	0.2474 ± 0.0052
Trompt	–	–
TabPFN	–	–
Excel	0.2533 ± 0.0046	$0.2485 \pm nan$
AutoINT	–	–
SAINT	0.2467 ± 0.0019	$0.2442 \pm nan$
FT−T	0.2440 ± 0.0038	0.2367 ± 0.0010
MLP−Mixer	–	–
T2G	0.2416 ± 0.0025	–
TabR	0.2820 ± 0.0323	0.2603 ± 0.0048
MNCA	0.2593 ± 0.0053	0.2520 ± 0.0032
MLP†	0.2528 ± 0.0055	0.2503 ± 0.0029
MLP[PLE]	0.2383 ± 0.0032	0.2327 ± 0.0009
XGBoost	0.2419 ± 0.0012	0.2416 ± 0.0007
LightGBM	0.2468 ± 0.0009	0.2467 ± 0.0002
CatBoost	0.2482 ± 0.0034	0.2473 ± 0.0016
MNCA†	0.2448 ± 0.0039	0.2404 ± 0.0025
TabM	0.2469 ± 0.0035	0.2440 ± 0.0026
TabR†	0.2542 ± 0.0101	0.2448 ± 0.0021
TabM$^\dagger_{\text{mini}}$	0.2357 ± 0.0025	0.2333 ± 0.0007

ecom-offers ↑		
Method	Single model	Ensemble
Tuned Hyperparameters		
DCNv2	0.5996 ± 0.0043	0.6039 ± 0.0028
SNN	0.5912 ± 0.0056	0.5961 ± 0.0033
MLP	0.5989 ± 0.0017	0.5995 ± 0.0011
Trompt	–	–
TabPFN	–	–
Excel	0.5759 ± 0.0066	$0.5759 \pm nan$
AutoINT	–	–
SAINT	0.5812 ± 0.0098	$0.5834 \pm nan$
FT−T	0.5775 ± 0.0063	0.5817 ± 0.0021
MLP−Mixer	–	–
T2G	0.5791 ± 0.0056	–
TabR	0.5943 ± 0.0019	0.5977 ± 0.0009
MNCA	0.5765 ± 0.0087	0.5820 ± 0.0047
MLP†	0.5800 ± 0.0029	0.5819 ± 0.0011
MLP[PLE]	0.5949 ± 0.0013	0.5953 ± 0.0006
XGBoost	0.5763 ± 0.0072	0.5917 ± 0.0035
LightGBM	0.5758 ± 0.0006	0.5758 ± 0.0003
CatBoost	0.5596 ± 0.0068	0.5067 ± 0.0011
MNCA†	0.5758 ± 0.0050	0.5796 ± 0.0009
TabM	0.5948 ± 0.0006	0.5952 ± 0.0004
TabR†	0.5762 ± 0.0052	0.5794 ± 0.0008
TabM$^\dagger_{\text{mini}}$	0.5919 ± 0.0016	0.5926 ± 0.0006

maps-routing ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.1656 ± 0.0004	0.1636 ± 0.0001
SNN	0.1634 ± 0.0002	0.1625 ± 0.0000
MLP	0.1625 ± 0.0001	0.1621 ± 0.0000
Trompt	–	–
TabPFN	–	–
Excel	0.1628 ± 0.0001	$0.1621 \pm nan$
AutoINT	–	–
SAINT	$0.1634 \pm nan$	–
FT−T	0.1625 ± 0.0003	0.1619 ± 0.0001
MLP−Mixer	–	–
T2G	0.1616 ± 0.0001	–
TabR	0.1639 ± 0.0003	0.1622 ± 0.0002
MNCA	0.1625 ± 0.0001	0.1621 ± 0.0001
MLP†	0.1618 ± 0.0002	0.1613 ± 0.0000
MLP[PLE]	0.1620 ± 0.0002	0.1614 ± 0.0000
XGBoost	0.1616 ± 0.0001	0.1614 ± 0.0000
LightGBM	0.1618 ± 0.0000	0.1616 ± 0.0000
CatBoost	0.1619 ± 0.0001	0.1615 ± 0.0000
MNCA†	0.1627 ± 0.0002	0.1623 ± 0.0001
TabM	0.1612 ± 0.0001	0.1609 ± 0.0000
TabR†	0.1622 ± 0.0002	0.1614 ± 0.0000
TabM$^\dagger_{\text{mini}}$	0.1610 ± 0.0001	0.1607 ± 0.0001

homesite-insurance ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.9398 ± 0.0053	0.9432 ± 0.0018
SNN	0.9473 ± 0.0013	0.9484 ± 0.0007
MLP	0.9506 ± 0.0005	0.9514 ± 0.0001
Trompt	–	–
TabPFN	–	–
Excel	0.9622 ± 0.0004	$0.9635 \pm nan$
AutoINT	–	–
SAINT	$0.9613 \pm nan$	–
FT−T	0.9622 ± 0.0006	0.9633 ± 0.0001
MLP−Mixer	–	–
T2G	0.9624 ± 0.0006	–
TabR	0.9487 ± 0.0014	0.9505 ± 0.0001
MNCA	0.9514 ± 0.0038	0.9522 ± 0.0027
MLP†	0.9609 ± 0.0009	0.9626 ± 0.0003
MLP[PLE]	0.9582 ± 0.0014	0.9599 ± 0.0002
XGBoost	0.9601 ± 0.0002	0.9602 ± 0.0000
LightGBM	0.9603 ± 0.0002	0.9604 ± 0.0001
CatBoost	0.9606 ± 0.0003	0.9609 ± 0.0001
MNCA†	0.9620 ± 0.0006	0.9635 ± 0.0002
TabM	0.9641 ± 0.0004	0.9644 ± 0.0003
TabR†	0.9556 ± 0.0021	0.9600 ± 0.0008
TabM$^\dagger_{\text{mini}}$	0.9627 ± 0.0002	0.9630 ± 0.0001

cooking-time ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.4834 ± 0.0003	0.4822 ± 0.0001
SNN	0.4835 ± 0.0006	0.4818 ± 0.0002
MLP	0.4828 ± 0.0002	0.4822 ± 0.0000
Trompt	–	–
TabPFN	–	–
Excel	0.4821 ± 0.0005	$0.4808 \pm nan$
AutoINT	–	–
SAINT	$0.4840 \pm nan$	–
FT−T	0.4820 ± 0.0008	0.4813 ± 0.0005
MLP−Mixer	–	–
T2G	0.4809 ± 0.0008	–
TabR	0.4828 ± 0.0008	0.4814 ± 0.0004
MNCA	0.4825 ± 0.0004	0.4819 ± 0.0003
MLP†	0.4811 ± 0.0004	0.4805 ± 0.0001
MLP[PLE]	0.4812 ± 0.0004	0.4807 ± 0.0002
XGBoost	0.4823 ± 0.0001	0.4821 ± 0.0000
LightGBM	0.4826 ± 0.0001	0.4825 ± 0.0001
CatBoost	0.4823 ± 0.0001	0.4820 ± 0.0001
MNCA†	0.4818 ± 0.0005	0.4809 ± 0.0003
TabM	0.4803 ± 0.0006	0.4797 ± 0.0003
TabR†	0.4818 ± 0.0006	0.4807 ± 0.0000
TabM$^\dagger_{\text{mini}}$	0.4805 ± 0.0007	0.4795 ± 0.0003

homecredit-default ↑

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.8471 ± 0.0019	0.8549 ± 0.0002
SNN	0.8541 ± 0.0016	0.8569 ± 0.0010
MLP	0.8538 ± 0.0014	0.8566 ± 0.0005
Trompt	–	–
TabPFN	–	–
Excel	0.8513 ± 0.0024	$0.8564 \pm nan$
AutoINT	–	–
SAINT	$0.8377 \pm nan$	–
FT−T	0.8571 ± 0.0023	0.8611 ± 0.0013
MLP−Mixer	–	–
T2G	0.8595 ± 0.0009	–
TabR	0.8501 ± 0.0027	0.8548 ± 0.0003
MNCA	0.8531 ± 0.0018	0.8569 ± 0.0004
MLP†	0.8598 ± 0.0009	0.8607 ± 0.0003
MLP[PLE]	0.8568 ± 0.0039	0.8614 ± 0.0014
XGBoost	0.8670 ± 0.0005	0.8674 ± 0.0001
LightGBM	0.8664 ± 0.0004	0.8667 ± 0.0000
CatBoost	0.8621 ± 0.0007	0.8636 ± 0.0003
MNCA†	0.8544 ± 0.0033	0.8606 ± 0.0024
TabM	0.8583 ± 0.0010	0.8599 ± 0.0006
TabR†	0.8547 ± 0.0021	0.8602 ± 0.0002
TabM$^\dagger_{\text{mini}}$	0.8632 ± 0.0017	0.8656 ± 0.0003

delivery-eta ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	0.5516 ± 0.0014	0.5495 ± 0.0004
SNN	0.5495 ± 0.0008	0.5479 ± 0.0001
MLP	0.5493 ± 0.0007	0.5478 ± 0.0006
Trompt	–	–
TabPFN	–	–
Excel	0.5552 ± 0.0030	$0.5524 \pm nan$
AutoINT	–	–
SAINT	$0.5528 \pm nan$	–
FT−T	0.5542 ± 0.0026	0.5523 ± 0.0018
MLP−Mixer	–	–
T2G	0.5527 ± 0.0016	–
TabR	0.5514 ± 0.0024	0.5480 ± 0.0005
MNCA	0.5498 ± 0.0007	0.5488 ± 0.0002
MLP†	0.5521 ± 0.0014	0.5512 ± 0.0005
MLP[PLE]	0.5521 ± 0.0019	0.5511 ± 0.0007
XGBoost	0.5468 ± 0.0002	0.5463 ± 0.0001
LightGBM	0.5468 ± 0.0001	0.5465 ± 0.0000
CatBoost	0.5465 ± 0.0001	0.5461 ± 0.0000
MNCA†	0.5507 ± 0.0013	0.5494 ± 0.0006
TabM	0.5510 ± 0.0015	0.5504 ± 0.0004
TabR†	0.5520 ± 0.0015	0.5495 ± 0.0009
TabM$^\dagger_{\mathrm{mini}}$	0.5508 ± 0.0013	0.5497 ± 0.0003

weather ↓

Method	Single model	Ensemble
	Tuned Hyperparameters	
DCNv2	1.5606 ± 0.0057	1.5292 ± 0.0028
SNN	1.5280 ± 0.0085	1.5013 ± 0.0034
MLP	1.5378 ± 0.0054	1.5111 ± 0.0029
Trompt	–	–
TabPFN	–	–
Excel	1.5131 ± 0.0022	$1.4707 \pm nan$
AutoINT	–	–
SAINT	1.5097 ± 0.0045	–
FT−T	1.5104 ± 0.0097	1.4719 ± 0.0040
MLP−Mixer	–	–
T2G	1.4849 ± 0.0087	–
TabR	1.4666 ± 0.0039	1.4547 ± 0.0008
MNCA	1.5062 ± 0.0054	1.4822 ± 0.0013
MLP†	1.5170 ± 0.0040	1.4953 ± 0.0023
MLP[PLE]	1.5162 ± 0.0020	1.5066 ± 0.0008
XGBoost	1.4671 ± 0.0006	1.4629 ± 0.0002
LightGBM	1.4625 ± 0.0008	1.4581 ± 0.0003
CatBoost	1.4688 ± 0.0019	–
MNCA†	1.5008 ± 0.0034	1.4782 ± 0.0011
TabM	1.4786 ± 0.0039	1.4715 ± 0.0020
TabR†	1.4458 ± 0.0018	1.4362 ± 0.0013
TabM$^\dagger_{\mathrm{mini}}$	1.4709 ± 0.0047	1.4611 ± 0.0023