

Supplementary material

A Software and hardware

For most model-dataset pairs the workflow was as follows:

- tune the model on any suitable hardware
- evaluate the tuned model on one or more NVidia Tesla V100 32Gb

All the experiments were conducted under the same conditions in terms of software versions. For almost all experiments the used hardware can be found in the source code.

B Data

B.1 Datasets

Table 1: Datasets description

Name	Abbr	# Train	# Validation	# Test	# Num	# Cat	Task type	Batch size
California Housing	CA	13209	3303	4128	8	0	Regression	256
Adult	AD	26048	6513	16281	6	8	Binclass	256
Helena	HE	41724	10432	13040	27	0	Multiclass	512
Jannis	JA	53588	13398	16747	54	0	Multiclass	512
Higgs Small	HI	62752	15688	19610	28	0	Binclass	512
ALOI	AL	69120	17280	21600	128	0	Multiclass	512
Epsilon	EP	320000	80000	100000	2000	0	Binclass	1024
Year	YE	370972	92743	51630	90	0	Regression	1024
Covtype	CO	371847	92962	116203	54	0	Multiclass	1024
Yahoo	YA	473134	71083	165660	699	0	Regression	1024
Microsoft	MI	723412	235259	241521	136	0	Regression	1024

B.2 Preprocessing

For regression problems, we standardize the target values:

$$y_{new} = \frac{y_{old} - \text{mean}(y_{train})}{\text{std}(y_{train})} \quad (1)$$

The feature preprocessing for DL models is described in the main text. Note that we add noise from $\mathcal{N}(0, 1e-3)$ to train numerical features for calculating the parameters (quantiles) of the quantile preprocessing as a workaround for features with few distinct values (see the source code for the exact implementation). The preprocessing is then applied to *original* features. We do not preprocess features for GBDTs, since this family of algorithms is insensitive to feature shifts and scaling.

C Results for all algorithms on all datasets

To measure statistical significance in the main text and in the tables in this section, we use the one-sided Wilcoxon (1945) test with $p = 0.01$.

Table 2 and Table 3 report all results for all models on all datasets.

Table 2: Results for single models with standard deviations. For each dataset, top results for baseline neural networks are in **bold**, top results for baseline neural networks and FT-Transformer are in **blue**, the overall top results are in **red**. “Top” means “the gap between this result and the result with the best mean score is not statistically significant”. “d” stands for “default”. The remaining notation follows those from the main text.

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓
Baseline Neural Networks											
TabNet	0.510 \pm 7.6e-3	0.850 \pm 5.2e-3	0.378 \pm 1.7e-3	0.728 \pm 3.5e-3	0.719 \pm 1.7e-3	0.954 \pm 1.0e-3	0.8896 \pm 3.1e-3	8.909 \pm 2.3e-2	0.957 \pm 7.5e-3	0.823 \pm 9.2e-3	0.751 \pm 9.4e-4
SNN	0.493 \pm 4.6e-3	0.854 \pm 1.8e-3	0.373 \pm 2.8e-3	0.719 \pm 1.6e-3	0.722 \pm 2.2e-3	0.954 \pm 1.6e-3	0.8975 \pm 2.4e-4	8.895 \pm 1.9e-2	0.961 \pm 2.0e-3	0.761 \pm 5.3e-4	0.751 \pm 5.2e-4
AutoInt	0.474 \pm 3.3e-3	0.859 \pm 1.5e-3	0.372 \pm 2.5e-3	0.721 \pm 2.3e-3	0.725 \pm 1.7e-3	0.945 \pm 1.3e-3	0.8949 \pm 5.8e-4	8.882 \pm 3.3e-2	0.934 \pm 3.5e-3	0.768 \pm 1.1e-3	0.750 \pm 6.1e-4
GrowNet	0.487 \pm 7.1e-3	0.857 \pm 1.9e-3	—	—	0.722 \pm 1.6e-3	—	0.8970 \pm 5.7e-4	8.827 \pm 3.8e-2	—	—	0.751 \pm 4.7e-4
MLP	0.499 \pm 2.9e-3	0.852 \pm 1.9e-3	0.383 \pm 2.6e-3	0.719 \pm 1.3e-3	0.723 \pm 1.8e-3	0.954 \pm 1.4e-3	0.8977 \pm 4.1e-4	8.853 \pm 3.1e-2	0.962 \pm 1.1e-3	0.757 \pm 3.5e-4	0.747 \pm 3.3e-4
DCN2	0.484 \pm 2.4e-3	0.853 \pm 3.9e-3	0.385 \pm 3.0e-3	0.716 \pm 1.5e-3	0.723 \pm 1.3e-3	0.955 \pm 1.2e-3	0.8977 \pm 2.6e-4	8.890 \pm 2.8e-2	0.965 \pm 1.0e-3	0.757 \pm 1.9e-3	0.749 \pm 5.8e-4
NODE	0.464 \pm 1.5e-3	0.858 \pm 1.6e-3	0.359 \pm 2.0e-3	0.727 \pm 1.6e-3	0.726 \pm 1.3e-3	0.918 \pm 5.4e-3	0.8958 \pm 4.7e-4	8.784 \pm 1.6e-2	0.958 \pm 1.1e-3	0.753 \pm 2.5e-4	0.745 \pm 2.0e-4
ResNet	0.486 \pm 2.9e-3	0.854 \pm 1.7e-3	0.396 \pm 1.7e-3	0.728 \pm 1.5e-3	0.727 \pm 1.7e-3	0.963 \pm 7.5e-4	0.8969 \pm 4.4e-4	8.846 \pm 2.4e-2	0.964 \pm 1.1e-3	0.757 \pm 6.2e-4	0.748 \pm 3.1e-4

FT-Transformer

FT-Transformer _d	0.469 \pm 3.8e-3	0.857 \pm 1.1e-3	0.381 \pm 2.4e-3	0.725 \pm 2.3e-3	0.725 \pm 1.8e-3	0.953 \pm 1.1e-3	0.8959 \pm 4.9e-4	8.889 \pm 4.6e-2	0.967 \pm 7.9e-4	0.756 \pm 8.2e-4	0.747 \pm 7.9e-4
FT-Transformer	0.459 \pm 3.5e-3	0.859 \pm 1.0e-3	0.391 \pm 1.2e-3	0.732 \pm 2.0e-3	0.729 \pm 1.5e-3	0.960 \pm 1.1e-3	0.8982 \pm 2.8e-4	8.855 \pm 3.1e-2	0.970 \pm 6.6e-4	0.756 \pm 8.2e-4	0.746 \pm 4.9e-4
GBDT											
CatBoost _d	0.430 \pm 7.4e-4	0.873 \pm 9.6e-4	0.381 \pm 1.5e-3	0.721 \pm 1.1e-3	0.726 \pm 8.0e-4	0.946 \pm 9.3e-4	0.8880 \pm 4.5e-4	8.913 \pm 5.5e-3	0.908 \pm 2.4e-4	0.751 \pm 2.0e-4	0.745 \pm 2.3e-4
CatBoost	0.431 \pm 1.5e-3	0.873 \pm 1.2e-3	0.385 \pm 1.1e-3	0.728 \pm 1.5e-3	0.725 \pm 1.5e-3	—	0.8880 \pm 5.8e-4	8.877 \pm 6.0e-3	0.966 \pm 2.7e-4	0.743 \pm 2.4e-4	0.743 \pm 2.1e-4
XGBoost _d	0.462 \pm 0.0	0.874 \pm 0.0	0.348 \pm 0.0	0.711 \pm 0.0	0.717 \pm 0.0	0.924 \pm 0.0	0.8799 \pm 0.0	9.192 \pm 0.0	0.964 \pm 0.0	0.761 \pm 0.0	0.751 \pm 0.0
XGBoost	0.433 \pm 1.6e-3	0.872 \pm 4.6e-4	0.375 \pm 1.2e-3	0.721 \pm 1.0e-3	0.727 \pm 1.0e-3	—	0.8837 \pm 1.2e-3	8.947 \pm 8.5e-3	0.969 \pm 5.1e-4	0.736 \pm 2.1e-4	0.742 \pm 1.8e-4

Table 3: Results for ensembles with standard deviations. Notation follows Table 2.

	CA ↓	AD ↑	HE ↑	JA ↑	HI ↑	AL ↑	EP ↑	YE ↓	CO ↑	YA ↓	MI ↓
Baseline Neural Networks											
TabNet	0.488 \pm 1.8e-3	0.856 \pm 3.4e-4	0.391 \pm 3.1e-4	0.736 \pm 1.3e-3	0.727 \pm 1.3e-3	0.961 \pm 2.8e-4	0.8944 \pm 6.8e-4	8.728 \pm 8.0e-3	0.966 \pm 1.5e-3	0.815 \pm 3.4e-3	0.746 \pm 3.5e-4
SNN	0.478 \pm 1.0e-3	0.857 \pm 3.1e-4	0.380 \pm 1.2e-3	0.727 \pm 8.7e-4	0.729 \pm 2.2e-3	0.962 \pm 2.8e-4	0.8976 \pm 7.5e-5	8.759 \pm 1.4e-3	0.966 \pm 4.5e-4	0.754 \pm 4.0e-4	0.747 \pm 5.2e-4
AutoInt	0.459 \pm 3.7e-3	0.860 \pm 2.2e-4	0.382 \pm 3.7e-4	0.733 \pm 7.8e-4	0.732 \pm 6.6e-4	0.959 \pm 1.7e-4	0.8966 \pm 2.5e-4	8.736 \pm 3.0e-3	0.950 \pm 1.1e-3	0.758 \pm 1.7e-4	0.747 \pm 1.5e-4
GrowNet	0.468 \pm 1.4e-3	0.859 \pm 6.3e-4	—	—	0.730 \pm 4.1e-4	—	0.8978 \pm 1.5e-4	8.683 \pm 6.6e-3	—	—	0.747 \pm 1.4e-4
MLP	0.487 \pm 7.9e-4	0.855 \pm 4.8e-4	0.390 \pm 1.4e-3	0.725 \pm 2.1e-4	0.725 \pm 3.1e-3	0.960 \pm 3.2e-4	0.8979 \pm 1.1e-4	8.712 \pm 6.3e-3	0.966 \pm 9.1e-5	0.753 \pm 1.5e-4	0.746 \pm 1.4e-4
DCN2	0.477 \pm 3.7e-4	0.857 \pm 3.2e-4	0.388 \pm 1.5e-3	0.719 \pm 1.5e-3	0.725 \pm 1.0e-3	0.960 \pm 4.1e-4	0.8977 \pm 4.8e-5	8.800 \pm 9.9e-3	0.969 \pm 4.4e-4	0.752 \pm 7.7e-4	0.746 \pm 3.7e-4
NODE	0.461 \pm 6.9e-4	0.860 \pm 7.0e-4	0.361 \pm 7.9e-4	0.730 \pm 8.4e-4	0.727 \pm 9.1e-4	0.921 \pm 1.6e-3	0.8970 \pm 3.7e-4	8.716 \pm 3.1e-3	0.965 \pm 5.0e-4	0.750 \pm 2.1e-5	0.744 \pm 8.2e-5
ResNet	0.478 \pm 7.9e-4	0.857 \pm 4.3e-4	0.398 \pm 7.2e-4	0.734 \pm 1.3e-3	0.731 \pm 8.5e-4	0.966 \pm 4.9e-4	0.8976 \pm 2.7e-4	8.770 \pm 8.0e-3	0.967 \pm 6.7e-4	0.751 \pm 7.5e-5	0.745 \pm 1.9e-4
FT-Transformer											
FT-Transformer _d	0.454 \pm 1.1e-3	0.860 \pm 4.9e-4	0.395 \pm 9.4e-4	0.734 \pm 7.5e-4	0.731 \pm 8.0e-4	0.966 \pm 3.9e-4	0.8969 \pm 1.9e-4	8.727 \pm 1.6e-2	0.973 \pm 3.2e-4	0.747 \pm 3.8e-4	0.742 \pm 3.3e-4
FT-Transformer	0.448 \pm 7.5e-4	0.860 \pm 3.9e-4	0.398 \pm 4.3e-4	0.739 \pm 5.9e-4	0.731 \pm 7.7e-4	0.967 \pm 4.8e-4	0.8984 \pm 1.6e-4	8.751 \pm 9.4e-3	0.973 \pm 1.1e-4	0.747 \pm 3.8e-4	0.743 \pm 1.1e-4
GBDT											
CatBoost _d	0.428 \pm 4.5e-5	0.873 \pm 4.2e-4	0.386 \pm 1.0e-3	0.724 \pm 4.8e-4	0.728 \pm 7.4e-4	0.948 \pm 9.2e-4	0.8893 \pm 2.7e-4	8.885 \pm 1.9e-3	0.910 \pm 3.0e-4	0.749 \pm 1.1e-4	0.744 \pm 4.4e-5
CatBoost	0.423 \pm 8.9e-4	0.874 \pm 4.5e-4	0.388 \pm 2.7e-4	0.727 \pm 6.4e-4	0.729 \pm 1.6e-3	—	0.8898 \pm 7.7e-5	8.837 \pm 3.2e-3	0.968 \pm 2.2e-5	0.740 \pm 1.7e-4	0.741 \pm 7.8e-5
XGBoost _d	0.462 \pm 0.0	0.874 \pm 0.0	0.348 \pm 0.0	0.711 \pm 0.0	0.717 \pm 0.0	0.924 \pm 0.0	0.8799 \pm 0.0	9.192 \pm 0.0	0.964 \pm 0.0	0.761 \pm 0.0	0.751 \pm 0.0
XGBoost	0.431 \pm 3.6e-4	0.872 \pm 2.3e-4	0.377 \pm 7.6e-4	0.724 \pm 3.4e-4	0.728 \pm 5.3e-4	—	0.8861 \pm 1.6e-4	8.819 \pm 4.0e-3	0.969 \pm 1.9e-4	0.732 \pm 5.4e-5	0.742 \pm 1.8e-5

D Additional results

D.1 Training times

Table 4: Training times in seconds averaged over 15 runs.

	CA	AD	HE	JA	HI	AL	EP	YE	CO	YA	MI
ResNet	72	144	363	163	91	933	704	777	4026	923	1243
FT-Transformer	187	128	536	576	257	2864	934	1776	5050	12712	2857
Overhead	2.6x	0.9x	1.5x	3.5x	2.8x	3.1x	1.3x	2.3x	1.3x	13.8x	2.3x

For most experiments, training times can be found in the source code. In Table 4, we provide the comparison between ResNet and FT-Transformer in order to “visualize” the overhead introduced by FT-Transformer compared to the main “conventional” DL baseline. The big difference on the Yahoo dataset is expected because of the large number of features (700).

D.2 How tuning time budget affects performance?

In this section, we aim to answer the following questions:

- how does the relative performance of tuned models depends on tuning *time* budget?
- does the number of tuning iterations used in the main text allow models to reach most of their potential?

The first question is important for two main reasons. First, we have to make sure that longer tuning *times* of FT-Transformer (the number of tuning *iterations* is the same as for all other models) is not the reason of its strong performance. Second, we want to test FT-Transformer in the regime of low tuning time budget.

We consider four algorithms: XGBoost (as a fast GBDT implementation), MLP (as the fastest and simplest DL model), ResNet (as a stronger but slower DL model), FT-Transformer (as the strongest and the slowest DL model). We consider three datasets: California Housing, Adult, Higgs Small. On each dataset, for each algorithm, we run five independent (five random seeds) hyperparameter optimizations. Each run is constrained only by *time*. For each of the considered time budgets (15 minutes, 30 minutes, 1 hour, 2 hours, 3 hours, 4 hours, 5 hours, 6 hours), we pick the best model identified by Optuna on the validation set using no more than this time budget. Then, we report its performance and the number of Optuna iterations averaged over the five random seeds. The results are reported in Table 5. The takeaways are as follows:

- interestingly, FT-Transformer achieves good metrics just after several randomly sampled configurations (Optuna performs simple random sampling during the first 10 (default) iterations).
- FT-Transformer is slower to train, which is expected
- extended tuning (in terms of iterations) for other algorithms does not lead to any meaningful improvements

Table 5: Performance of tuned models with different tuning time budgets. Tuned model performance and the number of Optuna iterations (in parentheses) are reported (both metrics are averaged over five random seeds). Best results among DL models are in bold, overall best results are in bold red.

	0.25h	0.5h	1h	2h	3h	4h	5h	6h
California Housing								
XGBoost	0.437 (31)	0.436 (56)	0.434 (120)	0.433 (252)	0.433 (410)	0.432 (557)	0.433 (719)	0.432 (867)
MLP	0.503(16)	0.496(42)	0.493(103)	0.488(230)	0.489(349)	0.489(466)	0.488(596)	0.488(724)
ResNet	0.488(7)	0.487(15)	0.483(30)	0.481(64)	0.482(101)	0.482(131)	0.482(164)	0.484(197)
FT-Transformer	0.466 (4)	0.464 (9)	0.465 (20)	0.460 (47)	0.458 (74)	0.458 (99)	0.457 (124)	0.459 (153)
Adult								
XGBoost	0.871 (165)	0.873 (311)	0.872 (638)	0.872 (1296)	0.872 (1927)	0.872 (2478)	0.872 (2999)	0.872 (3500)
MLP	0.856(20)	0.857(37)	0.858(71)	0.857(130)	0.856(190)	0.856(247)	0.856(310)	0.856(375)
ResNet	0.856(8)	0.854(16)	0.854(32)	0.856(69)	0.855(105)	0.855(140)	0.856(174)	0.855(208)
FT-Transformer	0.861 (6)	0.860 (12)	0.859 (27)	0.859 (52)	0.860 (78)	0.860 (99)	0.860 (125)	0.860 (148)
Higgs Small								
XGBoost	0.725(88)	0.725(153)	0.724(291)	0.725(573)	0.725(823)	0.726(1069)	0.725(1318)	0.725(1559)
MLP	0.721(16)	0.720(29)	0.723(62)	0.722(137)	0.724(220)	0.723(300)	0.724(375)	0.724(447)
ResNet	0.724(8)	0.727(14)	0.727(32)	0.728(61)	0.728(84)	0.728(107)	0.728(132)	0.728(154)
FT-Transformer	0.727 (2)	0.729 (5)	0.728 (12)	0.728 (23)	0.729 (34)	0.729 (44)	0.730 (56)	0.729 (66)

E FT-Transformer

In this section, we formally describe the details of FT-Transformer its tuning and evaluation. Also, we share additional technical experience and observations that were not used for final results in the paper but may be of interest to researchers and practitioners.

E.1 Architecture

Formal definition.

$$\text{FT-Transformer}(x) = \text{Prediction}(\text{Block}(\dots(\text{Block}(\text{AppendCLS}(\text{FeatureTokenizer}(x))))))$$

$$\text{Block}(x) = \text{ResidualPreNorm}(\text{FFN}, \text{ResidualPreNorm}(\text{MHSA}, x))$$

$$\text{ResidualPreNorm}(\text{Module}, x) = x + \text{Dropout}(\text{Module}(\text{Norm}(x)))$$

$$\text{FFN}(x) = \text{Linear}(\text{Dropout}(\text{Activation}(\text{Linear}(x))))$$

We use LayerNorm (Ba et al., 2016) as the normalization. See the main text for the description of Prediction and FeatureTokenizer. For MHSA, we set $n_{heads} = 8$ and do not tune this parameter.

Activation. Throughout the whole paper we used the ReGLU activation, since it is reported to be superior to the usually used GELU activation (Narang et al., 2021; Shazeer, 2020). However, we did not observe strong difference between ReGLU and ReLU in preliminary experiments.

Dropout rates. We observed that the attention dropout is always beneficial and FFN-dropout is also usually set by the tuning process to some non-zero value. As for the final dropout of each residual branch, it is rarely set to non-zero values by the tuning process.

PreNorm vs PostNorm. We use the PreNorm variant of Transformer, i.e. normalizations are placed at the beginning of each residual branch. The PreNorm variant is known for better optimization properties as opposed to the original Transformer, which is a PostNorm-Transformer (Liu et al., 2020; Nguyen and Salazar, 2019; Wang et al., 2019). The latter one may produce better models in terms of target metrics (Liu et al., 2020), but it usually requires additional modifications to the model and/or the training process, such as learning rate warmup or complex initialization schemes (Huang et al., 2020; Liu et al., 2020). While the PostNorm variant can be an option for practitioners seeking for the best possible model, we use the PreNorm variant in order to keep the optimization simple and same

for all models. Note that in the PostNorm formulation the LayerNorm in the "Prediction" equation (see the section "FT-Transformer" in the main text) should be omitted.

E.2 The default configuration(s)

Table 6 describes the configuration of FT-Transformer referred to as "default" in the main text. Note that it includes hyperparameters for both the model and the optimization. In fact, the configuration is a result of an "educated guess" and we did not invest much resources in its tuning.

Table 6: Default FT-Transformer used in the main text.

Layer count	3	
Feature embedding size	192	
Head count	8	
Activation & FFN size factor	(ReLU, $4/3$)	
Attention dropout	0.2	
FFN dropout	0.1	
Residual dropout	0.0	
Initialization	Kaiming	(He et al., 2015a)
Parameter count	929K	The value is given for 100 numerical features
Optimizer	AdamW	
Learning rate	$1e-4$	
Weight decay	$1e-5$	0.0 for Feature Tokenizer, LayerNorm and biases

where "FFN size factor" is a ratio of the FFN's hidden size to the feature embedding size.

We also designed a heuristic scaling rule to produce "default" configurations with the number of layers from one to six. We applied it on the Epsilon and Yahoo datasets in order to reduce the number of tuning iterations. However, we did not dig into the topic and our scaling rule may be suboptimal, see Wies et al. (2021) for a theoretically sound scaling rule.

In Table 7, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019). For Epsilon, however, we iterated over several "default" configurations using a heuristic scaling rule, since the full tuning procedure turned out to be too time consuming. For Yahoo, we did not perform tuning at all, since the default configuration already performed well. In the main text, for FT-Transformer on Yahoo, we report the result of the default FT-Transformer.

Table 7: FT-Transformer hyperparameter space. Here (A) = {CA, AD, HE, JA, HI} and (B) = {AL, YE, CO, MI}

Parameter	(Datasets) Distribution
# Layers	(A) UniformInt[1, 4], (B) UniformInt[1, 6]
Feature embedding size	(A,B) UniformInt[64, 512]
Residual dropout	(A) {0, Uniform[0, 0.2]}, (B) Const(0.0)
Attention dropout	(A,B) Uniform[0, 0.5]
FFN dropout	(A,B) Uniform[0, 0.5]
FFN factor	(A) Uniform[$2/3$, $8/3$], (B) Const($4/3$)
Learning rate	(A) LogUniform[$1e-5$, $1e-3$], (B) LogUniform[$3e-5$, $3e-4$]
Weight decay	(A,B) LogUniform[$1e-6$, $1e-3$]
# Iterations	(A) 100, (B) 50

E.3 Training

On the Epsilon dataset, we scale FT-Transformer using the technique proposed by Wang et al. (2020) with the "headwise" sharing policy; we set the projection dimension to 128. We follow the popular

“transformers” library (Wolf et al., 2020) and do not apply weight decay to Feature Tokenizer, biases in linear layers and normalization layers.

F Models

In this section, we describe the implementation details for all models. See section E.1 for details on FT-Transformer.

F.1 ResNet

Architecture. The architecture is formally described in the main text.

We tested several configurations and observed measurable difference in performance between all of them. We found the ones with “clear main path” (i.e. with all normalizations (except the last one) placed only in residual branches as in He et al. (2016) or Wang et al. (2019)) to perform better. As expected, it is also easier for them to train deeper configurations. We found the block design inspired by Transformer (Vaswani et al., 2017) to perform better or on par with the one inspired by the ResNet from computer vision (He et al., 2015b).

We observed that in the “optimal” configurations (the result of the hyperparameter optimization process) the inner dropout rate (not the last one) of one block was usually set to higher values compared to the outer dropout rate. Moreover, the latter one was set to zero in many cases.

Implementation. Ours, see the source code.

In Table 8, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

Table 8: ResNet hyperparameter space. Here (A) = {CA, AD, HE, JA, HI, AL} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
# Layers	(A) UniformInt[1, 8], (B) UniformInt[1, 16]
Layer size	(A) UniformInt[64, 512], (B) UniformInt[64, 1024]
Hidden factor	(A,B) Uniform[1, 4]
Hidden dropout	(A,B) Uniform[0, 0.5]
Residual dropout	(A,B) {0, Uniform[0, 0.5]}
Learning rate	(A,B) LogUniform[1e-5, 1e-2]
Weight decay	(A,B) {0, LogUniform[1e-6, 1e-3]}
Category embedding size	({AD}) UniformInt[64, 512]
# Iterations	100

F.2 MLP

Architecture. The architecture is formally described in the main text.

Implementation. Ours, see the source code.

In Table 9, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019). Note that the size of the first and the last layers are tuned and set separately, while the size for “in-between” layers is the same for all of them.

F.3 XGBoost

Implementation. We fix and do not tune the following hyperparameters:

- `booster = "gbtree"`
- `early-stopping-rounds = 50`
- `n-estimators = 2000`

Table 9: MLP hyperparameter space. Here (A) = {CA, AD, HE, JA, HI, AL} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
# Layers	(A) UniformInt[1, 8], (B) UniformInt[1, 16]
Layer size	(A) UniformInt[1, 512], (B) UniformInt[1, 1024]
Dropout	(A,B) {0, Uniform[0, 0.5]}
Learning rate	(A,B) LogUniform[1e-5, 1e-2]
Weight decay	(A,B) {0, LogUniform[1e-6, 1e-3]}
Category embedding size	({AD}) UniformInt[64, 512]
# Iterations	100

In Table 10, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

Table 10: XGBoost hyperparameter space. Here (A) = {CA, AD, HE, JA, HI} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
Max depth	(A) UniformInt[3, 10], (B) UniformInt[6, 10]
Min child weight	(A,B) LogUniform[1e-8, 1e5]
Subsample	(A,B) Uniform[0.5, 1]
Learning rate	(A,B) LogUniform[1e-5, 1]
Col sample by level	(A,B) Uniform[0.5, 1]
Col sample by tree	(A,B) Uniform[0.5, 1]
Gamma	(A,B) {0, LogUniform[1e-8, 1e2]}
Lambda	(A,B) {0, LogUniform[1e-8, 1e2]}
Alpha	(A,B) {0, LogUniform[1e-8, 1e2]}
# Iterations	100

F.4 CatBoost

Implementation. We fix and do not tune the following hyperparameters:

- `early-stopping-rounds` = 50
- `od-pval` = 0.001
- `iterations` = 2000

In Table 11, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019). We set the `task_type` parameter to “GPU” (the tuning was unacceptably slow on CPU).

Evaluation. We set the `task_type` parameter to “CPU”, since for the used version of the CatBoost library it is crucial for performance in terms of target metrics.

F.5 SNN

Implementation. Ours, see the source code.

In Table 12, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

F.6 NODE

Implementation. We used the official implementation: <https://github.com/Qwicen/node>.

Table 11: CatBoost hyperparameter space. Here (A) = {CA, AD, HE, JA, HI} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
Max depth	(A) UniformInt[3, 10], (B) UniformInt[6, 10]
Learning rate	(A,B) LogUniform[1e-5, 1]
Bagging temperature	(A,B) Uniform[0, 1]
L2 leaf reg	(A,B) LogUniform[1, 10]
Leaf estimation iterations	(A,B) UniformInt[1, 10]
# Iterations	100

Table 12: SNN hyperparameter space. Here (A) = {CA, AD, HE, JA, HI, AL} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
# Layers	(A) UniformInt[2, 16], (B) UniformInt[2, 32]
Layer size	(A) UniformInt[1, 512], (B) UniformInt[1, 1024]
Dropout	(A,B) {0, Uniform[0, 0.1]}
Learning rate	(A,B) LogUniform[1e-5, 1e-2]
Weight decay	(A,B) {0, LogUniform[1e-5, 1e-3]}
Category embedding size	({AD}) UniformInt[64, 512]
# Iterations	100

Tuning. We iterated over the parameter grid from the original paper (Popov et al., 2020) plus the default configuration from the original paper. For multiclass datasets, we set the tree dimension being equal to the number of classes. For the Helena and ALOI datasets there was no tuning since NODE does not scale to classification problems with a large number of classes (for example, the minimal non-default configuration of NODE contains 600M+ parameters on the Helena dataset), so the reported results for these datasets are obtained with the default configuration.

F.7 TabNet

Implementation. We used the official implementation:

<https://github.com/google-research/google-research/tree/master/tabnet>.

We always set `feature-dim` equal to `output-dim`. We also fix and do not tune the following hyperparameters (let $A = \{CA, AD\}$, $B = \{HE, JA, HI, AL\}$, $C = \{EP, YE, CO, YA, MI\}$):

- `virtual-batch-size` = (A) 2048, (B) 8192, (C) 16384
- `batch-size` = (A) 256, (B) 512, (C) 1024

In Table 13, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

F.8 GrowNet

Implementation. We used the official implementation: <https://github.com/sbadirli/GrowNet>. Note that it does not support multiclass problems, hence the gaps in the main tables for multiclass problems. We use no more than 40 small MLPs, each MLP has 2 hidden layers, boosting rate is learned – as suggested by the authors.

In Table 14, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

Table 13: TabNet hyperparameter space.

Parameter	Distribution
# Decision steps	UniformInt[3, 10]
Layer size	{8, 16, 32, 64, 128}
Relaxation factor	Uniform[1, 2]
Sparsity loss weight	LogUniform[1e-6, 1e-1]
Decay rate	Uniform[0.4, 0.95]
Decay steps	{100, 500, 2000}
Learning rate	Uniform[1e-3, 1e-2]
# Iterations	100

Table 14: GrowNet hyperparameter space.

Parameter	(Datasets) Distribution
Correct epochs	(all) {1, 2}
Epochs per stage	(all) {1, 2}
Hidden dimension	(all) UniformInt[32, 512]
Learning rate	(all) LogUniform[1e-5, 1e-2]
Weight decay	(all) {0, LogUniform[1e-6, 1e-3]}
Category embedding size	({AD}) UniformInt[32, 512]
# Iterations	100

F.9 DCN V2

Architecture. There are two variants of DCN V2, namely, “stacked” and “parallel”. We tuned and evaluated both and did not observe strong superiority of any of them. We report numbers for the “parallel” variant as it was slightly better on large datasets.

Implementation. Ours, see the source code.

In Table 15, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

Table 15: DCN V2 hyperparameter space. Here (A) = {CA, AD, HE, JA, HI, AL} and (B) = {EP, YE, CO, YA, MI}

Parameter	(Datasets) Distribution
# Cross layers	(A) UniformInt[1, 8], (B) UniformInt[1, 16]
# Hidden layers	(A) UniformInt[1, 8], (B) UniformInt[1, 16]
Layer size	(A) UniformInt[64, 512], (B) UniformInt[64, 1024]
Hidden dropout	(A,B) Uniform[0, 0.5]
Cross dropout	(A,B) {0, Uniform[0, 0.5]}
Learning rate	(A,B) LogUniform[1e-5, 1e-2]
Weight decay	(A,B) {0, LogUniform[1e-6, 1e-3]}
Category embedding size	({AD}) UniformInt[64, 512]
# Iterations	100

F.10 AutoInt

Implementation. Ours, see the source code. We mostly follow the original paper (Song et al., 2019), however, it turns out to be necessary to introduce some modifications such as normalization in order to make the model competitive. We fix $n_{heads} = 2$ as recommended in the original paper.

In Table 16, we provide hyperparameter space used for Optuna-driven tuning (Akiba et al., 2019).

Table 16: AutoInt hyperparameter space. Here (A) = {CA, AD, HE, JA, HI} and (B) = {AL, YE, CO, MI}

Parameter	(Datasets) Distribution
# Layers	(A,B) UniformInt[1, 6]
Feature embedding size	(A,B) UniformInt[8, 64]
Residual dropout	(A) {0, Uniform[0.0, 0.2]}, (B) Const(0.0)
Attention dropout	(A,B) Uniform[0.0, 0.5]
Learning rate	(A) LogUniform[1e-5, 1e-3], (B) LogUniform[3e-5, 3e-4]
Weight decay	(A,B) LogUniform[1e-6, 1e-3]
# Iterations	(A) 100, (B) 50

G Analysis

G.1 When FT-Transformer is better than ResNet?

Data. Train, validation and test set sizes are 500 000, 50 000 and 100 000 respectively. One object is generated as $x \sim \mathcal{N}(0, I_{100})$. For each object, the first 50 features are used for target generation and the remaining 50 features play the role of “noise”.

f_{DL} . The function is implemented as an MLP with three hidden layers, each of size 256. Weights are initialized with Kaiming initialization (He et al., 2015a), biases are initialized with the uniform distribution $\mathcal{U}(-a, a)$, where $a = d_{input}^{-0.5}$. All the parameters are fixed after initialization and are not trained.

f_{GBDT} . The function is implemented as an average prediction of 30 randomly constructed decision trees. The construction of one random decision tree is demonstrated in algorithm 1. The inference process for one decision tree is the same as for ordinary decision trees.

CatBoost. We use the default hyperparameters.

FT-Transformer. We use the default hyperparameters. Parameter count: 930K.

ResNet. Residual block count: 4. Embedding size: 256. Dropout rate inside residual blocks: 0.5. Parameter count: 820K.

G.2 Ablation study

Table 17 is a more detailed version of the corresponding table from the main text.

Table 17: The results of the comparison between FT-Transformer and two attention-based alternatives. Means and standard deviations over 15 runs are reported

	CA ↓	HE ↑	JA ↑	HI ↑	AL ↑	YE ↓	CO ↑	MI ↓
AutoInt	0.474±3.3e-3	0.372±2.5e-3	0.721±2.3e-3	0.725±1.7e-3	0.945±1.3e-3	8.882±3.3e-2	0.934±3.5e-3	0.750±6.1e-4
FT-Transformer (w/o feature biases)	0.470±5.7e-3	0.381±1.6e-3	0.724±3.9e-3	0.727±1.9e-3	0.958±1.2e-3	8.843±2.5e-2	0.964±6.2e-4	0.751±5.6e-4
FT-Transformer	0.459±3.5e-3	0.391±1.2e-3	0.732±2.0e-3	0.729±1.5e-3	0.960±1.1e-3	8.855±3.1e-2	0.970±6.6e-4	0.746±4.9e-4

Algorithm 1: Construction of one random decision tree.

Result: Random Decision Tree

```
set of leaves  $L = \{\text{root}\}$ ;  
depths - mapping from nodes to their depths;  
left - mapping from nodes to their left children;  
right - mapping from nodes to their right children;  
features - mapping from nodes to splitting features;  
thresholds - mapping from nodes to splitting thresholds;  
values - mapping from leaves to their associated values;  
 $n = 0$  - number of nodes;  
 $k = 100$  - number of features;  
while  $n < 100$  do  
  randomly choose leaf  $z$  from  $L$  s.t.  $\text{depths}[z] < 10$ ;  
   $\text{features}[z] \sim \text{UniformInt}[1, \dots, k]$ ;  
   $\text{thresholds}[z] \sim \mathcal{N}(0, 1)$ ;  
  add two new nodes  $l$  and  $r$  to  $L$ ;  
  remove  $z$  from  $L$ ;  
  unset  $\text{values}[z]$ ;  
   $\text{left}[z] = l$ ;  
   $\text{right}[z] = r$ ;  
   $\text{depths}[l] = \text{depths}[r] = \text{depths}[z] + 1$ ;  
   $\text{values}[l] \sim \mathcal{N}(0, 1)$ ;  
   $\text{values}[r] \sim \mathcal{N}(0, 1)$ ;  
   $n = n + 2$ ;  
end  
return Random Decision Tree as  $\{L, \text{left}, \text{right}, \text{features}, \text{thresholds}, \text{values}\}$ .
```

H Additional datasets

Here, we report results for some datasets that turned out to be non-informative benchmarks, that is, where all models perform similarly. We report the average results over 15 random seeds for single models that are tuned and trained under the same protocol as described in the main text. The datasets include Bank (Moro et al., 2014), Kick¹, MiniBooNe², Click³. The dataset properties are given in Table 18 and the results are reported in Table 19.

Table 18: Additional datasets

Dataset	# objects	# Num	# Cat	Task type (metric)
Bank	45211	7	9	Binclass (accuracy)
Kick	72983	14	18	Binclass (accuracy)
MiniBooNe	130064	50	0	Binclass (accuracy)
Click	1000000	3	8	Binclass (accuracy)

¹<https://www.kaggle.com/c/DontGetKicked>

²<https://archive.ics.uci.edu/ml/datasets/MiniBooNE+particle+identification>

³<http://www.kdd.org/kdd-cup/view/kdd-cup-2012-track-2>

Table 19: Results for single models on additional datasets.

	Bank	Kick	MiniBooNE	Click
SNN	0.9076 (0.0016)	0.9014 (0.0007)	0.9493 (0.0006)	0.6613 (0.0006)
Grownnet	0.9093 (0.0012)	0.9016 (0.0006)	0.9494 (0.0007)	0.6614 (0.0009)
DCNv2	0.9085 (0.0010)	0.9014 (0.0007)	0.9496 (0.0005)	0.6615 (0.0003)
AutoInt	0.9065 (0.0014)	0.9005 (0.0005)	0.9478 (0.0008)	0.6614 (0.0005)
MLP	0.9059 (0.0014)	0.9012 (0.0004)	0.9501 (0.0006)	0.6617 (0.0006)
ResNet	0.9072 (0.0014)	0.9017 (0.0005)	0.9508 (0.0006)	0.6612 (0.0007)
FT-Transformer	0.9090 (0.0014)	0.9016 (0.0003)	0.9491 (0.0007)	0.6606 (0.0009)
FT-Transformer (default)	0.9088 (0.0013)	0.9013 (0.0006)	0.9476 (0.0007)	0.6610 (0.0007)
CatBoost	0.9068 (0.0015)	0.9021 (0.0009)	0.9465 (0.0005)	0.6635 (0.0002)
XgBoost	0.9087 (0.0009)	0.9034 (0.0003)	0.9461 (0.0005)	0.6399 (0.0006)

References

- T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *KDD*, 2019.
- J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv*, 1607.06450v1, 2016.
- K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015a.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv*, 1512.03385v1, 2015b.
- K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.
- X. S. Huang, F. Perez, J. Ba, and M. Volkovs. Improving transformer optimization through better initialization. In *ICML*, 2020.
- L. Liu, X. Liu, J. Gao, W. Chen, and J. Han. Understanding the difficulty of training transformers. In *EMNLP*, 2020.
- S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decis. Support Syst.*, 62:22–31, 2014.
- S. Narang, H. W. Chung, Y. Tay, W. Fedus, T. Fevry, M. Matena, K. Malkan, N. Fiedel, N. Shazeer, Z. Lan, Y. Zhou, W. Li, N. Ding, J. Marcus, A. Roberts, and C. Raffel. Do transformer modifications transfer across implementations and applications? *arXiv*, 2102.11972v1, 2021.
- T. Q. Nguyen and J. Salazar. Transformers without tears: Improving the normalization of self-attention. In *IWSLT*, 2019.
- S. Popov, S. Morozov, and A. Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *ICLR*, 2020.
- N. Shazeer. Glu variants improve transformer. *arXiv*, 2002.05202v1, 2020.
- W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang. AutoInt: Automatic feature interaction learning via self-attentive neural networks. In *CIKM*, 2019.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.
- Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao. Learning deep transformer models for machine translation. In *ACL*, 2019.
- S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. Linformer: Self-attention with linear complexity. *arXiv*, 2006.04768v3, 2020.

- N. Wies, Y. Levine, D. Jannai, and A. Shashua. Which transformer architecture fits my data? a vocabulary bottleneck in self-attention. In *ICLM*, 2021.
- F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80, 1945.
- T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface’s transformers: State-of-the-art natural language processing. *arXiv*, 1910.03771v5, 2020.