DOFEN: Deep Oblivious Forest ENsemble

Kuan-Yu Chen
Sinopac Holdings
lavamore@sinopac.com

Ping-Han Chiang
Sinopac Holdings
u10000129@gmail.com

Hsin-Rung Chou
Sinopac Holdings
sherry.chou@sinopac.com

Chih-Sheng Chen
Sinopac Holdings
sheng77@sinopac.com

Darby Tien-Hao Chang
Sinopac Holdings
National Cheng Kung University
darby@sinopac.com

Abstract

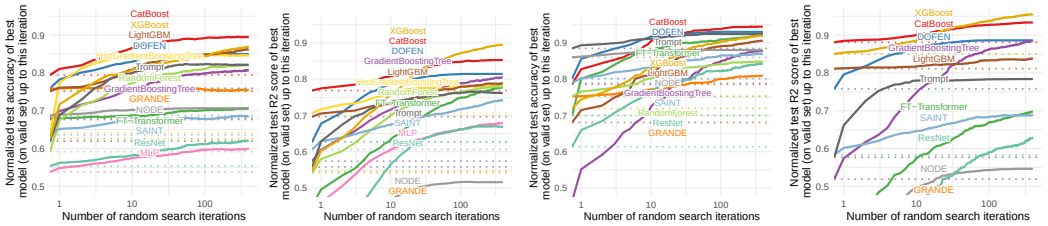
Deep Neural Networks (DNNs) have revolutionized artificial intelligence, achieving impressive results on diverse data types, including images, videos, and texts. However, DNNs still lag behind Gradient Boosting Decision Trees (GBDT) on tabular data, a format extensively utilized across various domains. This paper introduces DOFEN, which stands for **D**eep **O**blivious **F**orest **EN**semble. DOFEN is a novel DNN architecture inspired by oblivious decision trees and achieves on-off sparse selection of columns. DOFEN surpasses other DNNs on tabular data, achieving state-of-the-art performance on the well-recognized benchmark: Tabular Benchmark [1], which includes 73 total datasets spanning a wide array of domains. The code of DOFEN is available at: `https://github.com/Sinopac-Digital-Technology-Division/DOFEN`.

1 Introduction

Tree-based models, including RandomForest [2], Extra Trees [3], and Gradient Boosting Decision Tree (GBDT) frameworks such as XGBoost [4], LightGBM [5], and CatBoost [6], are widely recognized for their simplicity, efficiency, and remarkable performance with tabular data. This has inspired numerous studies investigating the integration of tree-based algorithms with deep neural networks (DNNs), leading to tree-inspired DNNs such as Deep Forest [7], NODE [8], and TabNet [9]. In another line of tabular DNN research, novel DNN architectures such as SAINT [10], FT-Transformer [11], and Trompt [12] have been proposed. These novel architectures, which are essentially attention-based, demonstrate better performance compared with tree-inspired DNNs but require significantly more time and space. While these tabular DNNs have shown promising performance in specific contexts, recent surveys and benchmarks generally indicate that they do not surpass the performance of GBDTs on tabular data [1, 11, 13–15].

Hence, we begin by questioning what's missing in existing tabular DNNs and identify one key difference: in tree-based models, only a limited number of features are used in the construction of each tree. This concept of "sparse selection of columns" not only increases feature diversity but also helps mitigate overfitting in tree-based models [2–6]. However, existing tabular DNNs are unable to achieve a sparse selection. For example, attention-based models [10–12] use the softmax operation to aggregate column information, resulting in a "dense selection" across columns. Some tree-inspired DNNs [8, 9] have utilized methods like entmax and sparsemax [16, 17] to enhance sparsity, but they can still only achieve near-sparse effects. Therefore, we opt to develop a new approach to achieve this characteristic.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).



(a) Medium, Classification. (b) Medium, Regression. (c) Large, Classification. (d) Large, Regression.

Figure 1: Evaluation results on the Tabular Benchmark. The model names are sorted by their performances at the end of the random search of hyperparameters. The result are averaged over various datasets included in each benchmark respectively, detailed number of datasets of each benchmark is provided in Appendix B.1

For deep learning models, the biggest challenge is that generating a sparse matrix for on-off column selection is non-differentiable. In this study, we propose a novel two-step process to work around this issue: (1) enumerating as many sparse selections of columns as possible, and (2) weighting the importance of these sparse selections, making the weights differentiable and trainable by a DNN model. We name this new tree-inspired DNN **DOFEN**, an abbreviation for **D**eep **O**blivious **F**orest **EN**semble, and further demonstrate how DOFEN implements these two steps below:

1. **Condition Generation and rODT Construction.** In DOFEN, the step of enumerating sparse selections is further divided into two parts. The first part generates conditions, each involving exactly one column and corresponding to a decision rule of a tree node, as described in Section 3.2.1. The second part combines conditions using a shuffle-then-reshape procedure, detailed in Section 3.2.2. Each resultant combination of conditions can be seen as a differentiable counterpart to the Oblivious Decision Tree (ODT) [18], referred to as a relaxed ODT (rODT) in the context. Consequently, all the combinations collectively form a pool of rODTs.

2. **Two-level rODT Ensemble.** To ensure that the weighting of a limited number of rODTs can achieve good predictive performance, the previous step requires a sufficiently large pool. However, assembling all the rODTs in the pool into a single giant forest tends to cause overfitting, as shown in Appendix F.1 and Figure 9. Thus, DOFEN implements the step of importance weighting using a two-level ensemble procedure. The first level involves ensembling only a randomly selected subset of the rODT pool to form individual rODT forests, which is similar to applying dropout [19] to the rODT pool. The second level treats each rODT forest as a weak learner and aggregates them into a forest ensemble. This level is designed to enhance performance and stability, similar to standard ensemble learning. Both level of ensemble is detailed in Algorithm 1 of Section 3.2.3.

To evaluate DOFEN comprehensively and objectively, we have chosen a recent and well-recognized benchmark: the Tabular Benchmark [1]. This benchmark addresses the issue of inconsistent dataset selection, which is prevalent in deep learning research on tabular data. It includes a variety of regression and classification datasets with standardized feature processing for consistency. Additionally, we have conducted detailed analyses focusing on the distinct features of DOFEN, thereby offering insights into its functionalities. In summary, our research makes two key contributions:

1. **Innovative Neural Network Architecture.** The DOFEN model is fundamentally inspired by ODTs and incorporates an innovative two-step process to achieve on-off sparse selection of columns. This unique approach enhances performance beyond that of current tree-inspired DNNs and offers differentiability compared to conventional tree-based models.

2. **State-of-the-Art Performance.** The DOFEN model exhibits outstanding performance, surpassing that of other neural network models and competing closely with GBDTs on the Tabular Benchmark. This achievement underscores its robustness and versatility across various tasks, as illustrated in Figure 1.

2 Related Work

In this section, we start by exploring ODT and detail our rationale for selecting ODT as the foundational element in our study. We then systematically categorize deep tabular neural networks into two distinct streams: tree-inspired DNN architectures and novel DNN architectures. Through comparing DOFEN with these established models, our goal is to highlight its unique contributions and position it within the broad landscape of deep tabular network research.

Oblivious Decision Tree. The ODT is a variant of the traditional decision tree algorithm [20], which makes a series of feature-based decisions along its root-to-leaf path to deliver a prediction. In the context, a feature-based decision rule, e.g. $age > 18$, is called a condition. The traditional decision tree algorithm [18] chooses different conditions on different nodes, while in ODT, all nodes at the same level apply the same condition, resulting in a more uniform decision-making process. This uniformity allows for streamlined and vectorized decision-making, thus enhancing computational efficiency, while it also comes at the cost of capacity [8]. However, studies have shown that ensembles of ODTs can achieve remarkable performance with sufficient capacity [6, 8]. In this research, we integrate ODTs as the foundational element in the DOFEN model and capitalize on the strengths of ODTs while mitigating their limitations through ensemble strategies and deep learning techniques.

Tree-inspired DNN Architectures. Integrating decision tree (DT) algorithms with DNNs has become prominent for handling tabular data. Pioneering works like Deep Forest [7], NODE [8], TabNet [9], GradTree [21] and GRANDE [22] have each introduced unique methodologies.

Deep Forest adapts the random forest algorithm and incorporates multi-grained feature scanning to leverage the representation learning capabilities of DNNs. TabNet models the sequential decision-making process of traditional decision trees using a DNN, featuring a distinct encoder-decoder architecture that enables self-supervised learning. GradTree recognizes the importance of hard, axis-aligned splits for tabular data and uses a straight-through operator to handle the non-differentiable nature of decision trees, allowing for the end-to-end training of decision trees. NODE and GRANDE share a similar observation and high-level structure to DOFEN, in that they ensemble multiple tree-like deep learning base models. NODE uses ODT as a base predictor and employs a DenseNet-like multi-layer ensemble to boost performance. GRANDE, a successor to GradTree, uses DT as a base predictor and introduces advanced instance-wise weighting for ensembling each base model's prediction.

However, DOFEN distinguishes itself from NODE and GRANDE through its unique architectural design. First, DOFEN employs a different approach to transform tree-based models into neural networks. Unlike NODE and GRANDE, which explicitly learn the decision paths (i.e., selecting features and thresholds for each node) and the leaf node values of a tree, DOFEN randomly selects features to form rODTs and uses a neural network to measure how well a sample aligns with the decision rule. Additionally, the leaf node value of an rODT is replaced with an embedding vector for further ensembling. Second, DOFEN introduces a novel two-level ensemble process to enhance model performance and stability. Unlike NODE and GRANDE, which simply perform a weighted sum on base model predictions, DOFEN first constructs multiple rODT forests by randomly aggregating selected rODT embeddings and then applies bagging on the predictions of these rODT forests.

Novel DNN Architectures. Beyond merging decision tree algorithms with DNNs, significant progress has been made in developing novel architectures for tabular data. Notable among these are TabTransformer [23], FT-Transformer [11], SAINT [10], TabPFN [24], and Trompt [12]. These models primarily leverage the transformer architecture [25], utilizing self-attention mechanisms to capture complex feature relationships.

TabTransformer applies transformer blocks specifically to numerical features, while FT-Transformer extends this approach to both numerical and categorical features. SAINT enhances the model further by applying self-attention both column-wise and sample-wise, increasing its capacity. TabPFN, a variant of the Prior Fitted Network (PFN) [26], is particularly effective with smaller datasets. Trompt introduces an innovative approach by incorporating prompt learning techniques from natural language processing [27], aiming to extract deeper insights from the tabular data's columnar structure.

These models have demonstrated impressive performance across various studies and benchmarks. As a result, we choose them as our baselines to offer a comprehensive evaluation for deep learning models on tabular data.

3 DOFEN: Deep Oblivious Forest Ensemble

In this section, we begin with discussion about how DOFEN relax an ODT to be differentiable in Section 3.1, and elaborate on the details of the overall architecture design in Section 3.2. In the following figures and equations, three sub-networks—composites of fundamental neural network layers such as linear layers, layer normalization, and dropout—are simplified into symbols Δ_1, Δ_2, and Δ_3 for readability. The detailed configurations of these sub-networks can be found in Appendix A.2.

3.1 ODT Relaxation

An ODT operates on an input vector $\vec{x}$, where $\vec{x} \in \mathbb{R}^{N_{\text{col}}}$ and N_{col} is the number of columns in a tabular dataset, as described in Equation (1). Although these columns can be either numerical or categorical, we focus on real numbers in Equations (2) and (3) to simplify the notations.

$$\vec{x} = (x_i \mid i = 1, 2, \ldots, N_{\text{col}}), x_i \in \mathbb{R} \tag{1}$$

Fundamentally, an ODT of depth d is a decision table consisting of d entries [28], as depicted in Equation (2). Here, I_j indicates the index of a selected column, and x_{I_j} denotes its column value at depth j. The corresponding threshold is denoted by b_j, and H denotes the Heaviside function. In practice, the choice of x_{I_j} is decided by a predefined criterion, e.g., entropy or Gini impurity. It is possible for a raw column to be selected multiple times at different depths, each with a varying threshold.

$$\begin{aligned} \text{ODT}(\vec{x}) = \{H(x_{I_j} - b_j)\},\ \vec{x} \xrightarrow{\text{decided by entropy, Gini impurity, etc.}} \{(x_{I_j}, b_j)\}, \\ I_j \in \{1, 2, \ldots, N_{\text{col}}\},\ b_j \in \mathbb{R},\ j = (1, 2, \ldots, d) \end{aligned} \tag{2}$$

Equation (2) involves non-differentiable calculations, including the Heaviside function and the predefined criterion. Consequently, the key to integrating an ODT within a neural network model lies in making the following operations differentiable: selecting columns, deciding thresholds, and modeling H.

To address these challenges, DOFEN proposes a method to relax an ODT, as shown in Equation (3). In DOFEN, the columns of an ODT at different depths are selected randomly. The thresholds and the Heaviside function for column I_j are replaced with a sub-network Δ_{1I_j}, which employs the sigmoid activation function to create soft conditions. To avoid confusion, we introduce a new term, relaxed ODT (rODT), in this context. This term distinguishes between the original ODT and the relaxed version proposed in this study, which can be integrated to neural networks.

$$\begin{aligned} \text{rODT}(\vec{x}) = \{\Delta_{1I_j}(x_{I_j})\},\ \vec{x} \xrightarrow{\text{randomly select}} \{x_{I_j}\}, \\ I_j \in \{1, 2, \ldots, N_{\text{col}}\},\ j = (1, 2, \ldots, d) \end{aligned} \tag{3}$$

3.2 DOFEN Model

3.2.1 Condition Generation

This module transforms input vector $\vec{x}$ into multiple soft conditions for subsequent modules. The raw input in tabular data comprises a combination of numerical and categorical columns. In this study, a soft condition is defined as a scalar indicating how well a column adheres to a decision rule.

This transformation process creates a matrix $\mathbf{M}$, as shown in Equation (4), where N_{cond} is a hyper-parameter denoting the number of conditions we aim to generate for each column. Notably, each column x_i is processed by individual sub-network Δ_{1i} in this context, where $i \in \{1, \ldots, N_{\text{col}}\}$. This design is derived from the original ODT, where each condition involves only a single column. The sub-network Δ_1 is an embedding layer for a categorical column or a linear layer for a numerical column. Further details of Δ_1 can be found in Appendix A.2. As depicted in Figure 2a, three instances of Δ_1 generate four conditions for each column, resulting in a 3×4 matrix.

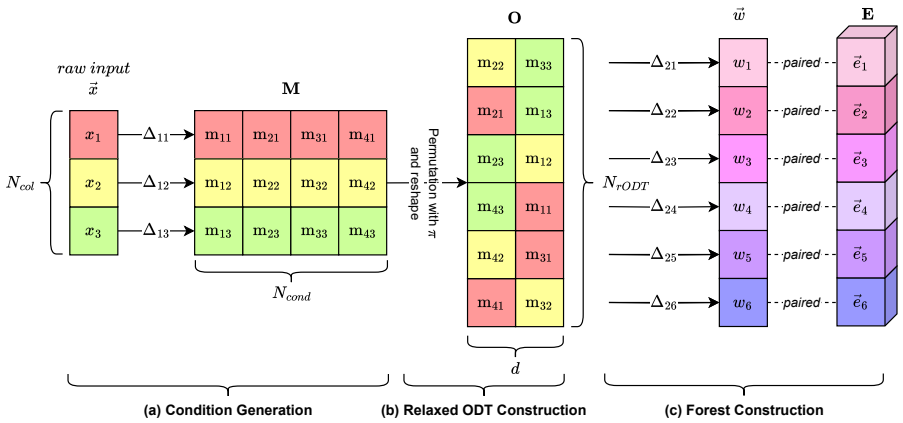


Figure 2: (a) Condition Generation: For each column x_i, N_{cond} conditions are generated through an individual sub-network Δ_{1i}. The aggregate of the conditions of all columns is denoted by the matrix $\mathbf{M}$. (b) Relaxed ODT Construction: The condition matrix $\mathbf{M}$ is shuffled (i.e. permutation with π) and reshape into $\mathbf{O}$, representing N_{rODT} rODTs each with depth d. (c) Forest Construction: To compute the weights w_i, an individual sub-networks Δ_{2i} is applied to each rODT. In addition, each w_i is paired with a learnable embedding vector $\mathbf{e}_i$. The aggregate of all weights and their corresponding embedding vectors are denoted as $\vec{w}$ and $\mathbf{E}$, respectively.

$$\mathbf{M} = \begin{bmatrix} m_{11} & \dots & m_{1N_{\text{col}}} \\ \vdots & \ddots & \vdots \\ m_{N_{\text{cond}}1} & \dots & m_{N_{\text{cond}}N_{\text{col}}} \end{bmatrix} \in \mathbb{R}^{N_{\text{cond}} \times N_{\text{col}}}, (m_{i1}, \dots, m_{iN_{\text{cond}}}) = \Delta_{1i}(x_i), i = (1, 2, \dots, N_{\text{col}}) \tag{4}$$

3.2.2 Relaxed ODT Construction

This module constructs multiple rODTs. Unlike traditional ODT, which selects columns and their corresponding thresholds based on predefined criteria, DOFEN randomly selects d elements from the $N_{\text{cond}} \times N_{\text{col}}$ conditions in matrix $\mathbf{M}$ without replacement to build an rODT with depth d. In our implementation, $\mathbf{M}$ is shuffled and reshaped into a matrix $\mathbf{O}$ with dimensions $N_{\text{rODT}} \times d$, as shown in Equation (5). Here, we use π to represent a bijective function that maps the index of each element in $\mathbf{M}$ to a unique position in $\mathbf{O}$ (i.e. permutation). The whole process is also illustrated in Figure 2b.

Specifically, $N_{rODT} = N_{cond}N_{col}/d$. To guarantee that N_{rODT} is an integer, we introduce an intermediate parameter, m, which ensures that N_{cond} is always a multiple of d by formulating $N_{cond} = md$. In practice, we use m to adjust N_{cond} instead of directly changing N_{cond}.

On the other hand, note that each row in $\mathbf{O}$ represents an rODT, which is crucial for subsequent operations. To ensure this consistency and the stability during training, the permutation is done only once during model construction and the configuration is then maintained throughout.

$$\mathbf{O} = \begin{bmatrix} o_{11} & \dots & o_{1d} \\ \vdots & \ddots & \vdots \\ o_{N_{\text{rODT}}1} & \dots & o_{N_{\text{rODT}}d} \end{bmatrix} \in \mathbb{R}^{N_{\text{rODT}} \times d},$$

$$\left\{ o_{jk} \mid j = \left\lceil \frac{\pi(n)}{d} \right\rceil, k = \pi(n) \bmod d, n = u \times N_{\text{col}} + v \right\} = \{m_{uv}\} \subset \mathbf{M}, \\ \text{where } 1 \le u \le N_{\text{cond}},\ 1 \le v \le N_{\text{col}} \tag{5}$$

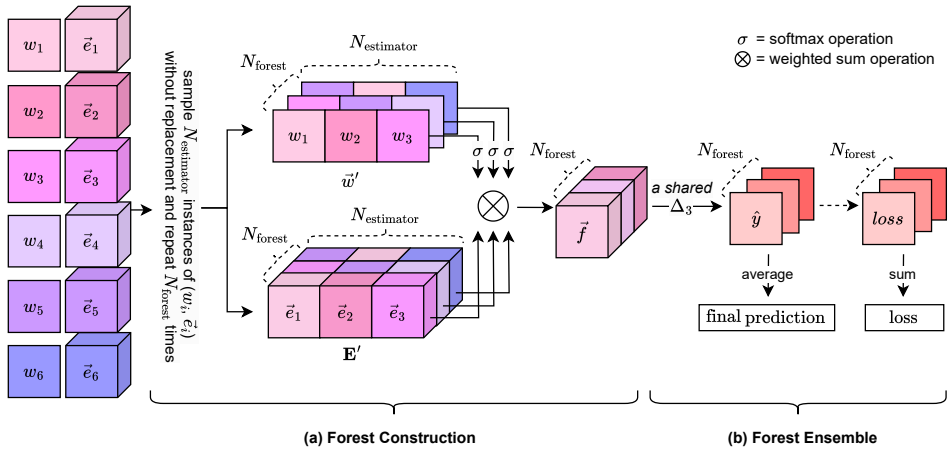


Figure 3: (a) Forest Construction: First, $N_{\text{estimator}}$ pairs of $(w_i, \vec{e}_i)$ are randomly sampled to form $\vec{w}'$ and $\mathbf{E}'$. Secondly, $\vec{w}'$ is transformed through a softmax function, and is used for computing the weighted sum of $\mathbf{E}'$ to form forest embedding $\vec{f}$. (b) Forest Ensemble: a shared-weight sub-network Δ_3 is employed to make a prediction $\hat{y}$ for each embedding. The final prediction is the average of all $\hat{y}$ values, and the total loss is the sum of their individual losses.

3.2.3 Two-level Relaxed ODT Ensemble

This module integrates rODTs to construct forests and then assembles multiple forests to conduct a final prediction.

Forest Construction. To construct an rODT forest using the generated rODTs, DOFEN introduces a sub-network and a standalone embedding vector for each rODT, denoted as Δ_{2i} and $\vec{e}_i$ respectively, where $i \in \{1, \ldots, N_{\text{rODT}}\}$. The role of Δ_{2i} is to evaluate how well a sample aligns with the conditions of an rODT, producing a weight scalar w_i, as shown in Equation (6) and Figure 2c.

$$\vec{w} = \begin{pmatrix} \Delta_{21}((o_{11}, \ldots, o_{1d})) \\ \vdots \\ \Delta_{2N_{\text{rODT}}}((o_{N_{\text{rODT}}1}, \ldots, o_{N_{\text{rODT}}d})) \end{pmatrix} = (w_1, \ldots, w_{N_{\text{rODT}}}) \in \mathbb{R}^{N_{\text{rODT}}} \tag{6}$$

The embedding vector $\vec{e}_i$ represents the tree information and is independent of the samples. The embedding vectors are combined into a matrix $\mathbf{E}$, as depicted in Equation (7), where N_{hidden} represents the size of the hidden dimension. Importantly, each tree embedding vector is directly linked to the specific conditions of its corresponding rODT. It is crucial to keep this association consistent throughout each training session to effectively train the tree embedding vectors.

$$\mathbf{E} = \begin{bmatrix} \vec{e}_1 \\ \vdots \\ \vec{e}_{N_{\text{rODT}}} \end{bmatrix} \in \mathbb{R}^{N_{\text{rODT}} \times N_{\text{hidden}}}, \text{where } \vec{e}_i \in \mathbb{R}^{N_{\text{hidden}}}, i = (1, 2, \ldots, N_{\text{rODT}}) \tag{7}$$

To further construct an rODT forest, $N_{\text{estimator}}$ of paired weights and embeddings are sampled from $\vec{w}$ and $\mathbf{E}$. This process is graphically represented in Figure 3a and described in line 3 to 7 of the pseudo-code for the two-level ensemble (Algorithm 1). The weights are processed through a softmax function and the weighted sum of embeddings forms the embedding vector $\vec{f}$ for an rODT forest. The magnitude of these softmaxed weights indicate the importance of the selected rODTs for making predictions. Noted that this process is repeated N_{forest} times to form N_{forest} instances of rODT forests.

Forest Ensemble. To make a prediction, DOFEN applies a shared sub-network Δ_3 to the embedding of each rODT forest to make individual predictions. The predictions are then averaged for a bagging

Algorithm 1: Two-level Relaxed ODT Ensemble

Input: $\vec{w}$, $\mathbf{E}$, N_{forest}, y, $\mathcal{L}$
Output: $\hat{y}$, $loss$

1 **Initialize** $\hat{y}, loss \leftarrow 0, 0$;
2 **for** $r \leftarrow 1$ **to** N_{forest} **do**
3 $\quad \vec{w}', \mathbf{E}' \xleftarrow{\text{sample without replacement}} \vec{w}, \mathbf{E}$; /* $N_{\text{estimator}}$ paired elements are sampled. */
4 $\quad \vec{w}' \in \mathbb{R}^{N_{\text{estimator}}}$;
5 $\quad \mathbf{E}' \in \mathbb{R}^{N_{\text{estimator}} \times N_{\text{hidden}}}$;
6 $\quad \vec{f} \leftarrow \sum^{N_{\text{estimator}}} \text{softmax}(\vec{w}') \circ \mathbf{E}'$; /* Element-wise multiplication with broadcast. */
7 $\quad \vec{f} \in \mathbb{R}^{N_{\text{hidden}}}$; /* $\vec{f}$ represents an rODT forest embedding. */
8 $\quad \hat{y}' \leftarrow \Delta_3(\vec{f})$; /* Give prediction with a shared Δ_3. */
9 $\quad loss \leftarrow loss + \mathcal{L}(\hat{y}', y)$; /* Calculate loss with loss function $\mathcal{L}$ and aggregate. */
10 $\quad \hat{y} \leftarrow \hat{y} + \hat{y}'$; /* Aggregate each forest's prediction. */
11 **end**
12 $\hat{y} \leftarrow \hat{y}/N_{\text{forest}}$;
13 **return** $(\hat{y}, loss)$;

ensemble. The process is detailed in line 1, 8, 10, and 12 in Algorithm 1 and is illustrated in Figure 3b. Notice that the output $\hat{y}_i$ is a scalar for regression tasks and a vector for classification tasks.

During training, DOFEN updates the model parameters by aggregating the loss from each prediction, as shown in line 9 in Algorithm 1. The loss function $\mathcal{L}$ is cross-entropy for classification tasks and mean squared error for regression tasks.

Notably, the sampling of weight-embedding pairs allows resampling in each forward pass without disrupting the training. In fact, the two-level rODT ensemble essentially implements a form of bootstrap aggregating (i.e. bagging) of trees. Conventional tree-based models like random forest bootstrap samples to generate a variety of trees, which are then combined to form a forest. In DOFEN, the $\vec{w}$ and $\mathbf{E}$ represent a tree pool. From this pool, trees are sampled with replacement to create diverse tree sets, or forests, represented by $\vec{w}'$ and $\mathbf{E}'$. These forests are then integrated to make the final prediction. The design of this tree bagging method enables the construction of varied base models (in this case, forests rather than individual trees) within a single training session, which is particularly suited to deep learning contexts. Although the randomization may seem chaotic, experiments shows that this approach contributes to the model's stability and generalizability, which is discussed in detail in Section 4.3.1 and Appendix F.1.

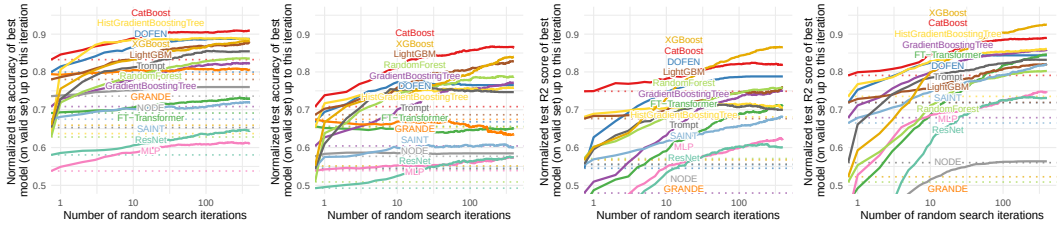
4 Experiments

This section presents a comprehensive analysis of our experimental results, organized as follows: The Tabular Benchmark and the baseline models are first introduced in Section 4.1. In Section 4.2, we evaluate DOFEN on the medium-sized Tabular Benchmark, while leaving the results for large-sized benchmark in Appendix G.1. Section 4.3 delves into DOFEN to elucidate the underlying mechanics that drive its performance. Additionally, we discuss DOFEN's computational efficiency in Appendices C.1 to C.3, analyze DOFEN's scalability in Appendix D, and show DOFEN's interpretability in Appendix E.

4.1 Tabular Benchmark Setup

Datasets. We strictly follow the protocols of the Tabular Benchmark as detailed in its official implementation[1]. This includes dataset splits, preprocessing methods, hyperparameter search guidelines, and evaluation metrics. For full details, please refer to the original paper [1]. The Tabular Benchmark categorized datasets into classification and regression, with features being either exclusively numerical or a combination of numerical and categorical (heterogeneous). These datasets are further

[1] https://github.com/LeoGrin/tabular-benchmark



(a) Exclusively Numerical, Classification (b) Heterogeneous, Classification (c) Exclusively Numerical, Regression (d) Heterogeneous, Regression

Figure 4: Results on medium-sized classification and regression datasets.

classified according to their sample size: medium-sized or large-sized. The dataset counts from Tabular Benchmark are provided in Appendix B.1, and the detailed datasets used in Tabular Benchmark are provided in Appendix B.3.

Model Selection. For model comparison, Tabular Benchmark includes four tree-based models: RandomForest, GradientBoostingTree [29], HGBT [30], and XGBoost; two generic DNN models: MLP and ResNet [11]; and two tabular DNN models: SAINT and FT-Transformer. To ensure a comprehensive comparison, we also included two additional tree-based models: LightGBM and CatBoost, and three tabular DNN models: NODE, Trompt, and GRANDE. LightGBM and CatBoost are selected due to their widespread use across various domains. NODE and GRANDE both share similar motivation and high-level structure with DOFEN, while Trompt represents the current state-of-the-art tabular DNNs when following the origin protocols of the Tabular Benchmark. The default hyperparameter configuration of DOFEN and hyperparameter search space of different models are presented in Appendices A.1 and H.2, and the list of some missing model baselines from Tabular Benchmark is provided in Appendix B.2.

4.2 Performance Evaluation

We analyze the results of medium-sized benchmark on classification and regression tasks separately. The evaluation metrics adhere to the Tabular Benchmark protocols, which use accuracy for classification datasets and the R-squared score for regression datasets. We discuss the overall performance in this section and provide comprehensive results for each dataset in Appendix G.2.

Classification. In Figure 4a, the models can be roughly categorized into three groups: (1) tree-based models and three tabular DNN models: DOFEN, Trompt and GRANDE, (2) three other tabular DNN models, and (3) the two generic DNN models. Prior to DOFEN, Trompt was the sole DNN model comparable to tree-based models. DOFEN not only matches but also surpasses the performance of most tree-based models, establishing a new benchmark for DNN models in tabular data. In Figure 4b, DOFEN and Trompt are again the only two DNN models grouped with tree-based models, yet they are positioned at the bottom of this group.

Regression. In Figure 4c, XGBoost stands out as a distinct category. Meanwhile, CatBoost and DOFEN represent a second level of performance. Notably, XGBoost and DOFEN demonstrate a significant improvement during the hyperparameter optimization, whereas CatBoost maintains strong performance consistently. In Figure 4d, XGBoost and CatBoost continue to hold the top two positions. DOFEN, ending up in sixth place, is overtaken by GradientBoostingTree as well as HGBT, and is comparable with FT-Transformer towards the end of the hyperparameter search process.

The analysis of Figure 4 allows us to draw several conclusions. When compared to DNN models, DOFEN consistently either ranks first or shares the top positions. Additionally, DOFEN exhibits strong competitiveness against tree-based models. In datasets with numerical features, it consistently places within the top three. However, in the context of heterogeneous features, DOFEN's performance is moderate, typically falling in the middle or lower tiers in comparison with tree-based models. This challenge in managing heterogeneous features is a prevalent issue among all DNN models, highlighting an area for potential improvement in future tabular DNN models.

Table 1: Mean (μ) and standard deviation (σ) of DOFEN's performance with 15 random seeds on 4 datasets from different tasks.

	N_{forest}	**1**	**10**	**20**	**50**	**100 (default)**	**400**
jannis	μ (↑)	0.7382	0.7747	0.7782	0.7800	0.7808	<u>0.7814</u>
(numerical classification)	σ (↓)	0.0060	0.0019	0.0015	0.0006	0.0007	<u>0.0004</u>
road-safety	μ (↑)	0.7517	0.7712	0.7720	0.7728	<u>0.7732</u>	<u>0.7732</u>
(heterogeneous classification)	σ (↓)	0.0118	0.0010	0.0007	0.0004	0.0005	<u>0.0003</u>
delays-zurich	μ (↑)	0.0054	0.0248	0.0258	0.0265	0.0268	<u>0.0270</u>
(numerical regression)	σ (↓)	0.0033	0.0009	0.0005	0.0003	0.0003	<u>0.0002</u>
abalone	μ (↑)	0.5469	0.5810	0.5846	0.5862	0.5868	<u>0.5870</u>
(heterogeneous regression)	σ (↓)	0.0181	0.0038	0.0026	0.0017	0.0010	<u>0.0004</u>

4.3 Additional Analysis

This section is dedicated to a deeper exploration of the DOFEN model. Randomness plays an important role in DOFEN, as both the condition selection of an rODT and rODT selection of a forest involve random processes. A straightforward concern is the stability of DOFEN, which is examined in Section 4.3.1. Moreover, given that the conditions are randomly selected, we investigate whether this randomness leads to redundant trees in Section 4.3.2. In addition to randomness, another distinct feature of DOFEN is the introduction of a higher-level ensemble that combines multiple forests, instead of merely assembling trees into a forest. Appendix F.1 discusses the impact of removing this higher-level ensemble on DOFEN.

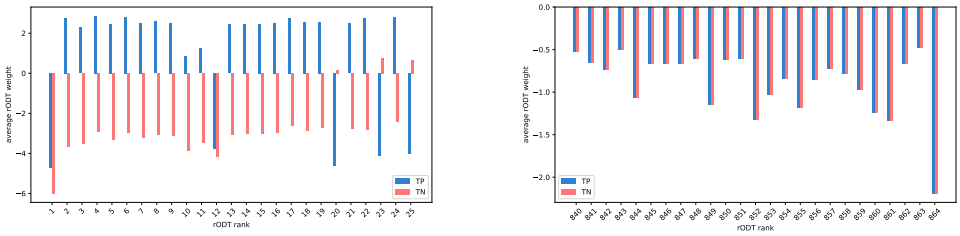
All experiments in this section are conducted using the default hyperparameters and medium-sized datasets from the Tabular Benchmark. For evaluation metrics, accuracy is used for classification datasets, while the R-squared score for regression datasets. Except for the cases evaluated on individual datasets, the results represent the averaged metrics across the corresponding datasets.

4.3.1 Model Stability

DOFEN incorporates randomness at two steps: firstly, in the selection of conditions as shown in Equation (5) for rODT construction, and secondly, in the sampling of rODTs as shown in line 3 of Algorithm 1 for a two-level rODT ensemble. This section explores how randomness affects the stability of DOFEN.

We start by analyzing the variation in performance of four datasets, where DOFEN ranks first, as shown in Table 1. The standard deviations are even negligible when $N_{\text{forest}} = 1$ (about 0.1% to 1% to mean), except for the delays-zurich dataset. Moreover, with increased N_{forest}, the standard deviations become even smaller (about 0.01% to 0.1% to mean). These results suggest that the stability of DOFEN is not an issue in most cases ($N_{forest} > 10$), and using the default setting of DOFEN ($N_{forest} = 100$) ensures both adequate performance and stability for most datasets. Furthermore, the performance improves as the N_{forest} increases, indicating that the tree bagging of DOFEN not only mitigates instability but also enhances the model's generalizability.

In addition to analyzing the intrinsic instability, we also replace the steps involving randomness with deterministic alternatives to assess the impact of the randomness on DOFEN from a different perspective. For the selection of conditions, we utilize CatBoost to choose columns based on a predefined criterion. The detailed results, presented in Appendix F.3, reveal that the predefined criterion perform only slightly better than the shuffle-then-reshape process. Considering the differentiability and the potential for end-to-end training, random selection of conditions remains a viable and promising option. For sampling rODTs, we implement a sliding window technique to lock in the selected trees for each forest. The results are detailed in Appendix F.4, which suggests that our straightforward approach is comparable to a more sophisticated approach.



(a) Relaxed ODTs with large weight variation.

(b) Relaxed ODTs with small weight variation.

Figure 5: In the covertype dataset, Figure 5a shows that the average weights of true positives differ significantly from those of true negatives. Conversely, Figure 5b reveals a contrasting result for rODTs with small weight variation.

4.3.2 Weights of Individual Relaxed ODT

In DOFEN, an rODT is assigned a weight to predict a sample, as shown in Equation (6). In this section, we analyze a binary classification dataset (covertype) to observe the variation in the weights assigned to individual rODTs, as shown in Figure 5.

Figure 5a shows that, for most rODTs ranked in the top 25 according to their standard deviations of weights, there is a significant difference between the average weights of true positives and those of true negatives. Conversely, Figure 5b shows an opposite trend for rODTs with the smallest standard deviations of weights. These trends are also observed in another dataset, as shown in Appendix F.5. These observations imply that rODTs with larger standard deviations of weights is more crucial role in classifying samples.

In addition, we come up with an idea to examine the performance change after pruning weights with small standard deviations and their corresponding embeddings, since they are not sensitive to samples with different label. The results are provided in Appendix F.6 and suggest that the variation serves as a reliable indicator of the importance of rODTs. Moreover, pruning the less important rODTs not only enhances the model's efficiency but also its performance.

5 Limitation and Conclusion

Limitation. Although DOFEN shows promising results, it still contains two weaknesses. First, the inference time of DOFEN is relatively long compared to other DNN models, as shown in Appendix C.1. However, Appendix C.1 also shows that DOFEN possesses the fewest floating point operations (FLOPs). This inconsistency between inference time and FLOPs is mainly caused by the group convolution operation for calculating weights for each rODT (Appendix C.2), which can be improved in the future implementation of DOFEN. Second, the randomization steps involved in DOFEN result in a slower convergence speed, meaning that DOFEN requires more training steps to reach optimal performance. This is reflected in the relatively larger number of training epochs needed for DOFEN. Therefore, the workaround strategy of differentiable sparse selection proposed in this study is merely a starting point, demonstrating its potential. Finding more efficient strategies will be the future work.

Conclusion. In this work, we proposed DOFEN, a novel tree-inspired DNN for tabular data that achieves on-off sparse selections of columns. DOFEN first constructs sufficiently large number of rODTs and randomly ensembles these rODTs into multiple rODT forests to make prediction. DOFEN was evaluated on the Tabular Benchmark, achieving state-of-the-art results compared to DNN-based models and proving competitive with tree-based ones. Furthermore, we showed that the randomization steps involved in DOFEN do not compromise stability but do yield redundant rODTs. Nevertheless, redundant rODTs can be efficiently removed through our pruning method. In summary, based on DOFEN's outstanding performance, it has the potential to serve as the backbone model for tabular data across various scenarios, including self- and semi-supervised learning, as well as multi-modal training.

References

[1] Grinsztajn, L.; Oyallon, E.; Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems* **2022**, *35*, 507–520.

[2] Breiman, L. Random forests. *Machine learning* **2001**, *45*, 5–32.

[3] Geurts, P.; Ernst, D.; Wehenkel, L. Extremely randomized trees. *Machine learning* **2006**, *63*, 3–42.

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

[5] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* **2017**, *30*.

[6] Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A. V.; Gulin, A. CatBoost: unbiased boosting with categorical features. *Advances in neural information processing systems* **2018**, *31*.

[7] Zhou, Z.-H.; Feng, J. Deep forest. *National science review* **2019**, *6*, 74–86.

[8] Popov, S.; Morozov, S.; Babenko, A. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312* **2019**,

[9] Arik, S. Ö.; Pfister, T. Tabnet: Attentive interpretable tabular learning. Proceedings of the AAAI conference on artificial intelligence. 2021; pp 6679–6687.

[10] Somepalli, G.; Goldblum, M.; Schwarzschild, A.; Bruss, C. B.; Goldstein, T. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342* **2021**,

[11] Gorishniy, Y.; Rubachev, I.; Khrulkov, V.; Babenko, A. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* **2021**, *34*, 18932–18943.

[12] Chen, K.-Y.; Chiang, P.-H.; Chou, H.-R.; Chen, T.-W.; Chang, T.-H. Trompt: Towards a Better Deep Neural Network for Tabular Data. *arXiv preprint arXiv:2305.18446* **2023**,

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

[14] Borisov, V.; Leemann, T.; Seßler, K.; Haug, J.; Pawelczyk, M.; Kasneci, G. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems* **2022**,

[15] McElfresh, D.; Khandagale, S.; Valverde, J.; Ramakrishnan, G.; Goldblum, M.; White, C.; others When Do Neural Nets Outperform Boosted Trees on Tabular Data? *arXiv preprint arXiv:2305.02997* **2023**,

[16] Peters, B.; Niculae, V.; Martins, A. F. Sparse Sequence-to-Sequence Models. Proc. ACL. 2019.

[17] Martins, A.; Astudillo, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. International conference on machine learning. 2016; pp 1614–1623.

[18] Kohavi, R. Bottom-up induction of oblivious read-once decision graphs. European Conference on Machine Learning. 1994; pp 154–169.

[19] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* **2014**, *15*, 1929–1958.

[20] Quinlan, J. R. Induction of decision trees. *Machine learning* **1986**, *1*, 81–106.

[21] Marton, S.; Lüdtke, S.; Bartelt, C.; Stuckenschmidt, H. GradTree: Learning axis-aligned decision trees with gradient descent. Proceedings of the AAAI Conference on Artificial Intelligence. 2024; pp 14323–14331.

[22] Marton, S.; Lüdtke, S.; Bartelt, C.; Stuckenschmidt, H. GRANDE: Gradient-Based Decision Tree Ensembles for Tabular Data. The Twelfth International Conference on Learning Representations. 2024.

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

[24] Hollmann, N.; Müller, S.; Eggensperger, K.; Hutter, F. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848* **2022**,

[25] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **2017**, *30*.

[26] Müller, S.; Hollmann, N.; Arango, S. P.; Grabocka, J.; Hutter, F. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510* **2021**,

[27] Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; others Improving language understanding by generative pre-training. **2018**,

[28] Lou, Y.; Obukhov, M. Bdt: Gradient boosted decision tables for high accuracy and scoring efficiency. Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017; pp 1893–1901.

[29] Friedman, J. H. Stochastic gradient boosting. *Computational statistics & data analysis* **2002**, *38*, 367–378.

[30] Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.

[31] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; others Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* **2019**, *32*.

[32] Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* **2017**,

[33] fvcore library. `https://github.com/facebookresearch/fvcore/`.

[34] Averagemn LGBM with hyperopt tuning. 2019; `https://www.kaggle.com/code/donkeys/lgbm-with-hyperopt-tuning/notebook`, [Online; accessed 5-January-2023].

[35] Bahmani, M. Understanding LightGBM Parameters (and How to Tune Them). 2022; `https://neptune.ai/blog/lightgbm-parameters-guide`, [Online; accessed 5-January-2023].

[36] Vanschoren, J.; van Rijn, J. N.; Bischl, B.; Torgo, L. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* **2013**, *15*, 49–60.

Appendix

Table of Contents

A More DOFEN Settings **14**
- A.1 Default Hyperparameters Settings for DOFEN . . . 14
- A.2 Detailed Model Configurations. . . . 14
- A.3 Actual $N_{\text{estimator}}$ for each Dataset . . . 14

B More Tabular Benchmark Settings **16**
- B.1 Dataset Counts . . . 16
- B.2 Missing Model Baselines . . . 16
- B.3 Mappings of OpenML Task ID and Dataset Name . . . 16

C Computational Efficiency Analysis **18**
- C.1 Computational Efficiency Analysis . . . 18
- C.2 Long Inference Time of DOFEN . . . 19
- C.3 Training Time of DOFEN . . . 20

D Scalability of DOFEN **21**

E Interpretability of DOFEN **22**

F More Analysis **23**
- F.1 Sampling in Relaxed ODT Forest Ensemble . . . 23
- F.2 Seed Ensemble . . . 24
- F.3 An Alternative Strategy for Condition Selection . . . 25
- F.4 An Alternative Strategy for Weight Selection . . . 25
- F.5 More Experiments for Section 4.3.2 (Activated rODT for Different Classes) . . . 26
- F.6 Pruning of Relaxed ODT . . . 26

G More Evaluation Results on Tabular Benchmark **27**
- G.1 Performance Evaluation on Large-sized Benchmark . . . 27
- G.2 Detailed Evaluation Results . . . 28

H More Experiment Settings **44**
- H.1 Hardware Used . . . 44
- H.2 Hyperparameter Search Space . . . 44

A More DOFEN Settings

A.1 Default Hyperparameters Settings for DOFEN

In this section, we describe the hyperparameters used in our DOFEN model, along with their default values, as shown in Table 2. All notations used here have been previously introduced in Section 3, except for dropout_rate. The dropout_rate is applied in dropout layers, and its usage is detailed in Appendix A.2.

The calculated $N_{\text{estimator}}$ for each dataset can be found in Appendix A.3. Additionally, the hyperparameter search spaces for both the DOFEN model and all baseline models are detailed in Appendix H.2.

DOFEN is implemented in Pytorch [31]. For hyperparameters used in model optimization (e.g. optimizer, learning rate, weight decay, etc.), all experiments share the same settings. Specifically, DOFEN uses AdamW optimizer [32] with $1\mathrm{e}{-3}$ learning rate and no weight decay. The batch size is set to 256, and DOFEN is trained for 500 epochs without using learning rate scheduling or early stopping.

Table 2: The default hyperparameters of DOFEN.

Hyperparameter	**Default Value**
N_{col}	depends on dataset
d [1]	4
m [2]	16
N_{cond}	md
N_{rODT}	$N_{\text{col}}N_{\text{cond}}/d = N_{\text{col}}m$
$N_{\text{estimator}}$	$\max\{2, \lfloor\sqrt{N_{\text{col}}}\rfloor\} \cdot N_{\text{cond}}/d$
N_{forest}	100
N_{hidden}	128
N_{class}	depends on dataset
dropout_rate	0.0

[1] depth of a rODT
[2] an intermediate parameter to ensure that N_{rODT} is an integer

A.2 Detailed Model Configurations.

In this appendix, we elucidate the specific configurations of the neural network layer composites, denoted as Δ_1, Δ_2, and Δ_3 in the main paper.

1. Δ_1 - Generate conditions for each column: Δ_1 is designed to generate conditions for both numerical and categorical data columns, as detailed in Figure 6. For categorical columns in particular, we employ embedding layers. These layers are utilized to transform categorical features into a format that the neural network can effectively process.
2. Δ_2 and Δ_3 - Derive weights and make predictions: The layers represented by Δ_2 and Δ_3 are responsible for generating weights based on the combination of conditions and making predictions, respectively. The relevant structures and processes are illustrated in Figure 7 and Figure 8.
3. Key Parameters:
    - *num_categories*: This parameter represents the number of distinct categories in a given categorical column.
    - drop_rate: This hyperparameter defines the extent of dropout operations applied within the network.

A.3 Actual $N_{\text{estimator}}$ for each Dataset

The $N_{\text{estimator}}$ is calculated through a pre-defined formula as shown in Table 2. In this section, we provide the calculated $N_{\text{estimator}}$ for each dataset in Table 3 when using default hyperparameters. Datasets are represented by their OpenML ID as described in Appendix B.3.

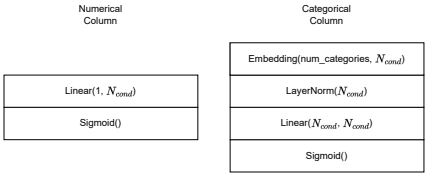


Figure 6: Detailed network layer composite for Δ_1.

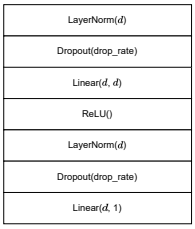


Figure 7: Detailed network layer composite for Δ_2.

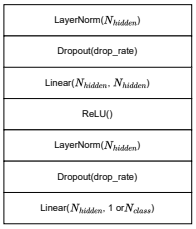


Figure 8: Detailed network layer composite for Δ_3.

Table 3: $N_{\text{estimator}}$ for each dataset, as long as their N_{col} and N_{rODT}.

OpenML ID	361086	361294	361094	361289	361293	361085	361082	361103	361080
N_{col}	3	3	4	4	5	6	6	6	6
N_{rODT}	48	48	64	64	80	96	96	96	96
$N_{\text{estimator}}$	32	32	32	32	32	32	32	32	32
OpenML ID	361273	361066	361060	361280	361093	361110	361288	361281	361277
N_{col}	7	7	7	7	7	8	8	8	8
N_{rODT}	112	112	112	112	112	128	128	128	128
$N_{\text{estimator}}$	32	32	32	32	32	32	32	32	32
OpenML ID	361081	361078	361104	361083	361096	361055	361061	361065	361095
N_{col}	8	8	9	9	9	10	10	10	10
N_{rODT}	128	128	144	144	144	160	160	160	160
$N_{\text{estimator}}$	32	32	48	48	48	48	48	48	48
OpenML ID	361076	361098	361291	361099	361286	361087	361084	361063	361074
N_{col}	11	11	11	11	11	13	15	16	16
N_{rODT}	176	176	176	176	176	208	240	256	256
$N_{\text{estimator}}$	48	48	48	48	48	48	48	64	64
OpenML ID	361079	361101	361102	361070	361275	361072	361283	361278	361111
N_{col}	16	16	17	20	20	21	21	22	23
N_{rODT}	256	256	272	320	320	336	336	352	368
$N_{\text{estimator}}$	64	64	64	64	64	64	64	64	64
OpenML ID	361069	361062	361073	361282	361285	361077	361279	361068	361113
N_{col}	24	26	26	31	32	33	42	50	54
N_{rODT}	384	416	416	496	512	528	672	800	864
$N_{\text{estimator}}$	64	80	80	80	80	80	96	112	112
OpenML ID	361274	361088	361091	361292	361287	361097	361276		
N_{col}	54	79	91	124	255	359	419		
N_{rODT}	864	1264	1456	1984	4080	5744	6704		
$N_{\text{estimator}}$	112	128	144	176	240	288	320		

B More Tabular Benchmark Settings

B.1 Dataset Counts

In this section, we provide the dataset counts for each task for your reference, as presented in Table 4.

Table 4: Dataset counts for each task.

Task	Feature	Count
medium-sized classification	numerical	16
	heterogenous	7
medium-sized regression	numerical	19
	heterogenous	17
large-sized classification	numerical	4
	heterogenous	2
large-sized regression	numerical	3
	heterogenous	5

B.2 Missing Model Baselines

We found that two baselines, MLP and HGBT, are absent from the evaluation results in the large-sized classification task because they are missing from the official repository. Furthermore, MLP, HGBT, and RandomForest are not included in the large-sized regression task for the same reason.

B.3 Mappings of OpenML Task ID and Dataset Name

In this section, we introduce the mappings between OpenML Task IDs and elaborate on how to download the corresponding datasets using these IDs.

The mappings are provided in Tables 5 to 8. To access the datasets, please follow the links below, which direct you to the OpenML website for each type of dataset. You can then search using the OpenML ID.

- Classification datasets with numerical features only:
  `https://www.openml.org/search?type=benchmark&study_type=task&id=337`
- Classification datasets with heterogeneous features:
  `https://www.openml.org/search?type=benchmark&study_type=task&id=334`
- Regression datasets with numerical features only:
  `https://www.openml.org/search?type=benchmark&study_type=task&id=336`
  `https://www.openml.org/search?type=benchmark&study_type=task&id=297`
  (only for task ID 361091)
- Regression datasets with heterogeneous features:
  `https://www.openml.org/search?type=benchmark&study_type=task&id=335`
  `https://www.openml.org/search?type=benchmark&study_type=task&id=299`
  (only for task ID 361095)

Table 5: OpenML Task ID mappings for **classification** datasets with **numerical features only**.

OpenML ID	Dataset
361055	credit
361060	electricity
361061	covertype
361062	pol
361063	house_16H
361065	MagicTelescope
361066	bank-marketing
361068	MiniBooNE
361069	Higgs
361070	eye_movements
361273	Diabetes130US
361274	jannis
361275	default-of-credit-card-clients
361276	Bioresponse
361277	california
361278	heloc

Table 6: OpenML Task ID mappings for **classification** datasets with **heterogeneous features**.

OpenML ID	Dataset
361110	electricity
361111	eye_movements
361113	covertype
361282	albert
361283	default-of-credit-card-clients
361285	road-safety
361286	compas-two-years

Table 7: OpenML Task ID mappings for **regression** datasets with **numerical features only**.

OpenML ID	Dataset
361072	cpu_act
361073	pol
361074	elevators
361076	wine_quality
361077	Ailerons
361078	houses
361079	house_16H
361080	diamonds
361081	Brazilian_houses
361082	Bike_Sharing_Demand
361083	nyc-taxi-green-dec-2016
361084	house_sales
361085	sulfur
361086	medical_charges
361087	MiamiHousing2016
361088	superconduct
361091	year
361279	yprop_4_1
361280	abalone
361281	delays_zurich_transport

Table 8: OpenML Task ID mappings for **regression** datasets with **heterogeneous features**.

OpenML ID	Dataset
361093	analcatdata_supreme
361094	visualizing_soil
361095	black_friday
361096	diamonds
361097	Mercedes_Benz_Greener_Manufacturing
361098	Brazilian_houses
361099	Bike_Sharing_Demand
361101	nyc-taxi-green-dec-2016
361102	house_sales
361103	particulate-matter-ukair-2017
361104	SGEMM_GPU_kernel_performance
361287	topo_2_1
361288	abalone
361289	seattlecrime6
361291	delays_zurich_transport
361292	Allstate_Claims_Severity
361293	Airlines_DepDelay_1M
361294	medical_charges

C Computational Efficiency Analysis

C.1 Computational Efficiency Analysis

To discuss the computational efficiency, we analyzed the average floating point operations (FLOPs) [33], parameter sizes, and inference time of DOFEN and other baseline models. Our analyses covered both the default and optimal hyperparameter settings, where the optimal hyperparameter delivers the best performance for each model on each dataset. The experiments involving DNN-based models were performed using an NVIDIA GeForce RTX 2080 Ti, while those for the GBDT-based models utilized an AMD EPYC 7742 64-core Processor with 16 threads.

We begin with the comparison between DNN-based and GBDT-based models. This comparison primarily focuses on inference time, as FLOPs and parameter sizes are applicable for evaluating the efficiency of DNN-based models but cannot be applied to GBDTs. Additionally, inference times under the optimal parameters are provided only when those parameters are available. As shown in Tables Table 9 to Table 12, the inference times for all DNN-based models are slower than those for GBDT-based models. This is expected due to the inherent differences between the two types of models.

When compared to other DNN baselines, DOFEN achieves the highest performance, the lowest FLOPs, and the smallest parameter sizes but exhibits the relatively long inference time among all the DNN-based models. This inconsistency between FLOPs and inference time suggests that there is still room for implementation improvements in DOFEN. Hence, we conduct additional experiments to analyze which part of the DOFEN model is the computational bottleneck, as discussed in Appendix C.2, showing that the bottleneck of DOFEN arises from using group operations when constructing rODTs. Although this does not affect DOFEN's article, improvements can be made during future open-source releases.

Table 9: Computational efficiency analysis of default hyperparameters on medium-sized classification datasets.

Model	Performance (Accuracy)	FLOPs (M)	Parameters (M)	Inference time (sec.)
DOFEN	0.7725	0.1845	0.0140	0.0125
Trompt	0.7704	53.2127	3.8608	0.0225
FT-Transformer	0.7662	3.3147	0.0908	0.0058
NODE	0.7658	0.8299	0.7525	0.0041
XGBoost	0.7717	–	–	0.0015
LightGBM	0.7757	–	–	0.0016
CatBoost	0.7777	–	–	0.0029

Table 10: Computational efficiency analysis of optimal hyperparameters on medium-sized classification datasets.

Model	Performance (Accuracy)	FLOPs (M)	Parameters (M)	Inference time (sec.)
DOFEN	0.7805	0.2093	0.0437	0.0213
Trompt	0.7797	38.7712	2.0398	0.0202
FT-Transformer	0.7686	6.0696	0.2514	0.0061
NODE	0.7677	3.2860	2.6778	0.0033
XGBoost	0.7848	–	–	0.0014
LightGBM *	0.7838	–	–	N/A
CatBoost *	0.7858	–	–	N/A

* The evaluation results are obtained from the Trompt paper without the corresponding optimal hyperparameters. Thus, the inference time under the optimal hyperparameters is unavailable.

Table 11: Computational efficiency analysis of default hyperparameters on medium-sized regression datasets.

Model	Performance (R2 Score)	FLOPs (M)	Parameters (M)	Inference time (sec.)
DOFEN	0.6611	0.1875	0.0173	0.0105
Trompt	0.6541	45.8507	3.8591	0.0224
FT-Transformer	0.6359	2.7795	0.0909	0.0039
NODE	0.1080	0.5839	0.5065	0.0039
XGBoost	0.6719	–	–	0.0012
LightGBM	0.6832	–	–	0.0014
CatBoost	0.6896	–	–	0.0030

Table 12: Computational efficiency analysis of optimal hyperparameters on medium-sized regression datasets.

Model	Performance (R2 Score)	FLOPs (M)	Parameters (M)	Inference time (sec.)
DOFEN	0.6882	0.2030	0.0364	0.0182
Trompt	0.6830	17.9560	1.2857	0.0200
FT-Transformer	0.6834	9.0576	0.2965	0.0065
NODE	0.6631	2.1379	1.6930	0.0035
XGBoost	0.6985	–	–	0.0014
LightGBM *	0.6896	–	–	N/A
CatBoost *	0.6940	–	–	N/A

* The evaluation results are obtained from the Trompt paper without the corresponding optimal hyperparameters. Thus, the inference time under the optimal hyperparameters is unavailable.

C.2 Long Inference Time of DOFEN

To find out the computation bottleneck of DOFEN, we analyzed the inference time of each DOFEN module in proportion, as shown in Table 13 and Table 14, which is averaged across 59 medium-sized

datasets with default hyperparameters. Table A1 shows that the Forest Construction module consumes the most inference time. In Table A2, more detailed operations reveal that the sub-module Δ_2 in the Forest Construction module, which generates weights for each rODT, has the longest inference time.

The sub-module Δ_2 is designed with multiple MLP and normalization layers, implemented using group convolution and group normalization to parallelize scoring for each rODT. However, the efficiency of group convolution in PyTorch has been problematic and remains unresolved. Specifically, the operation efficiency decreases as the number of groups increases, sometimes making it slower than separate convolutions in CUDA streams (see PyTorch issues 18631, 70954, 73764). The sub-module Δ_1 also uses group convolution to parallelize condition generation across different numerical columns, resulting in slower inference times compared to other operations, though less significant than Δ_2 due to fewer groups being used.

However, we mainly focus on the concept and model structure in this paper, acknowledging that model implementation can be further optimized. For example, attention operations are originally slow due to quadratic complexity, and many recent works have successfully accelerated the speed of attention operations and reduced their memory usage. Hence, we believe there will be better implementations of these group operations with much greater efficiency in the future.

Table 13: Average inference time proportion of each DOFEN module across 59 medium-sized datasets.

Module Name	**Source**	**Inference time proportion (mean)**	**Inference time proportion (std)**
Condition Generation	Figure 2a	7.03 %	5.29 %
Relaxed ODT Construction	Figure 2b	1.64 %	0.90 %
Forest Construction	Figure 2c and Figure 3a	87.39 %	7.68 %
Forest Ensemble	Figure 3b	3.94 %	2.36 %

Table 14: Average inference time proportion of each DOFEN operation across 59 medium-sized datasets.

Module Name	**Operation**	**Source**	**Inference time proportion (mean)**
Condition Generation	Δ_1	Equation (4)	7.03 %
Relaxed ODT Construction	permutation and reshape	Equation (5)	1.64 %
Forest Construction	Δ_2	Equation (6)	85.11 %
	get rODT embedding	Equation (7)	0.22 %
	sample rODTs to form forests	Algorithm 1, line 3	1.01 %
	softmax + weighted sum	Algorithm 1, line 6	1.05 %
Forest Ensemble	Δ_3	Algorithm 1, line 8	3.55 %
	average forest predictions	Algorithm 1, line 10 and 12	0.39 %

C.3 Training Time of DOFEN

To know more about how the slow inference time will affect the training time of DOFEN, we also conducted an experiment to compare the training time of DOFEN with other deep learning methods included in our paper (i.e. Trompt, FT Transformer, and NODE). We measured the training time on medium-sized datasets using both default and optimal hyperparameter settings, where the optimal hyperparameters refers to the settings that deliver the best performance for each model on each dataset.

This experiment was conducted using a single NVIDIA Tesla V100 GPU. During model training, we carefully ensured that no other computational processes were running concurrently to enable a fair comparison. Additionally, we excluded datasets that would cause OOM (Out of Memory) issues during training, resulting in the selection of 50 out of 59 medium-sized datasets.

The average training time across datasets for each model is provided in Table A7. The results show that the training time for DOFEN is relatively long, approximately twice as long as Trompt when

using optimal hyperparameters. This extended training time may be due to the inefficient group operations involved in DOFEN, which consume about 80% of the computation time during the forward pass. For more details, please refer to Appendix C.2. Therefore, improving the efficiency of group operations could reduce both the training and inference time of DOFEN.

Table 15: Average training time of different methods using default and optimal hyperparameter settings on 50 medium-sized datasets. Numbers are in Seconds, with lower values indicating faster training speed.

Model Name	**Training Time (Default)**	**Training Time (Optimal)**
DOFEN	332.6998 +/- 125.1965	1143.7674 +/- 804.3809
Trompt	552.3495 +/- 213.3278	535.1781 +/- 291.9933
FT-Transformer	80.3425 +/- 57.2647	99.1068 +/- 79.2272
NODE	95.0274 +/- 54.7463	427.8625 +/- 394.1191

D Scalability of DOFEN

To discuss the scalability of DOFEN, we have conducted experiments to investigate its performance given changes in hyperparameters m, d, and the number of MLP layers (num_layers). In detail, changes in m and d affect the number of conditions (N_{cond}), while alterations in m impact both the total number of rODTs (N_{rODT}) and the number of rODTs within an rODT forest ($N_{\text{estimator}}$). For further details on these parameters, please refer to Table 2. The num_layers hyperparameter, newly introduced, refers to the number of MLP layers in neural networks Δ_1, Δ_2, and Δ_3. A detailed introduction to Δ_1, Δ_2, and Δ_3 can be found in Appendix A.2.

Due to limited computational resources, we only conducted this experiment on datasets that would not cause out-of-memory (OOM) issues on our machine across all hyperparameter settings. This selection resulted in 51 out of 59 medium-sized datasets and 10 out of 14 large-sized datasets.

Based on Table 16 to Table 21, we observed that larger values of m and d enhance DOFEN's performance. Notably, improvements are more significant with large-sized datasets than with medium-sized datasets, likely because larger datasets benefit more from increased model capacity. In contrast, Table 20 reveals that an increase in num_layers generally results in poorer performance. This could be attributed to the substantial growth in parameter size and FLOPs, compared to adjustments in the m and d, potentially leading to overfitting.

Table 16: Analysis of performance and efficiency across varied settings of m on medium-sized datasets.

	m	**4**	**8**	**16 (default)**	**32**	**64**
Classification	Performance (Accuracy)	0.7491	0.7552	0.7602	0.7601	<u>0.7603</u>
	Parameters (M)	<u>0.0029</u>	0.0042	0.0070	0.0134	0.0296
	FLOPs (M)	<u>0.1797</u>	0.1802	0.1815	0.1849	0.1951
Regression	Performance (R2 score)	0.6496	0.6488	0.6796	<u>0.6940</u>	0.6603
	Parameters (M)	<u>0.0026</u>	0.0035	0.0056	0.0105	0.0235
	FLOPs (M)	<u>0.1783</u>	0.1787	0.1797	0.1825	0.1912

Table 17: Analysis of performance and efficiency across varied settings of m on large-sized datasets.

	m	**4**	**8**	**16 (default)**	**32**	**64**
Classification	Performance (Accuracy)	0.7498	0.7635	0.7800	0.7922	<u>0.8010</u>
	Parameters (M)	<u>0.0033</u>	0.0050	0.0084	0.0159	0.0333
	FLOPs (M)	<u>0.1798</u>	0.1804	0.1819	0.1854	0.1949
Regression	Performance (R2 score)	0.7227	0.7521	0.7583	<u>0.7698</u>	0.7697
	Parameters (M)	<u>0.0025</u>	0.0034	0.0058	0.0127	0.0350
	FLOPs (M)	<u>0.1783</u>	0.1788	0.1803	0.1856	0.2045

Table 18: Analysis of performance and efficiency across varied settings of d on medium-sized datasets.

	d	**2**	**3**	**4 (default)**	**6**	**8**
Classification	Performance (Accuracy)	0.7402	0.7588	0.7602	0.7583	0.7545
	Parameters (M)	0.0058	0.0064	0.0070	0.0087	0.0108
	FLOPs (M)	0.1801	0.1807	0.1815	0.1834	0.1857
Regression	Performance (R2 score)	0.5961	0.6699	0.6796	0.6111	0.6914
	Parameters (M)	0.0047	0.0051	0.0056	0.0069	0.0087
	FLOPs (M)	0.1786	0.1791	0.1797	0.1812	0.1831

Table 19: Analysis of performance and efficiency across varied settings of d on large-sized datasets.

	d	**2**	**3**	**4 (default)**	**6**	**8**
Classification	Performance (Accuracy)	0.7433	0.7726	0.7800	0.7853	0.7916
	Parameters (M)	0.0071	0.0077	0.0084	0.0102	0.0125
	FLOPs (M)	0.1803	0.1810	0.1819	0.1840	0.1865
Regression	Performance (R2 score)	0.6572	0.7443	0.7583	0.7694	0.7704
	Parameters (M)	0.0043	0.0050	0.0058	0.0082	0.0113
	FLOPs (M)	0.1787	0.1794	0.1803	0.1828	0.1860

Table 20: Analysis of performance and efficiency across varied settings of num_layers on medium-sized datasets.

	num_layers	**Default (1, 2, 2)**	**Twice (2, 4, 4)**	**Triple (3, 6, 6)**
Classification	Performance (Accuracy)	0.7602	0.7592	0.7481
	Parameters (M)	0.0070	0.0189	0.0308
	FLOPs (M)	0.1815	0.5311	0.8808
Regression	Performance (R2 score)	0.6796	0.6595	0.7731
	Parameters (M)	0.0056	0.0150	0.0245
	FLOPs (M)	0.1797	0.5267	0.8737

Table 21: Analysis of performance and efficiency across varied settings of num_layers on large-sized datasets.

	num_layers	**Default (1, 2, 2)**	**Twice (2, 4, 4)**	**Triple (3, 6, 6)**
Classification	Performance (Accuracy)	0.7800	0.7959	0.7638
	Parameters (M)	0.0084	0.0231	0.0379
	FLOPs (M)	0.1819	0.5346	0.8873
Regression	Performance (R2 score)	0.7583	0.7575	0.6715
	Parameters (M)	0.0058	0.0139	0.0220
	FLOPs (M)	0.1803	0.5259	0.8715

E Interpretability of DOFEN

This section aims to demonstrate the interpretability of DOFEN. Specifically, we adopt a feature importance metric akin to the "split" or "weight" importance used in LightGBM and XGBoost, which counts how often a feature is used in the model.

To calculate DOFEN's feature importance of a specific sample, let $\mathbf{F} \in \mathbb{R}^{N_{\text{rODT}} \times N_{\text{col}}}$ be a matrix of feature occurrences across different rODTs. We then use the output of sub-module Δ_2, a vector $\vec{w} \in \mathbb{R}^{N_{\text{rODT}}}$ (Equation (6)), to represent the importance across all rODTs for each sample, as this weight $\vec{w}$ is used for constructing rODT forest to perform prediction in DOFEN model. A softmax

operation is further applied to the vector $\vec{w}$ to ensure the importance sums to 1 (also done in line 6 of Algorithm 1). Finally, we perform a weighted sum between the feature occurrences and the importance of each rODT, resulting in a single vector $\vec{t} \in \mathbb{R}^{N_{\text{col}}}$ representing DOFEN's feature importance for a specific sample. To calculate DOFEN's overall feature importance of a dataset, we simply average the feature importance of all samples in training dataset.

We tested the reliability of DOFEN's feature importance on three real-world datasets: the mushroom dataset, the red wine quality dataset, and the white wine quality dataset, following the experimental design used by Trompt. The results of these three datasets are shown in Tables 22 to 24, respectively. The results indicate that the top-3 important features identified by DOFEN align closely with those selected by other tree-based models, with only minor ranking differences. This demonstrates DOFEN's ability to reliably identify key features while maintaining interpretability despite its deep learning architecture. This further indicates that DOFEN may contain similar decision-making process as tree-based model does, as it is a tree-inspired deep neural network.

Table 22: Top 3 Feature importance of DOFEN on mushroom dataset.

	1st	**2nd**	**3rd**
Random Forest	odor (15.11%)	gill-size (12.37%)	gill-color (10.42 %)
XGBoost	spore-print-color (29.43%)	odor (22.71%)	cap-color (14.07%)
LightGBM	spore-print-color (22.08%)	gill-color (14.95%)	odor (12.96%)
CatBoost	odor (72.43%)	spore-print-color (10.57%)	gill-size (2.71%)
GradientBoostingTree	gill-color (31.08%)	spore-print-color (19.89%)	odor (17.44%)
Trompt	odor (24.93%)	gill-size (8.13%)	gill-color (5.73%)
DOFEN (ours)	odor (13.15%)	spore-print-color (6.84%)	gill-size (5.58%)

Table 23: Feature importance of DOFEN on red wine dataset.

	1st	**2nd**	**3rd**
Random Forest	alcohol (27.17%)	sulphates (15.44%)	volatile acidity (10.92%)
XGBoost	alcohol (35.42%)	sulphates (15.44%)	volatile acidity (7.56%)
LightGBM	alcohol (26.08%)	sulphates (15.75%)	volatile acidity (10.63%)
CatBoost	sulphates (16.29%)	alcohol (15.67%)	volatile acidity (10.40%)
GradientBoostingTree	alcohol (26.27%)	sulphates (16.24%)	volatile acidity (11.12%)
Trompt	alcohol (11.83%)	sulphates (10.94%)	total sulfur dioxide (9.78%)
DOFEN (ours)	alcohol (11.16%)	volatile acidity (10.77%)	sulphates (10.17%)

Table 24: Feature importance of DOFEN on white wine dataset.

	1st	**2nd**	**3rd**
Random Forest	alcohol (24.22%)	volatile acidity (12.44%)	free sulfur dioxide (11.78%)
XGBoost	alcohol (31.87%)	free sulfur dioxide (11.38%)	volatile acidity (10.05%)
LightGBM	alcohol (24.02%)	volatile acidity (12.47%)	free sulfur dioxide (11.45%)
CatBoost	alcohol (17.34%)	volatile acidity (12.07%)	free sulfur dioxide (11.47%)
GradientBoostingTree	alcohol (27.84%)	volatile acidity (13.59%)	free sulfur dioxide (12.87%)
Trompt	fixed acidity (10.91%)	volatile acidity (10.47%)	pH (10.37%)
DOFEN (ours)	alcohol (10.90%)	free sulfur dioxide (10.21%)	volatile acidity (10.01%)

F More Analysis

F.1 Sampling in Relaxed ODT Forest Ensemble

The forest ensemble in DOFEN is a level higher than common tree-based models. This section attempts to explore its impact on model performance by ablating this higher-level ensemble. The experiment involves using all the constructed rODTs from Equation (5) to form a single forest, without the sampling of weights and embeddings as described in line 6 in Algorithm 1. In practice,

we directly apply a softmax function to the weight vector $\vec{w}$ and calculate the weighted sum of corresponding embeddings $\mathbf{E}$. As a result, there is only one prediction for each sample, unlike N_{forest} predictions as in Algorithm 1.

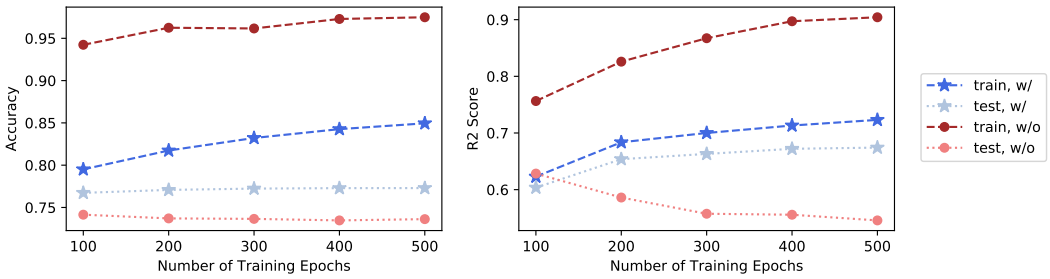
From the results in Table 25, it can be seen that with the ensemble of forests, the performance of DOFEN is improved across all datasets, independent of the types of tasks and features. The average improvement in classification datasets has reached 0.0363 in accuracy, while in regression datasets has reached 0.3367 in the R-squared score.

To further investigate the drastic drop in performance without using sampling in forest ensembles, we analyze performance at various training checkpoints. As shown in Figure 9, when sampling is not used in forest ensembles, training performance is significantly better compared to testing performance, and testing performance decreases with increasing training epochs, indicating an overfitting issue. Conversely, with an ensemble of multiple forests, both training and testing performance improve concurrently, thus mitigating the overfitting issue.

Based on the performance improvements from introducing a forest ensemble, we have applied an extra level of ensemble that combines multiple DOFEN models. However, the enhancement in performance is negligible. The detailed results can be found in Appendix F.2.

Table 25: Comparing DOFEN with and without sampling in forest ensemble

	w/ (default)	**w/o**
Classification	0.7725	0.7362
– Numerical Only	0.7920	0.7526
– Heterogeneous	0.7281	0.6988
Regression	0.6605	0.3238
– Numerical Only	0.6814	0.1867
– Heterogeneous	0.6371	0.4770



(a) Classification (b) Regression

Figure 9: Overfitting arises when not using sampling in the forest ensemble, affecting both (a) classification and (b) regression tasks. "Train" refers to training performance, and "Test" refers to testing performance. "w/" indicates the use of sampling to construct multiple forests, while "w/o" indicates the use of all constructed rODTs to form a single forest.

F.2 Seed Ensemble

In this section, we explore the results of applying an additional layer of bagging ensemble to the DOFEN model, a technique we denote as the seed ensemble. Specifically, we infer the trained DOFEN model using 15 different random seeds. This results in 15 distinct predictions due to the random sampling procedure conducted in the rODT Forest Construction module. Building upon these predictions, we average the 15 different predictions to create another layer of bagging ensemble, which we then present as the final prediction. This experiment is conducted on a medium-sized tabular benchmark. As shown in Table 26, this seed ensemble approach further enhances the performance of the DOFEN model, even with a small N_{forest}. The results further suggest that the DOFEN model can easily benefit from ensemble strategies, thanks to the random sampling procedure.

Table 26: Comparing evaluation performance with and without seed ensemble at varying N_{forest}.

	N_{forest}	**10**	**20**	**50**	**100 (default)**	**300**
w/o seed ensemble	Classification	0.7698	0.7713	0.7725	0.7725	0.7726
	Regression	0.6568	0.6586	0.6589	0.6605	0.6607
w/ seed ensemble	Classification	0.7727	<u>0.7732</u>	0.7731	0.7731	0.7731
	Regression	<u>0.6619</u>	<u>0.6619</u>	0.6616	0.6616	0.6616

F.3 An Alternative Strategy for Condition Selection

In the rODT Construction module, we implement a shuffle-then-reshape procedure to construct rODTs as outlined in Equation (5). The shuffle of matrix $\mathbf{M}$ serves as a straightforward approach to delivering a diverse set of condition combinations for subsequent segmentation. In this section, we aim to experiment with an alternative strategy for selecting columns to construct rODTs, which we adopt the column selection strategy used by CatBoost to form the rODTs in our DOFEN model, we denote this approach as 'Catboost-Init'. Specifically, we begin by selecting a machine learning algorithm that also employs ODT as the base element, namely, CatBoost. Subsequently, we train the CatBoost model and use the columns it identifies post-training to construct our rODTs. In the context of experimental configurations, to ensure a fair comparision, it is crucial to equate the capacity of CatBoost model with the default settings of DOFEN. To achieve this, we set the depth and number of boosting iterations of Catboost as the depth of an rODT (d) and the total number of rODT (N_{rODT}) in DOFEN, respectively. The CatBoost trained based on these configurations is denoted as 'CatBoost*' in the context.

The results are presented in Table 27. As can be seen, the 'Catboost-Init' approach achieves performance comparable to the 'CatBoost*' approach and generally outperforms the default shuffle approach. This indicates that designing a more sophisticated approach for rODT construction indeed contributes to better performance. However, adopting a selection strategy from a tree-based model results in a two-stage modeling process, which contradicts our goal of designing an end-to-end differentiable DNN model. This intriguing insight leads us to consider a more innovative end-to-end condition selection approach, which we will explore in future work.

Table 27: Comparing the column selection strategy of DOFEN.

	Shuffle (default)	**Catboost-Init**	**Catboost***
Classification	0.7725	<u>0.7769</u>	0.7722
Regression	0.6604	0.6792	<u>0.6811</u>

F.4 An Alternative Strategy for Weight Selection

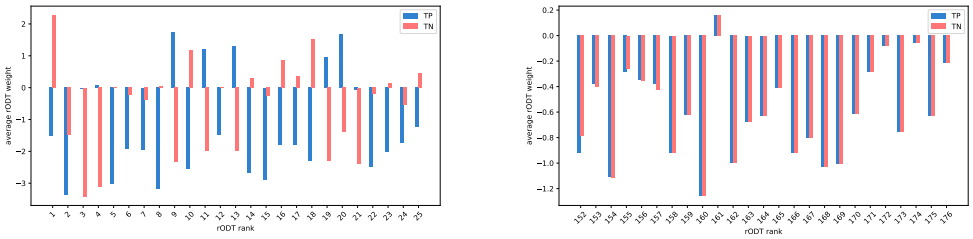
In the Two-level rODT Ensemble module, our default method involves random sampling of weights. We contrast this approach with an alternative weight selection strategy, denoted as sliding window selection. The method creates N_{forest} windows, each containing $N_{\text{estimator}}$ weights. These windows are then evenly distributed across N_{rODT} weights. Within each window, the weights are collectively treated as a rODT forest. As shown in Table 28, our results indicate that random sampling yields better performance compared to the sliding window selection. This finding substantiates that the random sampling approach already works well compared to a sophisticated alternative. Investigating a more advanced and effective approach is worth considering in future work.

Table 28: The comparison of random sampling and sliding window selection of weights.

	random sampling (default)	**sliding window selection**
Classification	<u>0.7725</u>	0.7716
Regression	<u>0.6605</u>	0.6593

F.5 More Experiments for Section 4.3.2 (Activated rODT for Different Classes)

In this section, we aim to test whether the findings presented in Section 4.3.2 can be replicated on another dataset. Consequently, we have chosen another binary classification dataset (compass-two-years) and repeated the experiment using the same settings. The experimental results are depicted in Figure 10. As shown in Figure 10a, weights with higher standard deviation exhibit distinct distributions between true positive and true negative samples. Conversely, Figure 10b illustrates an opposite trend. Overall, conducting the experiment on another dataset further validates our findings.



(a) Relaxed ODTs with large weight variation.

(b) Relaxed ODTs with small weight variation.

Figure 10: In the compass dataset, the weights w_i of rODT are sorted based on the standard deviation calculated across true positive (TP) and true negative (TN) samples in the testing data. Figure 10a shows that the weights of TP samples differ significantly from those of TN samples when the standard deviation of the weights is higher. Conversely, Figure 10b reveals contrasting results for weights with a lower standard deviation.

F.6 Pruning of Relaxed ODT

Following Section 4.3.2, in this section, we aim to examine the performance change after pruning weights with small standard deviations and their corresponding embeddings.

Table 29 shows the performance under different pruning ratios. The column labeled 'by dataset' indicates that we tailored the pruning ratio for each dataset based on its validation data. As shown in Table 29, pruning these rODTs does not negatively affect performance. In fact, a minor degree of pruning can actually enhance performance, with the optimal pruning ratio being 0.02 for classification datasets and 0.1 for regression datasets. Notice that the 'by dataset' approach is better suited to real-world scenarios, even though it does not always yield the best performance.

Table 29: Pruning of rODT with varying ratio. Weights w_i with *lower* standard deviation are pruned.

Ratio	**0.0 (default)**	**0.02**	**0.1**	**0.2**	**by dataset**
Classification	0.7725	<u>0.7733</u>	0.7726	0.7709	0.7732
Regression	0.6605	0.6629	0.6630	0.6621	<u>0.6657</u>

We then investigate the outcomes when weights with higher standard deviations are pruned. Consequently, we sort the weights and prune them from the higher end. The results, presented in Table 30, show that the performance in both classification and regression tasks monotonically drops as the prune ratio increases. This finding suggests that the standard deviation of weights is a good indicator of their importance in making predictions. It further validates why pruning weights with lower standard deviation does not harm performance and, in some cases, even helps.

Table 30: rODT pruning with varying ratio. Weights w_i with *higher* standard deviation are pruned.

Ratio	**0.0(default)**	**0.02**	**0.05**	**0.10**	**0.2**
Classification	<u>0.7725</u>	<u>0.7725</u>	0.7715	0.7667	0.763
Regression	<u>0.6605</u>	0.6571	0.6484	0.6383	0.601

In addition, we discuss another, potentially more straightforward, pruning approach. Specifically, we prune the weights w_i based on their average value across samples. Similar to the experiments that use standard deviation as the metric for pruning, this time we sort the weights by their average. We then attempt to prune the weights from both the top and bottom ends. The results are provided in Table 31 and Table 32, suggesting that the value of weights is not an effective indicator for pruning. Although there is some improvement in performance at a low ratio, this approach generally diminishes performance with larger ratios, regardless of whether the weights are pruned from the higher or lower end.

Table 31: rODT pruning with varying ratio. Weights w_i with *lower* average value are pruned.

Ratio	**0.0(default)**	**0.02**	**0.05**	**0.10**	**0.2**
Classification	0.7725	0.773	0.7715	0.7722	0.7703
Regression	0.6605	0.6611	0.6592	0.6575	0.6425

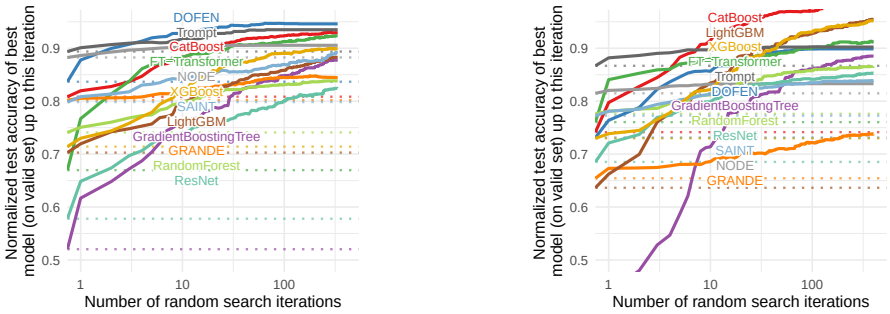
Table 32: rODT pruning with varying ratio. Weights w_i with *higher* average value are pruned.

Ratio	**0.0(default)**	**0.02**	**0.05**	**0.10**	**0.2**
Classification	0.7725	0.7731	0.7725	0.7704	0.7643
Regression	0.6605	0.6619	0.6573	0.6352	0.4881

G More Evaluation Results on Tabular Benchmark

G.1 Performance Evaluation on Large-sized Benchmark

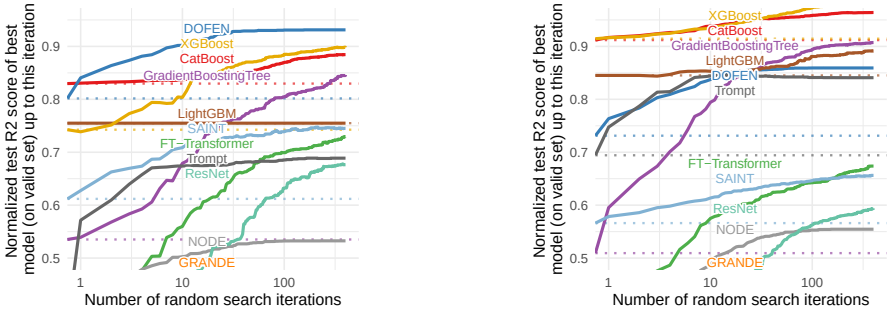
This section discusses the evaluation results on large-sized classification and regression tasks. Overall, the results demonstrate a similar trend as the medium-sized tabular benchmark. Notably, DOEFN achieves the top ranks in both tasks with numerical features.



(a) Exclusively Numerical (b) Heterogeneous

Figure 11: Results on **large-sized classification** datasets.

Classification. In Figure 11a, DOFEN even surpasses CatBoost to become the top performer. Conversely, in Figure 11b, CatBoost clearly outperforms other models. FT-Transformer, Trompt and DOFEN are the best-performing tabular DNN models, though they rank in the middle among all models. As a result, with the current development of tabular DNN models, their performance in processing numerical features is already on par with or even surpass that of tree-based models, and they are more advantageous for large-sized datasets. However, DOFEN and other DNN models are still less efficient in handling heterogeneous features.



(a) Large; Numerical

(b) Large; Heterogeneous

Figure 12: Results on **large-sized regression** datasets.

Regression. In Figure 12a, the leading models remain DOFEN, XGBoost, and CatBoost. DOFEN's proficiency in handling numerical features, further enhanced by the increased data volume, enables it to secure the top position once again. In Figure 12b, DOFEN and Trompt barely maintain their positions within the leading group, yet they still stand out from the other DNN models.

G.2 Detailed Evaluation Results

In the main paper, we have discussed the overall performance of DOFEN. To simplify tables, we map dataset names with their OpenML ID, as described in Appendix B.3. The evaluation results of each task are organized in Table 33. Please refer to the detailed figures and tables for each task of your interest. The evaluation metrics are accuracy for classification tasks and R^2 score for regression tasks, consistent with our main paper. Furthermore, we calculate the mean and standard deviation of ranks across datasets to provide the rank for each model in the tables.

Table 33: Tables and figures for each task.

Task	Feature	Figure	Table
medium-sized classification	numerical	Figure 13	Tables 34 and 35
	heterogeneous	Figure 14	Table 36
medium-sized regression	numerical	Figure 15	Tables 37 and 38
	heterogeneous	Figure 16	Tables 39 and 40
large-sized classification	numerical	Figure 17	Table 41
	heterogeneous	Figure 18	Table 42
large-sized regression	numerical	Figure 19	Table 43
	heterogeneous	Figure 20	Table 44

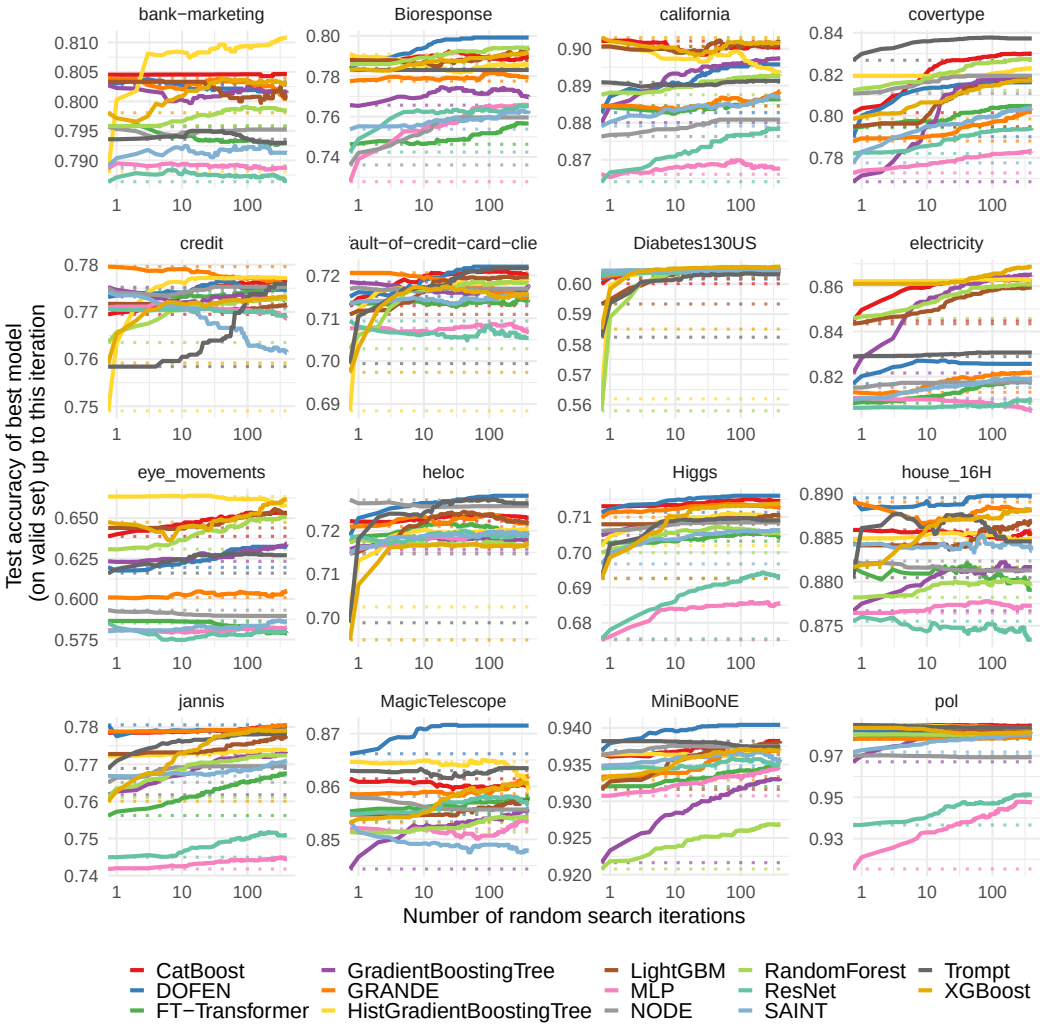


Figure 13: Results on each **medium-sized classification** datasets with only **numerical** features.

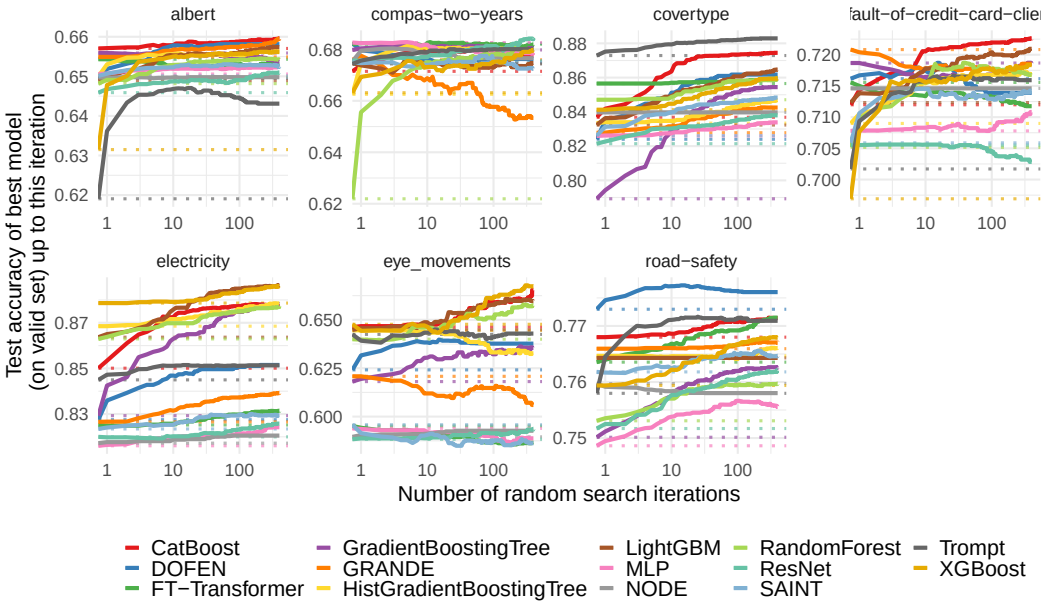


Figure 14: Results on each **medium-sized classification** datasets with **heterogeneous** features.

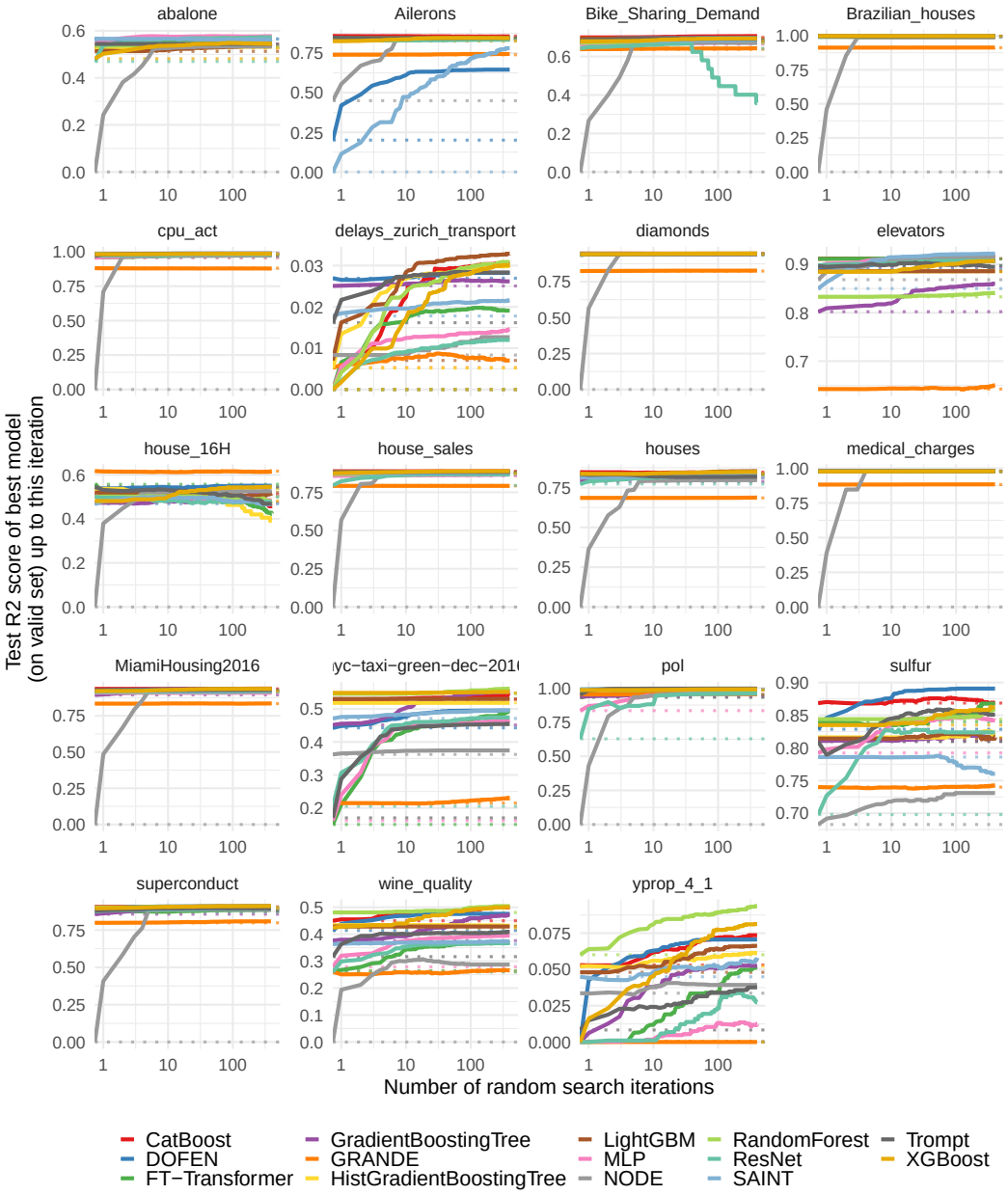


Figure 15: Results on each **medium-sized regression** datasets with **numerical** features.

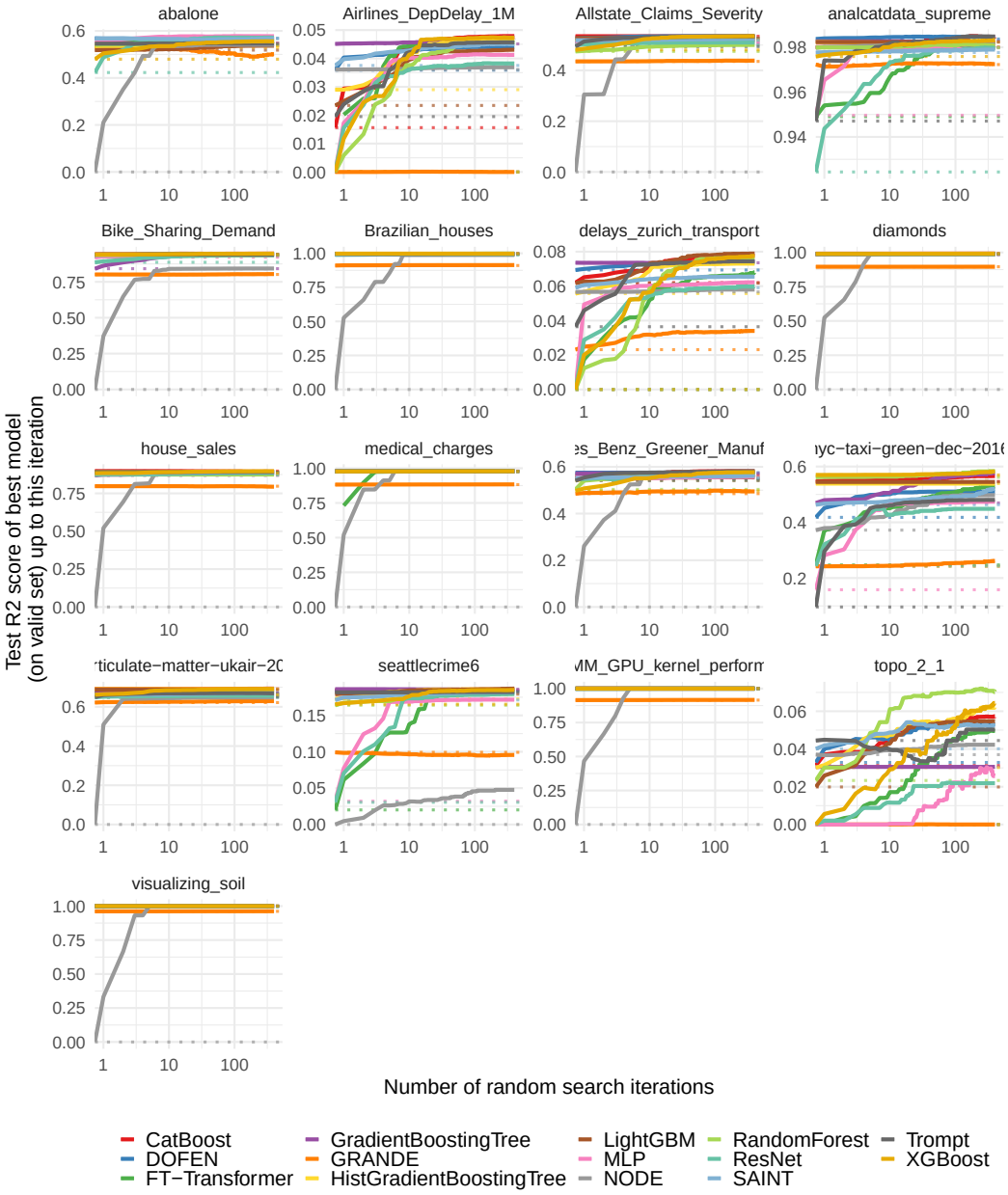


Figure 16: Results on each **medium-sized regression** datasets with **heterogeneous** features.

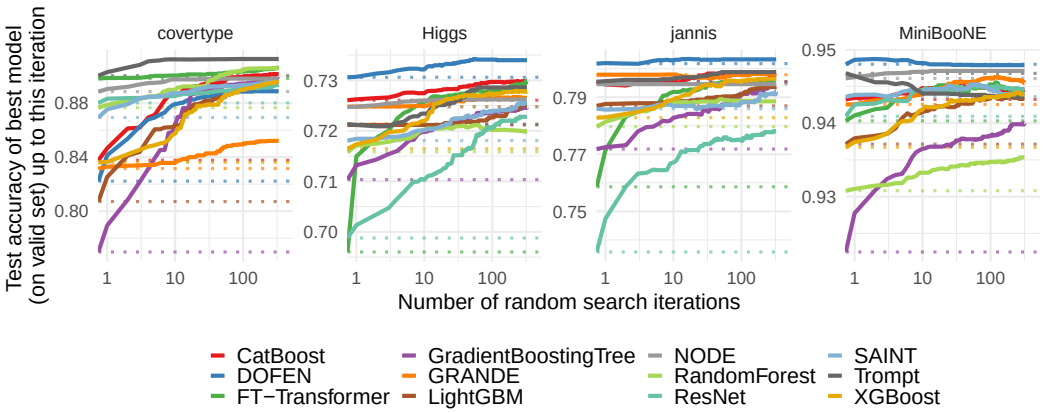


Figure 17: Results on each **large-sized classification** datasets with only **numerical** features.

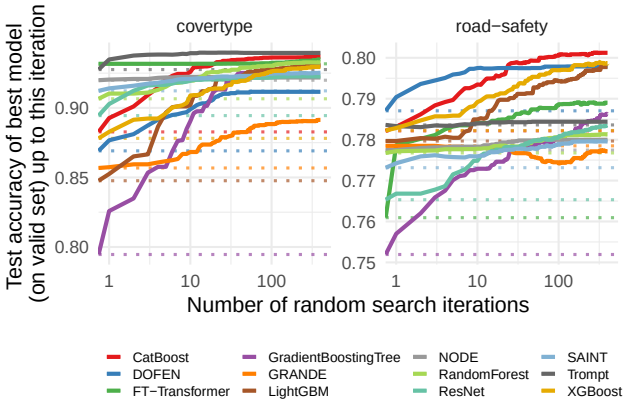


Figure 18: Results on each **large-sized classification** datasets with **heterogeneous** features.

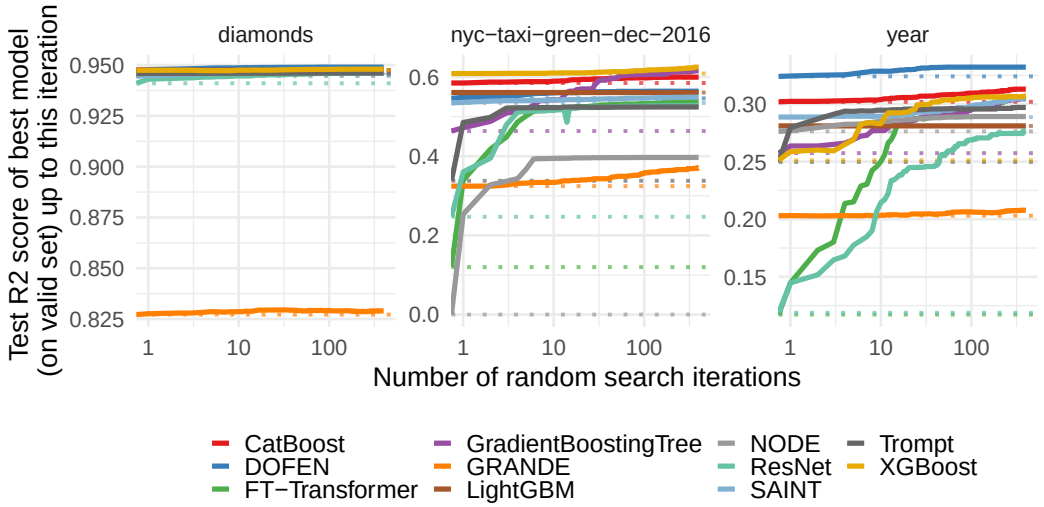


Figure 19: Results on each **large-sized regression** datasets with **numerical** features.

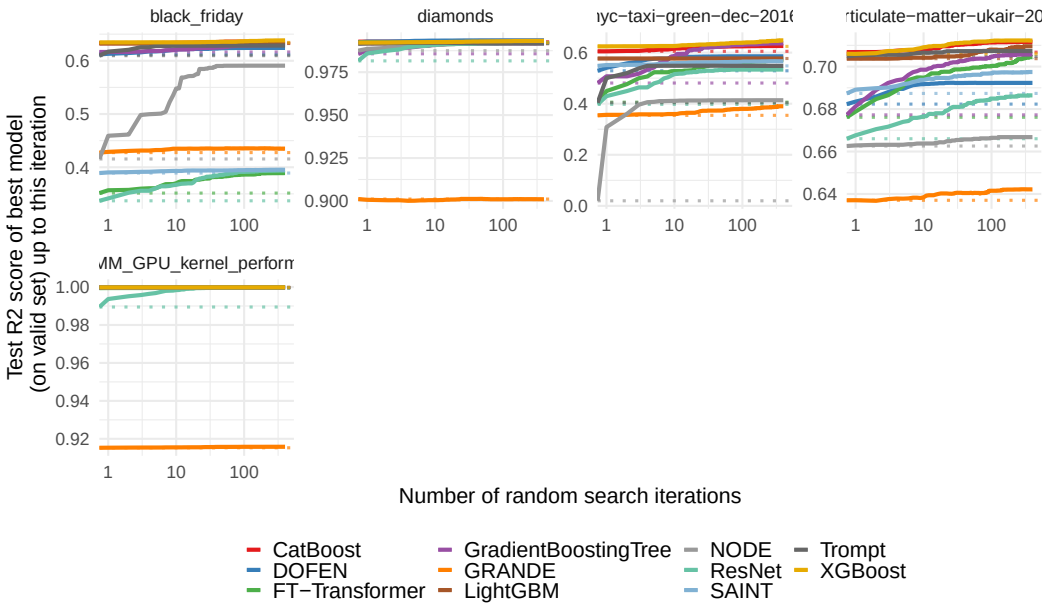


Figure 20: Results on each **large-sized regression** datasets with **heterogeneous** features.

Table 34: The performance of **medium-sized classification** task (*numerical features only*) (1).

	361276	**361273**	**361069**	**361065**	**361068**	**361066**	**361277**	**361061**	**361055**	**361275**
Default										
DOFEN (ours)	0.7839	0.6016	0.7113	0.8662	0.9369	0.8030	0.8827	0.7901	0.7732	0.7151
Trompt	0.7831	0.5823	0.6926	0.8630	0.9382	0.7936	0.8909	0.8268	0.7584	0.6994
GRANDE	0.7776	0.6023	0.7099	0.8586	0.9334	0.8039	0.8845	0.7880	0.7796	0.7206
FT-Transformer	0.7463	0.6025	0.7031	0.8553	0.9320	0.7958	0.8846	0.7944	0.7745	0.7137
ResNet	0.7424	0.6029	0.6755	0.8548	0.9345	0.7864	0.8641	0.7820	0.7706	0.7093
MLP	0.7277	0.6033	0.6752	0.8520	0.9307	0.7886	0.8661	0.7727	0.7710	0.7077
SAINT	0.7537	0.6044	0.6967	0.8534	0.9348	0.7891	0.8791	0.7775	0.7741	0.7133
NODE	0.7360	0.6039	0.7060	0.8581	0.9363	0.7957	0.8763	0.8108	0.7750	0.7169
CatBoost	0.7881	0.6001	0.7130	0.8614	0.9364	0.8045	0.9021	0.8016	0.7695	0.7129
LightGBM	0.7878	0.5934	0.7079	0.8547	0.9316	0.8033	0.9006	0.7950	0.7717	0.7109
XGBoost	0.7831	0.5850	0.6925	0.8531	0.9329	0.7981	0.9030	0.7987	0.7591	0.6974
HistGradientBoostingTree	0.7909	0.5619	0.7018	0.8647	0.9364	0.7880	0.9007	0.8193	0.7490	0.6884
GradientBoostingTree	0.7657	0.6018	0.7048	0.8444	0.9216	0.8027	0.8800	0.7685	0.7752	0.7184
RandomForest	0.7859	0.5579	0.6998	0.8514	0.9208	0.7958	0.8876	0.8124	0.7635	0.7029
Searched										
DOFEN (ours)	0.7992	0.6043	0.7160	0.8715	0.9404	0.8017	0.8958	0.8162	0.7747	0.7220
Trompt	0.7831	0.6032	0.7090	0.8635	0.9374	0.7930	0.8913	0.8373	0.7760	0.7217
GRANDE	0.7795	0.6042	0.7125	0.8620	0.9368	0.8005	0.8885	0.8021	0.7759	0.7150
FT-Transformer	0.7566	0.6044	0.7042	0.8582	0.9350	0.7929	0.8865	0.8048	0.7754	0.7145
ResNet	0.7652	0.6046	0.6931	0.8565	0.9357	0.7866	0.8783	0.7942	0.7692	0.7054
MLP	0.7658	0.6033	0.6855	0.8535	0.9346	0.7888	0.8676	0.7834	0.7686	0.7067
SAINT	0.7629	0.6044	0.7063	0.8478	0.9355	0.7912	0.8870	0.8041	0.7613	0.7155
NODE	0.7596	0.6043	0.7084	0.8556	0.9375	0.7952	0.8808	0.8194	0.7753	0.7169
CatBoost	0.7898	0.6052	0.7144	0.8599	0.9382	0.8047	0.9003	0.8299	0.7768	0.7201
LightGBM	0.7942	0.6050	0.7105	0.8572	0.9378	0.8011	0.9017	0.8188	0.7715	0.7196
XGBoost	0.7917	0.6057	0.7138	0.8606	0.9369	0.8031	0.9016	0.8176	0.7732	0.7156
HistGradientBoostingTree	0.7859	0.6050	0.7092	0.8592	0.9372	0.8108	0.8932	0.8231	0.7772	0.7194
GradientBoostingTree	0.7694	0.6044	0.7100	0.8557	0.9331	0.8015	0.8974	0.8182	0.7728	0.7172
RandomForest	0.7936	0.6047	0.7053	0.8541	0.9269	0.7985	0.8924	0.8275	0.7727	0.7182

Table 35: The performance of **medium-sized classification** task (*numerical features only*) (2).

	361060	**361070**	**361278**	**361063**	**361274**	**361062**	**Ranking**
Default							
DOFEN (ours)	0.8169	0.6196	0.7189	<u>0.8895</u>	<u>0.7806</u>	0.9822	4.81 ± 3.42
Trompt	0.8289	0.6160	0.6987	0.8805	0.7689	<u>0.9849</u>	7.41 ± 4.22
GRANDE	0.8131	0.6010	0.7210	0.8890	0.7788	0.9783	5.38 ± 3.61
FT-Transformer	0.8082	0.5864	0.7175	0.8816	0.7562	0.9780	8.19 ± 2.62
ResNet	0.8062	0.5852	0.7186	0.8755	0.7449	0.9366	10.75 ± 3.87
MLP	0.8105	0.5808	0.7151	0.8765	0.7418	0.9153	11.56 ± 3.82
SAINT	0.8098	0.5799	0.7146	0.8842	0.7668	0.9718	8.88 ± 3.56
NODE	0.8151	0.5931	<u>0.7271</u>	0.8823	0.7651	0.9701	6.75 ± 3.62
CatBoost	0.8448	0.6387	0.7222	0.8859	0.7785	0.9846	3.91 ± 3.55
LightGBM	0.8434	0.6439	0.7148	0.8843	0.7727	0.9838	6.06 ± 2.96
XGBoost	0.8611	0.6475	0.6948	0.8816	0.7600	0.9835	8.12 ± 4.05
HistGradientBoostingTree	<u>0.8623</u>	<u>0.6633</u>	0.7024	0.8848	0.7721	0.9846	6.22 ± 5.10
GradientBoostingTree	0.8216	0.6233	0.7157	0.8767	0.7618	0.9671	8.50 ± 3.88
RandomForest	0.8458	0.6308	0.7173	0.8782	0.7611	0.9803	8.47 ± 3.69
Searched							
DOFEN (ours)	0.8257	0.6323	<u>0.7281</u>	<u>0.8898</u>	0.7791	0.9802	4.41 ± 4.10
Trompt	0.8307	0.6271	0.7263	0.8846	0.7782	0.9838	5.69 ± 3.57
GRANDE	0.8217	0.6050	0.7217	0.8881	<u>0.7805</u>	0.9792	7.31 ± 3.37
FT-Transformer	0.8191	0.5786	0.7194	0.8791	0.7675	0.9836	10.06 ± 3.29
ResNet	0.8097	0.5801	0.7189	0.8734	0.7509	0.9511	11.88 ± 3.26
MLP	0.8048	0.5823	0.7187	0.8773	0.7444	0.9474	12.94 ± 3.11
SAINT	0.8188	0.5859	0.7194	0.8835	0.7709	0.9803	10.44 ± 2.62
NODE	0.8175	0.5895	0.7257	0.8813	0.7694	0.9693	9.09 ± 3.14
CatBoost	0.8627	0.6532	0.7230	0.8855	0.7802	<u>0.9846</u>	2.88 ± 2.96
LightGBM	0.8594	0.6526	0.7217	0.8869	0.7777	0.9819	5.00 ± 2.57
XGBoost	<u>0.8687</u>	<u>0.6615</u>	0.7171	0.8882	0.7790	0.9815	4.75 ± 3.67
HistGradientBoostingTree	0.8625	0.6578	0.7203	0.8849	0.7739	0.9835	4.69 ± 2.67
GradientBoostingTree	0.8653	0.6343	0.7179	0.8817	0.7735	0.9813	7.69 ± 2.74
RandomForest	0.8608	0.6506	0.7164	0.8798	0.7724	0.9812	8.19 ± 3.51

Table 36: The performance of **medium-sized classification** task (*heterogeneous features*).

	361282	**361286**	**361113**	**361283**	**361110**	**361111**	**361285**	**Ranking**
Default								
DOFEN (ours)	0.6495	0.6823	0.8240	0.7162	0.8275	0.6241	<u>0.7730</u>	6.00 ± 4.17
Trompt	0.6191	0.6743	<u>0.8729</u>	0.7017	0.8450	0.6425	0.7580	8.43 ± 4.43
GRANDE	0.6549	0.6759	0.8278	<u>0.7208</u>	0.8266	0.6208	0.7659	5.43 ± 4.03
FT-Transformer	0.6543	0.6820	0.8565	0.7156	0.8252	0.5952	0.7635	5.71 ± 3.93
ResNet	0.6459	0.6756	0.8214	0.7055	0.8200	0.5883	0.7517	11.43 ± 4.39
MLP	0.6506	<u>0.6826</u>	0.8259	0.7078	0.8161	0.5939	0.7486	9.43 ± 5.21
SAINT	0.6501	0.6750	0.8261	0.7059	0.8234	0.5958	0.7618	8.86 ± 2.70
NODE	0.6497	0.6753	0.8397	0.7146	0.8172	0.5895	0.7597	8.43 ± 3.28
CatBoost	<u>0.6570</u>	0.6715	0.8369	0.7120	0.8501	0.6462	0.7680	4.86 ± 4.60
LightGBM	0.6489	0.6747	0.8323	0.7123	0.8637	0.6448	0.7643	6.29 ± 2.92
XGBoost	0.6315	0.6632	0.8413	0.6969	<u>0.8786</u>	<u>0.6477</u>	0.7594	7.71 ± 5.24
HistGradientBoostingTree	0.6500	0.6625	0.8334	0.7090	0.8685	0.6446	0.7647	6.43 ± 3.45
GradientBoostingTree	0.6559	0.6798	0.7892	0.7187	0.8293	0.6181	0.7501	7.29 ± 4.60
RandomForest	0.6482	0.6219	0.8471	0.7052	0.8629	0.6400	0.7531	8.71 ± 4.32
Searched								
DOFEN (ours)	0.6581	0.6763	0.8618	0.7139	0.8515	0.6377	<u>0.7760</u>	6.14 ± 3.96
Trompt	0.6431	0.6801	<u>0.8829</u>	0.7159	0.8513	0.6429	0.7709	6.29 ± 4.00
GRANDE	0.6596	0.6534	0.8425	0.7186	0.8393	0.6060	0.7670	7.79 ± 4.03
FT-Transformer	0.6529	0.6819	0.8594	0.7118	0.8315	0.5865	0.7715	7.71 ± 4.21
ResNet	0.6510	<u>0.6843</u>	0.8380	0.7028	0.8258	0.5937	0.7621	10.43 ± 4.63
MLP	0.6524	0.6788	0.8339	0.7108	0.8242	0.5888	0.7556	12.00 ± 4.50
SAINT	0.6525	0.6724	0.8483	0.7142	0.8294	0.5862	0.7647	10.71 ± 3.37
NODE	0.6498	0.6766	0.8397	0.7146	0.8205	0.5927	0.7580	11.71 ± 3.82
CatBoost	<u>0.6596</u>	0.6775	0.8745	<u>0.7226</u>	0.8773	0.6655	0.7714	3.29 ± 4.63
LightGBM	0.6574	0.6747	0.8647	0.7209	<u>0.8864</u>	0.6596	0.7643	4.86 ± 4.53
XGBoost	0.6561	0.6798	0.8596	0.7183	0.8861	<u>0.6673</u>	0.7679	4.14 ± 3.58
HistGradientBoostingTree	0.6563	0.6786	0.8467	0.7169	0.8785	0.6327	0.7661	6.71 ± 2.17
GradientBoostingTree	0.6538	0.6805	0.8545	0.7186	0.8780	0.6355	0.7625	6.21 ± 2.84
RandomForest	0.6542	0.6795	0.8587	0.7167	0.8773	0.6575	0.7597	7.00 ± 2.27

Table 37: The performance of **medium-sized regression** task (*numerical features only*) (1).

	361077	**361082**	**361081**	**361087**	**361280**	**361072**	**361281**	**361080**	**361074**	**361079**
Default										
DOFEN (ours)	0.2011	0.6874	0.9931	0.9194	0.5651	0.9837	0.0269	0.9352	0.8921	0.5398
Trompt	0.8480	0.6829	0.9970	0.9275	0.5443	0.9723	0.0162	0.9415	0.8969	0.5453
GRANDE	0.7399	0.6376	0.9126	0.8340	0.5281	0.8812	0.0053	0.8239	0.6423	0.6173
FT-Transformer	0.8436	0.6691	0.9958	0.9205	0.5308	0.9594	0.0000	0.9419	0.9115	0.5571
ResNet	0.8331	0.6423	0.9923	0.9145	0.4696	0.9747	0.0000	0.9404	0.8979	0.4995
MLP	0.8299	0.6634	0.9939	0.9091	0.5435	0.9570	0.0000	0.9411	0.8958	0.5057
SAINT	0.0000	0.6816	0.9938	0.9158	0.5658	0.9835	0.0178	0.9422	0.8500	0.4679
NODE	0.4500	0.0000	0.0000	0.0000	0.0000	0.0000	0.0083	0.0000	0.8684	0.0000
CatBoost	0.8576	0.6993	0.9960	0.9356	0.5279	0.9856	0.0000	0.9457	0.9117	0.5101
LightGBM	0.8468	0.6928	0.9938	0.9225	0.5124	0.9846	0.0070	0.9449	0.8859	0.5195
XGBoost	0.8258	0.6793	0.9976	0.9203	0.4817	0.9825	0.0000	0.9409	0.8848	0.4814
HistGradientBoostingTree	0.8464	0.6932	0.9938	0.9233	0.5259	0.9828	0.0052	0.9448	0.8855	0.5361
GradientBoostingTree	0.8397	0.6758	0.9962	0.8942	0.5399	0.9835	0.0251	0.9441	0.8022	0.4733
RandomForest	0.8372	0.6720	0.9931	0.9141	0.5359	0.9826	0.0000	0.9394	0.8330	0.5016
Searched										
DOFEN (ours)	0.6458	0.6972	0.9941	0.9351	0.5637	0.9872	0.0282	0.9447	0.9068	0.5507
Trompt	0.8457	0.6915	0.9956	0.9280	0.5443	0.9873	0.0283	0.9427	0.8948	0.4650
GRANDE	0.7449	0.6426	0.9136	0.8352	0.5457	0.8786	0.0070	0.8270	0.6498	0.6165
FT-Transformer	0.8453	0.6805	0.9973	0.9223	0.5630	0.9844	0.0191	0.9435	0.9171	0.4214
ResNet	0.8342	0.3569	0.9969	0.9172	0.5731	0.9822	0.0120	nan	0.9079	0.4781
MLP	0.8367	0.6754	0.9932	0.9092	0.5776	0.9790	0.0146	0.9436	0.9181	0.4830
SAINT	0.7811	0.6858	0.9940	0.9245	0.5629	0.9849	0.0216	0.9442	0.9224	0.4660
NODE	0.8365	0.6704	0.9877	0.9260	0.5332	0.9730	0.0127	0.9427	0.9148	0.5257
CatBoost	0.8553	0.7062	0.9920	0.9377	0.5353	0.9865	0.0306	0.9450	0.9105	0.4586
LightGBM	0.8468	0.6928	0.9928	0.9339	0.5399	0.9811	0.0329	0.9449	0.8859	0.5167
XGBoost	0.8450	0.6943	0.9976	0.9360	0.5449	0.9861	0.0301	0.9456	0.9072	0.5454
HistGradientBoostingTree	0.8464	0.6932	0.9928	0.9265	0.5322	0.9745	0.0299	0.9449	0.8863	0.3912
GradientBoostingTree	0.8416	0.6889	0.9961	0.9249	0.5457	0.9854	0.0262	0.9450	0.8602	0.5158
RandomForest	0.8386	0.6871	0.9931	0.9242	0.5517	0.9829	0.0308	0.9453	0.8410	0.4806

Table 38: The performance of **medium-sized regression** task (*numerical features only*) (2).

	361084	**361078**	**361086**	**361083**	**361073**	**361085**	**361088**	**361076**	**361279**	**Ranking**
Default										
DOFEN (ours)	0.8723	0.8099	0.9756	0.4427	0.9885	0.8285	0.8950	0.4139	0.0000	6.84 ± 3.68
Trompt	0.8804	0.8352	0.9788	0.1699	0.9513	0.8096	0.8791	0.3168	0.0083	5.68 ± 3.30
GRANDE	0.7918	0.6854	0.8822	0.2149	0.9536	0.7400	0.7986	0.2609	0.0000	11.11 ± 3.70
FT-Transformer	0.8766	0.8235	0.9794	0.1499	0.9313	0.8400	0.8751	0.2648	0.0000	7.24 ± 3.77
ResNet	0.7948	0.7729	0.9772	0.2050	0.6279	0.6979	0.8739	0.2598	0.0000	10.71 ± 2.89
MLP	0.8575	0.8133	0.9789	0.1615	0.8343	0.7922	0.8842	0.2792	0.0000	9.08 ± 3.01
SAINT	0.8731	0.8139	0.9788	0.4713	0.9904	0.7859	0.8909	0.3632	0.0449	7.00 ± 3.49
NODE	0.0000	0.0000	0.0000	0.3622	0.0000	0.6828	0.0000	0.0000	0.0336	12.53 ± 3.80
CatBoost	0.8873	0.8472	0.9782	0.5291	0.9863	0.8685	0.9051	0.4500	0.0530	3.34 ± 4.03
LightGBM	0.8812	0.8351	0.9785	0.5306	0.9870	0.8143	0.8979	0.4286	0.0480	4.74 ± 2.98
XGBoost	0.8743	0.8374	0.9773	0.5487	0.9850	0.8349	0.8955	0.4237	0.0000	7.08 ± 3.36
HistGradientBoostingTree	0.8816	0.8325	0.9785	0.5186	0.9865	0.8161	0.8964	0.4336	0.0522	5.00 ± 2.64
GradientBoostingTree	0.8617	0.7874	0.9794	0.4516	0.9349	0.8106	0.8563	0.3763	0.0000	7.74 ± 3.62
RandomForest	0.8689	0.8270	0.9768	0.5460	0.9867	0.8439	0.9011	0.4807	0.0601	6.92 ± 3.81
Searched										
DOFEN (ours)	0.8824	0.8395	0.9788	0.4950	0.9928	0.8907	0.9091	0.4757	0.0706	5.26 ± 3.53
Trompt	0.8832	0.8187	0.9792	0.4550	0.9958	0.8508	0.8949	0.4091	0.0376	7.26 ± 3.42
GRANDE	0.7918	0.6877	0.8840	0.2305	0.9541	0.7425	0.8082	0.2654	0.0000	12.74 ± 4.12
FT-Transformer	0.8824	0.8341	0.9795	0.4926	0.9949	0.8676	0.8870	0.3675	0.0519	7.16 ± 3.62
ResNet	0.8668	0.8236	0.9793	0.4743	0.9606	0.8237	0.8934	0.3666	0.0273	9.50 ± 3.70
MLP	0.8669	0.8202	0.9796	0.4666	0.9708	0.8418	0.8930	0.3949	0.0126	8.84 ± 3.82
SAINT	0.8804	0.8259	0.9794	0.4958	0.9948	0.7603	0.8937	0.3736	0.0579	7.63 ± 2.91
NODE	0.8762	0.7969	0.9782	0.3743	0.9580	0.7309	0.8857	0.2874	0.0393	11.11 ± 3.57
CatBoost	0.8872	0.8487	0.9793	0.5404	0.9908	0.8692	0.9095	0.4996	0.0736	4.42 ± 4.06
LightGBM	0.8863	0.8539	0.9785	0.5306	0.9870	0.8172	0.9048	0.4286	0.0663	6.53 ± 3.59
XGBoost	0.8886	0.8495	0.9787	0.5519	0.9909	0.8620	0.9102	0.5006	0.0814	3.74 ± 3.57
HistGradientBoostingTree	0.8819	0.8375	0.9791	0.5186	0.9871	0.8203	0.9010	0.4313	0.0612	7.79 ± 3.12
GradientBoostingTree	0.8828	0.8400	0.9794	0.5531	0.9896	0.8153	0.9030	0.4710	0.0525	6.37 ± 2.64
RandomForest	0.8712	0.8291	0.9789	0.5618	0.9891	0.8585	0.9087	0.5044	0.0937	6.42 ± 3.65

Table 39: The performance of **medium-sized regression** task (*heterogeneous features*) (1).

	361293	**361292**	**361099**	**361098**	**361097**	**361104**	**361288**	**361093**	**361291**	**361096**
Default										
DOFEN (ours)	0.0359	0.5258	0.9341	0.9932	0.5750	0.9997	0.5686	0.9837	0.0694	0.9869
Trompt	0.0195	0.4939	0.9393	0.9963	0.5409	0.9996	0.5459	0.9470	0.0364	0.9888
GRANDE	0.0000	0.4356	0.8014	0.9101	0.4853	0.9151	0.5354	0.9723	0.0231	0.8957
FT-Transformer	nan	0.5160	0.9280	0.9960	0.5540	0.9997	0.5480	0.9490	0.0000	0.9872
ResNet	0.0000	0.4993	0.8861	0.9883	0.5470	0.9975	0.4229	0.9244	0.0000	0.9857
MLP	0.0000	0.5105	0.9213	0.9942	0.5546	0.9998	0.5486	0.9497	0.0000	0.9861
SAINT	0.0375	0.5191	0.9375	0.9930	0.5522	0.9990	0.5676	0.9777	0.0591	0.9867
NODE	0.0361	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.9797	0.0566	0.0000
CatBoost	0.0156	0.5347	0.9421	0.9959	0.5633	0.9997	0.5375	0.9801	0.0621	0.9911
LightGBM	0.0234	0.5275	0.9402	0.9938	0.5477	0.9997	0.5183	0.9823	0.0618	0.9901
XGBoost	0.0000	0.4807	0.9393	0.9976	0.4968	0.9997	0.4797	0.9759	0.0000	0.9896
HistGradientBoostingTree	0.0290	0.5226	0.9410	0.9939	0.5421	0.9997	0.5289	0.9823	0.0557	0.9908
GradientBoostingTree	0.0451	0.5066	0.8415	0.9962	0.5717	0.9997	0.5470	0.9827	0.0736	0.9842
RandomForest	0.0000	0.4748	0.9369	0.9929	0.5034	0.9998	0.5407	0.9799	0.0000	0.9878
Searched										
DOFEN (ours)	0.0440	0.5351	0.9453	0.9946	0.5755	0.9998	0.5641	0.9846	0.0738	0.9903
Trompt	0.0457	0.5330	0.9397	0.9957	0.5816	0.9997	0.5465	0.9848	0.0744	0.9899
GRANDE	0.0000	0.4386	0.8041	0.9137	0.4947	0.9166	0.5010	0.9725	0.0340	0.8957
FT-Transformer	0.0455	0.5217	0.9390	0.9979	0.5662	0.9998	0.5649	0.9797	0.0680	0.9899
ResNet	0.0382	0.5100	0.9361	0.9962	0.5685	0.9996	0.5712	0.9797	0.0599	nan
MLP	0.0413	0.5157	0.9337	0.9948	0.5572	0.9998	0.5775	0.9801	0.0622	0.9876
SAINT	0.0450	0.5253	0.9409	0.9958	0.5618	0.9997	0.5678	0.9789	0.0656	0.9893
NODE	0.0369	0.5169	0.8428	0.9878	0.5735	0.9998	0.5409	0.9823	0.0580	0.9860
CatBoost	0.0479	0.5350	0.9468	0.9921	0.5815	0.9997	0.5449	0.9847	0.0778	0.9917
LightGBM	0.0431	0.5336	0.9439	0.9941	0.5549	0.9997	0.5370	0.9823	0.0789	0.9907
XGBoost	0.0473	0.5352	0.9466	0.9976	0.5771	0.9998	0.5558	0.9832	0.0775	0.9911
HistGradientBoostingTree	0.0468	0.5267	0.9416	0.9931	0.5772	0.9997	0.5441	0.9814	0.0731	0.9909
GradientBoostingTree	0.0474	0.5298	0.9416	0.9956	0.5765	0.9998	0.5490	0.9816	0.0746	0.9898
RandomForest	0.0456	0.5004	0.9369	0.9934	0.5755	0.9998	0.5564	0.9807	0.0766	0.9881

Table 40: The performance of **medium-sized regression** task (*heterogeneous features*) (2).

	361102	**361294**	**361101**	**361103**	**361289**	**361287**	**361094**	**Ranking**
Default								
DOFEN (ours)	0.8838	0.9756	0.4178	0.6647	0.1835	0.0329	0.9996	5.47 ± 3.39
Trompt	0.8902	0.9782	0.0961	0.6494	0.1817	0.0445	0.9995	7.29 ± 3.71
GRANDE	0.7977	0.8822	0.2421	0.6217	0.0992	0.0000	0.9615	11.88 ± 2.95
FT-Transformer	0.8883	nan	0.2472	0.6710	0.0200	0.0000	0.9998	8.00 ± 3.17
ResNet	0.8736	0.9782	0.2434	0.6487	0.0305	0.0000	0.9958	11.00 ± 2.89
MLP	0.8751	0.9792	0.1580	0.6555	0.0321	0.0000	0.9999	8.06 ± 3.79
SAINT	0.8836	0.9777	0.4631	0.6602	0.1712	0.0401	0.9998	6.41 ± 2.62
NODE	0.0000	0.0000	0.3719	0.0000	0.0000	0.0370	0.0000	11.47 ± 4.58
CatBoost	0.8975	0.9776	0.5463	0.6916	0.1843	0.0313	0.9999	3.76 ± 3.45
LightGBM	0.8905	0.9779	0.5448	0.6874	0.1792	0.0199	0.9999	5.18 ± 2.89
XGBoost	0.8834	0.9773	0.5699	0.6619	0.1653	0.0000	1.0000	7.47 ± 3.94
HistGradientBoostingTree	0.8914	0.9785	0.5389	0.6904	0.1727	0.0302	0.9999	5.00 ± 3.11
GradientBoostingTree	0.8693	0.9794	0.4694	0.6717	0.1861	0.0305	0.9994	5.35 ± 4.27
RandomForest	0.8747	0.9767	0.5619	0.6551	0.1639	0.0233	1.0000	7.94 ± 3.49
Searched								
DOFEN (ours)	0.8908	0.9788	0.5102	0.6647	0.1829	0.0531	0.9996	6.65 ± 2.51
Trompt	0.8916	0.9787	0.4804	0.6690	0.1821	0.0503	0.9999	7.00 ± 3.09
GRANDE	0.7943	0.8837	0.2621	0.6290	0.0956	0.0000	0.9622	13.88 ± 3.05
FT-Transformer	0.8930	0.9796	0.5281	0.6731	0.1797	0.0496	0.9999	7.24 ± 2.85
ResNet	0.8846	0.9793	0.4483	0.6565	0.1798	0.0219	0.9977	10.12 ± 3.89
MLP	0.8849	0.9796	0.4737	0.6590	0.1716	0.0256	0.9999	8.94 ± 4.13
SAINT	0.8913	0.9796	0.5195	0.6706	0.1820	0.0525	0.9999	7.88 ± 2.91
NODE	0.8842	0.9782	0.4972	0.6477	0.0477	0.0423	0.9984	11.00 ± 3.40
CatBoost	0.8941	0.9787	0.5665	0.6930	0.1854	0.0574	1.0000	4.18 ± 4.04
LightGBM	0.8935	0.9781	0.5448	0.6874	0.1868	0.0546	1.0000	6.41 ± 3.97
XGBoost	0.8965	0.9788	0.5820	0.6909	0.1850	0.0644	1.0000	3.12 ± 3.10
HistGradientBoostingTree	0.8910	0.9791	0.5383	0.6904	0.1857	0.0623	1.0000	6.12 ± 2.95
GradientBoostingTree	0.8901	0.9794	0.5773	0.6841	0.1858	0.0305	1.0000	5.41 ± 3.00
RandomForest	0.8749	0.9787	0.5838	0.6744	0.1827	0.0701	1.0000	6.82 ± 3.77

Table 41: The performance of **large-sized classification** task (*numerical features only*).

	361069	**361068**	**361061**	**361274**	**Ranking**
Default					
DOFEN (ours)	<u>0.7306</u>	<u>0.9480</u>	0.8222	<u>0.8018</u>	3.25 ± 5.22
Trompt	0.7213	0.9468	<u>0.9004</u>	0.7954	2.88 ± 4.35
GRANDE	0.7248	0.9425	0.8315	0.7979	5.00 ± 3.05
FT-Transformer	0.6960	0.9403	0.8983	0.7586	8.25 ± 4.16
ResNet	0.6988	0.9409	0.8801	0.7358	8.50 ± 4.04
MLP	nan	nan	nan	nan	$nan \pm nan$
SAINT	0.7181	0.9436	0.8694	0.7860	6.00 ± 1.30
NODE	0.7247	0.9461	0.8886	0.7946	3.75 ± 2.92
CatBoost	0.7261	0.9432	0.8377	0.7954	4.38 ± 2.77
LightGBM	0.7212	0.9371	0.8071	0.7870	8.00 ± 2.30
XGBoost	0.7164	0.9367	0.8361	0.7828	8.50 ± 2.61
HistGradientBoostingTree	nan	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.7103	0.9225	0.7698	0.7718	11.00 ± 4.58
RandomForest	0.7158	0.9308	0.8767	0.7797	8.50 ± 3.29
Searched					
DOFEN (ours)	<u>0.7340</u>	<u>0.9479</u>	0.8888	<u>0.8033</u>	3.50 ± 5.76
Trompt	0.7286	0.9436	<u>0.9127</u>	0.7988	4.00 ± 4.09
GRANDE	0.7265	0.9454	0.8522	0.7980	6.00 ± 3.70
FT-Transformer	0.7299	0.9441	0.9062	0.7962	4.00 ± 3.35
ResNet	0.7228	0.9446	0.8935	0.7781	9.25 ± 4.49
MLP	nan	nan	nan	nan	$nan \pm nan$
SAINT	0.7255	0.9440	0.8956	0.7922	8.50 ± 2.11
NODE	0.7262	0.9471	0.8982	0.7946	5.25 ± 2.39
CatBoost	0.7299	0.9445	0.9015	0.7975	4.00 ± 2.77
LightGBM	0.7251	0.9433	0.8964	0.7938	8.50 ± 2.11
XGBoost	0.7279	0.9439	0.8956	0.7965	6.50 ± 1.52
HistGradientBoostingTree	nan	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.7247	0.9400	0.8978	0.7938	9.00 ± 3.27
RandomForest	0.7198	0.9353	0.9059	0.7885	9.50 ± 5.36

Table 42: The performance of **large-sized classification** task (*heterogeneous features*).

	361113	**361285**	**Ranking**
Default			
DOFEN (ours)	0.8691	<u>0.7870</u>	5.00 ± 5.11
Trompt	0.9276	0.7836	2.00 ± 5.77
GRANDE	0.8568	0.7785	8.00 ± 3.51
FT-Transformer	<u>0.9317</u>	0.7609	6.00 ± 5.03
ResNet	0.8945	0.7653	8.00 ± 3.51
MLP	nan	nan	$nan \pm nan$
SAINT	0.9123	0.7731	6.50 ± 2.57
NODE	0.9199	0.7774	5.00 ± 3.75
CatBoost	0.8827	0.7821	5.50 ± 2.29
LightGBM	0.8476	0.7797	8.00 ± 4.16
XGBoost	0.8781	0.7822	5.50 ± 3.04
HistGradientBoostingTree	nan	nan	$nan \pm nan$
GradientBoostingTree	0.7946	0.7519	12.00 ± 6.35
RandomForest	0.9066	0.7767	6.50 ± 1.61
Searched			
DOFEN (ours)	0.9116	0.7979	7.00 ± 4.16
Trompt	<u>0.9395</u>	0.7844	4.00 ± 4.80
GRANDE	0.8914	0.7771	12.00 ± 6.35
FT-Transformer	0.9348	0.7890	4.00 ± 3.88
ResNet	0.9226	0.7834	8.50 ± 2.65
MLP	nan	nan	$nan \pm nan$
SAINT	0.9252	0.7796	9.50 ± 4.04
NODE	0.9219	0.7800	10.00 ± 4.62
CatBoost	0.9368	<u>0.8012</u>	1.50 ± 6.08
LightGBM	0.9310	0.7977	4.50 ± 2.36
XGBoost	0.9294	0.7987	4.50 ± 3.40
HistGradientBoostingTree	nan	nan	$nan \pm nan$
GradientBoostingTree	0.9302	0.7862	6.00 ± 0.58
RandomForest	0.9327	0.7813	6.50 ± 2.52

Table 43: The performance of **large-sized regression** task (*numerical features only*).

	361080	**361083**	**361091**	**Ranking**
Default				
DOFEN (ours)	0.9469	0.5459	<u>0.3240</u>	3.00 ± 3.54
Trompt	0.9458	0.3379	0.2498	7.00 ± 1.29
GRANDE	0.8272	0.3243	0.2031	9.33 ± 3.40
FT-Transformer	0.9452	0.1198	0.1172	9.67 ± 4.27
ResNet	0.9410	0.2469	0.1188	9.67 ± 4.11
MLP	nan	nan	nan	$nan \pm nan$
SAINT	0.9445	0.5344	0.2887	5.67 ± 2.53
NODE	0.9453	0.0000	0.2763	7.67 ± 3.10
CatBoost	<u>0.9476</u>	0.5847	0.3020	1.67 ± 4.69
LightGBM	0.9475	0.5607	0.2810	3.00 ± 3.35
XGBoost	0.9474	<u>0.6087</u>	0.2512	3.67 ± 3.30
HistGradientBoostingTree	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.9459	0.4635	0.2574	5.67 ± 0.63
RandomForest	nan	nan	nan	$nan \pm nan$
Searched				
DOFEN (ours)	<u>0.9490</u>	0.5640	<u>0.3321</u>	2.00 ± 4.72
Trompt	0.9461	0.5242	0.2971	8.33 ± 2.59
GRANDE	0.8289	0.3707	0.2078	11.00 ± 5.00
FT-Transformer	0.9463	0.5382	0.3049	6.67 ± 1.50
ResNet	0.9465	0.5277	0.2770	8.33 ± 2.72
MLP	nan	nan	nan	$nan \pm nan$
SAINT	0.9465	0.5491	0.3053	5.33 ± 1.26
NODE	0.9460	0.3967	0.2892	9.33 ± 3.79
CatBoost	0.9480	0.5996	0.3130	2.33 ± 3.61
LightGBM	0.9475	0.5607	0.2810	6.00 ± 2.16
XGBoost	0.9480	<u>0.6249</u>	0.3070	2.33 ± 3.71
HistGradientBoostingTree	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.9471	0.6157	0.3046	4.33 ± 2.50
RandomForest	nan	nan	nan	$nan \pm nan$

Table 44: The performance of **large-sized regression** task (*heterogeneous features*).

	361104	**361095**	**361096**	**361101**	**361103**	**Ranking**
Default						
DOFEN (ours)	0.9998	0.6120	0.9923	0.5288	0.6823	4.00 ± 2.48
Trompt	0.9996	0.6097	0.9917	0.4035	0.7048	6.20 ± 1.86
GRANDE	0.9153	0.4275	0.9011	0.3544	0.6370	10.00 ± 3.99
FT-Transformer	0.9994	0.3514	0.9923	0.4061	0.6761	7.40 ± 2.79
ResNet	0.9895	0.3370	0.9816	0.3971	0.6660	9.80 ± 3.27
MLP	nan	nan	nan	nan	nan	$nan \pm nan$
SAINT	0.9997	0.3891	0.9918	0.5480	0.6874	5.80 ± 1.79
NODE	0.9997	0.4156	0.9875	0.0198	0.6626	8.80 ± 2.79
CatBoost	0.9998	0.6332	<u>0.9928</u>	0.6050	<u>0.7068</u>	2.00 ± 3.44
LightGBM	0.9998	0.6324	0.9916	0.5769	0.7037	4.00 ± 2.40
XGBoost	<u>0.9998</u>	<u>0.6345</u>	0.9922	<u>0.6244</u>	0.7060	1.80 ± 3.93
HistGradientBoostingTree	nan	nan	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.9998	0.6165	0.9857	0.4809	0.6773	6.20 ± 1.74
RandomForest	nan	nan	nan	nan	nan	$nan \pm nan$
Searched						
DOFEN (ours)	0.9998	0.6242	<u>0.9935</u>	0.5855	0.6923	4.40 ± 2.94
Trompt	0.9998	0.6286	0.9918	0.5479	0.7073	7.00 ± 2.34
GRANDE	0.9158	0.4350	0.9010	0.3899	0.6421	10.40 ± 4.02
FT-Transformer	0.9998	0.3891	0.9924	0.5708	0.7045	7.60 ± 2.83
ResNet	0.9998	0.3937	0.9923	0.5336	0.6864	7.80 ± 2.90
MLP	nan	nan	nan	nan	nan	$nan \pm nan$
SAINT	0.9998	0.3952	0.9926	0.5659	0.6974	6.40 ± 1.75
NODE	0.9998	0.5908	0.9918	0.4135	0.6668	8.20 ± 2.99
CatBoost	0.9998	0.6363	0.9932	0.6263	0.7116	3.20 ± 3.39
LightGBM	0.9998	0.6324	0.9924	0.5769	0.7096	5.20 ± 2.35
XGBoost	<u>0.9998</u>	<u>0.6383</u>	0.9932	<u>0.6479</u>	<u>0.7122</u>	1.40 ± 4.00
HistGradientBoostingTree	nan	nan	nan	nan	nan	$nan \pm nan$
GradientBoostingTree	0.9998	0.6301	0.9918	0.6361	0.7057	4.40 ± 3.07
RandomForest	nan	nan	nan	nan	nan	$nan \pm nan$

H More Experiment Settings

H.1 Hardware Used

The following hardware configuration was used for all of our experiments. The hardware selection was based on availability, with neural networks consistently run on GPUs and tree-based models executed on CPUs.

GPUs: NVIDIA GeForce RTX 2080 Ti, NVIDIA DGX1, NVIDIA A100

CPUs: Intel(R) Xeon(R) Silver 4210 CPU, Intel(R) Xeon(R) CPU E5-2698 v4, AMD EPYC605 7742 64-core Processor

H.2 Hyperparameter Search Space

This section details the hyperparameter search space adopted for each model, as referenced in various tables (Tables 45 to 56). We have employed search spaces consistent with those presented in the Tabular Benchmark [1] for models including XGBoost, GradientBoostingTree, RandomForest, FT-Transformer, SAINT, ResNet, and MLP.

Additionally, we have defined specific search spaces for newer baselines such as CatBoost, LightGBM, Trompt, NODE, and GRANDE. For CatBoost, our search space aligns with the parameters specified by the FT-Transformer study [11]. In the case of LightGBM, we have derived the search space based on recommendations from field practitioners, as cited in [34, 35]. For NODE, our approach follows the guidelines provided in TabZilla [15]. For GRANDE, we follow the settings provided in the notebook example from the official github of GRANDE.

Table 46: Hyperparameter search space of XGBoost.

Hyperparameter	Distribution
max_depth	uniform_int$[1, 11]$
num_estimators	1000
min_child_weight	log_uniform_int$[1, 1e2]$
subsample	unifrom$[0.5, 1]$
learning_rate	log_unifrom$[1e{-}5, 0.7]$
col_sample_by_level	uniform$[0.5, 1]$
col_sample_by_tree	uniform$[0.5, 1]$
gamma	log_uniform$[1e{-}8, 7]$
lambda	log_uniform$[1, 4]$
alpha	log_uniform$[1e{-}8, 1e2]$

Table 47: Hyperparameter search space of CatBoost.

Hyperparameter	Distribution
max_depth	$[3, 4, 5, 6, 7, 8, 9, 10]$
learning_rate	log_uniform$[1e{-}5, 1]$
iterations	quantile_uniform$[100, 6000]$
bagging_temperature	uniform$[0, 1]$
l2_leaf_reg	log_uniform$[1, 10]$
leaf_estimation_iteration	$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

In the context of our model, DOFEN, we have focused our search on the number of m and d, which relate to the varied number of N_{cond} and the conditions per rODT. Additionally, we have explored the drop_rate parameter to fine-tune the degree of regularization in our model. It is important to note that the overall search space for DOFEN is relatively compact when compared to the other baseline models while achieve competitive performance.

Table 45: Hyperparameter search space of DOFEN.

Hyperparameter	Distribution
d	$[3, 4, 6, 8]$
m	$[16, 32, 64]$
drop_rate	$[0.0, 0.1, 0.2]$

Table 48: Hyperparameter search space of LightGBM.

Hyperparameter	Distribution
learning_rate	uniform$[0.001, 1]$
max_depth	$[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$
bagging_fraction	uniform$[0.1, 1.0]$
bagging_frequency	$[1, 2, 3, 4, 5]$
num_leaves	quantile_uniform$[30, 150]$
feature_fraction	uniform$[0.1, 1.0]$
num_estimators	1000
boosting	[gbdt, rf, dart]

Table 49: Hyperparameter space of GradientBoostingTree.

Hyperparameter	Distribution
loss	[deviance, exponential]$(classification)$, [squared_error, absolute_error, huber]$(regression)$
learning_rate	log_normal[$\log(0.01), \log(10)$]
subsample	uniform[$0.5, 1$]
num_estimators	1000
criterion	[friedman_mse, squared_error]
max_depth	[none, $2, 3, 4, 5$]
min_samples_split	[2.3]
min_impurity_decrease	[$0.0, 0.01, 0.02, 0.05$]
max_leaf_nodes	[none, $5, 10, 15$]

Table 50: Hyperparameter search space of RandomForest.

Hyperparameter	Distribution
max_depth	[none, $2, 3, 4$]
num_estimators	250
criterion	[gini, entropy]
max_features	[sqrt, log2, none, $0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$]
min_samples_split	[$2, 3$]
min_samples_leaf	log_uniform_int[$1.5, 50.5$]
bootstrap	[true, false]
min_impurity_decrease	[$0.0, 0.01, 0.02, 0.05$]

Table 51: Hyperparameter search space of NODE.

Hyperparameter	Distribution
num_layers	[$2, 4, 8$]
total_tree_count	[$1024, 2048$]
tree_depth	[$6, 8$]
tree_output_dimension	[$2, 3$]$(regression)$, [num_classes]$(classification)$

Table 52: Hyperparameter search space of Trompt.

Hyperparameter	Distribution
hidden_dimension	[$18, 128$]
feature_importances_type	[concat, add]
feature_importances_dense	[true, false]
feature_importances_residual_connection	[true, false]
feature_importances_sharing_dense	[true, false]
feature_embeddings_residual_connection	[true, false]
minimal_batch_ratio	[$0.1, 0.01$]

Table 53: Hyperparameter search space of FT-Transformer.

Hyperparameter	Distribution
mum_layers	uniform_int[$1, 6$]
feature_embedding_size	uniform_int[$64, 512$]
residual_dropout	uniform[$0, 0.5$]
attention_dropout	uniform[$0, 0.5$]
FFN_dropout	uniform[$0, 0.5$]
FFN_factor	uniform[$2/3, 8/3$]
learning_rate	log_uniform[$1\mathrm{e}{-5}, 1\mathrm{e}{-3}$]
weight_decay	log_uniform[$1\mathrm{e}-6, 1\mathrm{e}-3$]
KV_compression	[true, false]
LKV_compression_sharing	[headwise, key_value]
learning_rate_scheduler	[true, false]
batch_size	[$256, 512, 1024$]

Table 54: Hyperparameter search space of SAINT.

Hyperparameter	Distribution
num_layers	uniform_int$[1, 2, 3, 6, 12]$
num_heads	$[2, 4, 8]$
layer_size	uniform_int$[32, 64, 128]$
dropout	$[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$
learning_rate	log_uniform$[1e{-}5, 1e{-}3]$
batch_size	$[128, 256]$

Table 55: Hyperparameter search space of ResNet.

Hyperparameter	Distribution
num_layers	uniform_int$[1, 16]$
layer_size	uniform_int$[64, 1024]$
hidden_factor	uniform$[1, 4]$
hidden_dropout	$[0, 0.5]$
residual_dropout	uniform$[0, 0.5]$
learning_rate	log_uniform$[1e{-}5, 1e{-}2]$
weight_decay	log_uniform$[1e{-}8, 1e{-}3]$
category_embedding_size	uniform_int$[64, 512]$
normalization	[batch_norm, layer_norm]
learning_rate_scheduler	[true, false]
batch_size	$[256, 512, 1024]$

Table 56: Hyperparameter search space of MLP.

Hyperparameter	Distribution
num_layers	uniform_int$[1, 8]$
layer_size	uniform_int$[16, 1024]$
dropout	$[0, 0.5]$
learning_rate	log_uniform$[1e{-}5, 1e{-}2]$
category_embedding_size	uniform_int$[64, 512]$
learning_rate_scheduler	[true, false]
batch_size	$[256, 512, 1024]$

Table 57: Hyperparameter search space of GRANDE.

Hyperparameter	Distribution
depth	$[4, 6]$
n_estimators	$[512, 1024, 2048]$
learning_rate_weights	log_uniform$[1e{-}4, 1e{-}1]$
learning_rate_index	log_uniform$[5e{-}3, 2e{-}1]$
learning_rate_values	log_uniform$[5e{-}3, 2e{-}1]$
learning_rate_leaf	log_uniform$[5e{-}3, 2e{-}1]$
cosine_decay_steps	$[0, 100, 1000]$
loss	[crossentropy, focal_crossentropy]$(classification)$, [mse]$(regression)$
dropout	$[0.0, 0.25, 0.5]$
selected_variables	uniform$[0.5, 1.0]$
data_subset_fraction	uniform$[0.8, 1.0]$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We have verified our idea using experiments as shown in Section 4. The results not only reveal the state-of-the-art performance of DOFEN among deep neural networks on tabular data but also demonstrate the nuance of its architecture design, which echos the contribution and scope we mentioned in the abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: See Section 5 for limitations.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This paper focuses on model architecture design and empirical evaluations based on hypotheses, without making theoretical assumptions or providing proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: See Section 3.2 and Appendix A.2 for model implementation details, Appendices A.1 and H.2 for hyperparameter settings, and Section 4.1 and Appendices B.1 to B.3 for dataset settings.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code will be published in the future; currently, we provide an anonymous version in the supplemental material. For datasets, we use an open-source benchmark and strictly follow its official implementation as described in Section 4.1.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: For dataset settings, see Section 4.1 and Appendices B.1 to B.3. For detailed hyperparameter settings, see Appendices A.1 and H.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For tables in Appendix G.2, we calculate the mean and standard deviation of ranks across datasets to provide the rank for each model with their confidence interval.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix H.1 for hardware settings and Appendix C.1 for the FLOPs and inference time of different models.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in our paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This paper proposes a novel DNN model, with experiments focusing on performance comparisons with other existing DNN and tree-based models, as well as exploring the mechanisms behind this novel method. We acknowledge that the introduction of new model structures can have both positive and negative societal impacts (e.g. fairness considerations); however, these aspects are not the primary scope or focus of this paper.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The public assets accessed in this paper include models and datasets. We have clearly cited each of them and strictly followed their terms of use. For all public models, please refer to their respective websites for the license information. Regarding the Tabular Benchmark, all datasets used are publicly available from OpenML [36] under the CC-BY 4.0 license. Please refer to Appendix B.3 for detailed information on the datasets used.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The source code for this paper will be published under the Apache License 2.0 in the future. During the reviewing process, a minimal workable example of DOFEN is provided in the supplementary material, along with a simple README explaining how to execute the code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.