Language Models Are Good Tabular Learners

Zhenhan Huang *huangz12@rpi.edu*
Department of Computer Science
Rensselaer Polytechnic Institute

Kavitha Srinivas *Kavitha.Srinivas@ibm.com*
IBM Research

Horst Samulowitz *samulowit@us.ibm.com*
IBM Research

Niharika S. D'Souza *Niharika.Dsouza@ibm.com*
IBM Research

Charu C. Aggarwal *charu@us.ibm.com*
IBM Research

Pin-Yu Chen *Pin-Yu.Chen@ibm.com*
IBM Research

Jianxi Gao *gaoj8@rpi.edu*
Department of Computer Science
Rensselaer Polytechnic Institute

Reviewed on OpenReview: *https://openreview.net/forum?id=6o3vVBWYis&referrer*

Abstract

Transformer-based language models have become the de facto standard in natural language processing. However, they underperform in the tabular data domain compared to traditional tree-based methods. We posit that current models fail to achieve the full potential of language models due to (i) heterogeneity of tabular data; and (ii) challenges faced by the model in interpreting numerical values. Based on this hypothesis, we propose the *Tabular Domain Transformer* (TDTransformer) framework. TDTransformer has distinct embedding processes for different types of columns. The alignment layers for different column-types transform these embeddings to a common space. Besides, TDTransformer adapts piece-wise linear encoding for numerical values for better performance. We test the proposed method on 76 real-world tabular classification datasets from the OpenML benchmark. Extensive experiments indicate that TDTransformer improves the state-of-the-art methods.

1 Introduction

Deep learning methods have achieved state-of-the-art (SOTA) performance in various areas including vision (Rombach et al., 2022; He et al., 2022; Zou et al., 2024; Jiang et al., 2024), language (Radford et al., 2019; Touvron et al., 2023), and multimodal processing (Radford et al., 2021; Liu et al., 2023). Even though deep learning methods have shown great potential in many domains, their performance for tabular data has so far been unimpressive. This has led to the question as to whether deep learning is a fundamentally superior approach for tabular data (Shwartz-Ziv & Armon, 2022; Grinsztajn et al., 2022; Borisov et al., 2022; McElfresh et al., 2024). Experimental benchmarks Grinsztajn et al. (2022); Borisov et al. (2022) have shown the broad superiority of tree-based methods over deep learning. Among deep learning methods, the generalization of transformer-based architectures (Vaswani et al., 2017) to tabular data has shown some promise — however, they continue to lag tree-based methods such as XGBoost (Chen & Guestrin, 2016).

The broad-based success of transformers in learning high-dimensional representations, especially in NLP, is evidence of their potential. A natural question arises as to *what makes transformer-based architectures underperform tree-based methods*. Based on prior studies, we posit that this phenomenon is a result of (i) difficulty in learning irregular patterns of the target function owing to data heterogeneity (Shwartz-Ziv & Armon, 2022; Mathov et al., 2022; Borisov et al., 2023; Yan et al., 2023; Chen et al., 2024a), and (ii) the challenges faced by the model in interpreting numerical features (Gorishniy et al., 2021; 2022).

On the one hand, spectral analysis of neural networks indicates that neural networks tend to learn the low-frequency components of a function in lieu of relatively high-frequency components (Rahaman et al., 2019; Xu et al., 2019; Beyazit et al., 2024). Owing to the different types of columns, the feature spaces in the tabular data domain are generally heterogeneous. On the other hand, numerical reasoning is known to be a formidable challenge for language models (Lu et al., 2022; Lee et al., 2023; Shen et al., 2023; Testolin, 2024; Ahn et al., 2024).

We propose a framework named **T**abular **D**omain **Transformer** (TDTransformer) that overcomes the aforementioned obstacles in the way of achieving the full potential of transformer-based architectures. TDTransformer embeds different types of table columns using different approaches to obtain the hidden representations. For each column type, we use an alignment layer to map the hidden representation to a common embedding space. Alignment layers for different column types are independent of one another. This design is inspired by multimodal models such as CLIP (Radford et al., 2021) where one alignment layer transforms the hidden dimension of the image branch d_i to a dimension d, $\psi_i : d_i \to d$. The other one transforms the hidden dimension of the text branch d_t to d, $\psi_t : d_t \to d$. To enhance the model's understanding of numerical values, we use a piecewise linear encoding (PLE) that directly maps scalars to high-dimensional embeddings. Compared to conventional tokenization and embedding, PLE introduces an inductive bias that is beneficial to the training process. Our use of the PLE is inspired by the pioneering work of Gorishniy et al. (2022). We adapt PLE such that the hidden representation is close to the conventional hidden representation of transformer-based architectures. We combine hidden representations as the input to the backbone model. The pipeline of the training process is the pre-training model followed by fine-tuning.

We examine the performance of TDTransformer on the standard tabular data benchmark OpenML [1]. Extensive experiments on more than 70 tabular data sets show the superiority of TDTransformer. In summary, the main contributions of this work are as follows:

- To avoid the performance degradation caused by the heterogeneous nature of tabular data, we design different embedding approaches to obtain the hidden representations of columns. Alignment layers are applied to hidden representations to ensure that embeddings for different types of columns are in the same embedding space.
- We adapt the piece-wise linear encoding to improve the representation of numerical values so that the model can interpret them well. These encoded representations are combined with those of categorical and binary columns and then input to the backbone model.
- We propose a column-type dependent corruption for pre-training. We also propose a column-type-aware positional encoding that further boosts the performance of TDTransformer.

2 Related Work

Tabular deep learning A key line of work in tabular deep learning focuses on the use of graph learning to enhance the understanding of relations among columns. An auxiliary knowledge graph is used to regularize a multilayer perceptron (Ruiz et al., 2024). Chen et al. (2024b) utilizes a hypergraph to capture tabular structures. With the development of large-scale foundational models, researches have emerged on adapting foundation models in the tabular data domain. Zhang et al. (2023) uses parameter-efficient fine-tuning to

[1] OpenML benchmark: `https://www.openml.org/`

adapt the pre-trained LLaMA 2 model to the tabular domain. Zhu et al. (2024) converts tables to formats that are consistent with the pre-training data (*e.g.* markdown format). The converted input data are directly fed to the pre-trained LLaMA 2 model (Touvron et al., 2023). Deng et al. (2024) treat tables as images and utilize the multimodal capability of GPT-4 (Achiam et al., 2023) and Gemini (Team et al., 2023). Hegselmann et al. (2023) serializes column names and values into a natural language string. Input strings are used for fine-tuning pre-trained large language models.

To tackle the feasibility of transformer architectures for processing heterogeneous tabular data, TABBIE (Iida et al., 2021) proposes the pre-training objective of detecting corrupted cells, and the architecture combining row and column transformers to enhance contextualization across tables. TAPAS (Herzig et al., 2020) combines BERT encoder (Devlin, 2018) with table-structure-related embeddings. TABERT (Yin et al., 2020) jointly learns natural language sentences and tabular data. Besides, both self-attention and vertical self-attention are applied to enhance the understanding of table structure. TUTA (Wang et al., 2021b) utilizes tree-based attention and position embedding to capture spatial and hierarchical information within tables. These works focus on help language models understand the structured tabular data. For different types of columns, an embedding process similar to the embedding of natural language word tokens is applied. Without using specialized embeddings for different types of columns or merely replying on the column type embedding makes it challenging to tackle the heterogeneity issue within tables.

Numerical reasoning Large language models mainly focus on NLP and code generation. However, their application to numerical reasoning has turned out to be less successful (Lewkowycz et al., 2022; Imani et al., 2023; Ahn et al., 2024; Romera-Paredes et al., 2024). This difficulty arises for multiple reasons: (i) numerical reasoning might require intricate intermediate steps internally. Language models map scalars to high-dimensional embeddings. The intermediate steps with high dimensional embeddings turn out to be intractable; (ii) there is no built-in mechanism within transformer-based architectures to perform mathematical operations; (iii) numerical values are continuous, whereas transformer-based architectures are inherently designed for (discrete) word toens; (iv) there are repeated patterns in tokenized numerical values, and each token holds equal significance (while omitting unimportant tokens).

Geva et al. (2020) enhances numerical reasoning by adding automatically generated synthetic numerical data to the pre-training process. Lee et al. (2023) incorprates ideas from chain-of-thought (Wei et al., 2022), with intermediate step results. Shen et al. (2023) focuses on data format modification to the boost model's understanding of numerical values. McLeish et al. (2024) helps models track the position of each digit by adding an embedding that encodes its relative position. These works focus on enhancing the understanding of numerical values from the language perspective. In the tabular data domain, however, the distribution of numerical values might be more important than the values themselves. Besides, due to the limitation of the sequence length in transformer-based architectures, treating numerical values as tokens can severely limit the context length of embeddings of other types of columns when numerical values contain a large number of digits.

3 TDTransformer: Tabular Data Transformer Framework

Task formulation Tabular data are denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, $x \in \mathbb{X}$, $y \in \mathbb{Y}$. The dataset is split into 3 disjoint subsets $D = D_{\text{train}} \cup D_{\text{val}} \cup D_{\text{test}}$. $\mathbb{Y} = \{0, 1\}$ for the binary classification task, $\mathbb{Y} = \{1, \ldots, C\}$ for the multiclass classification task. The supervised training process maximizes the likelihood of the correct label y:

$$\max_{\boldsymbol{\theta}} \mathbb{P}_{\boldsymbol{\theta}}(y|\mathbf{x}, \boldsymbol{\theta}) \ . \tag{1}$$

Figure 1 shows the proposed framework. Input data are relational tables that have a unique column given a column name. We use different embedding processes for different types of columns. Alignment layers are used to transform embeddings to the same embedding space. We combine embeddings as the input to the backbone model. The training pipeline consists of pre-training and fine-tuning steps.

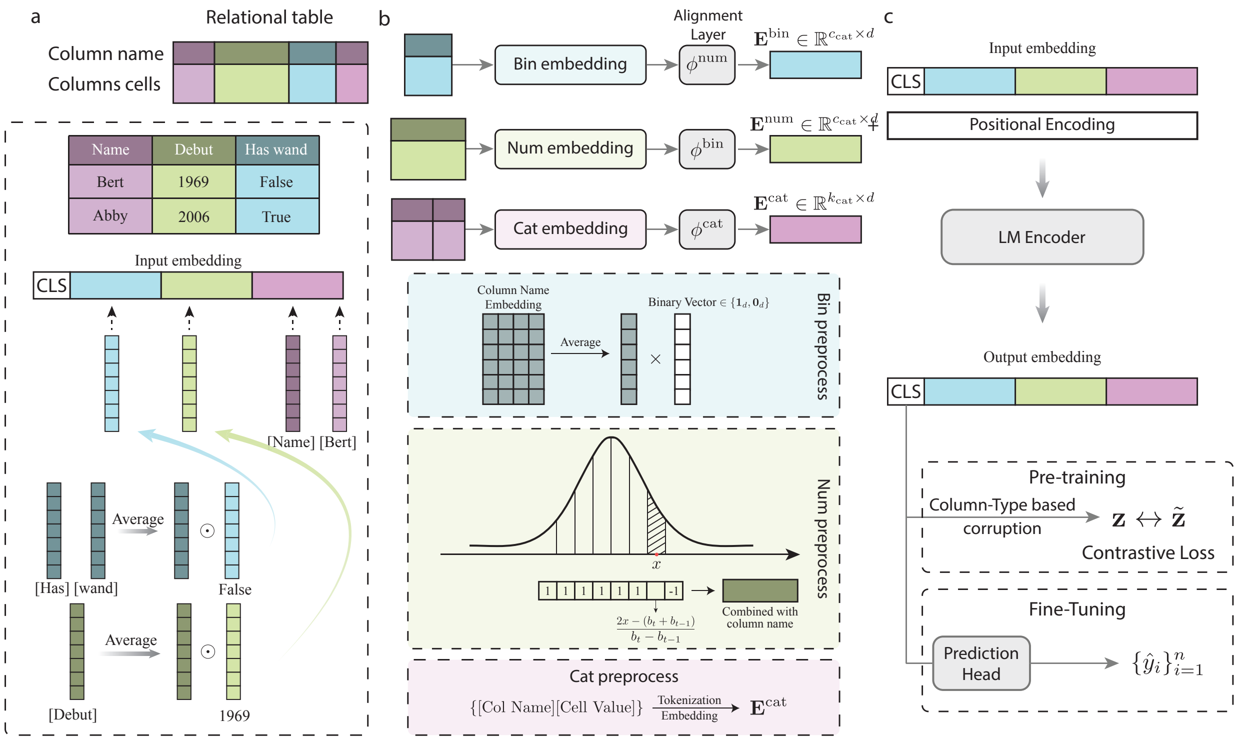

Figure 1: The TDTransformer framework: (a) Input data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. x consists of column name and column cells. The lower part shows a toy example of converting a relational table into the input embedding for LM encoder. (b) Embeddings of three types of columns (categorical, numerical, and binary). (c) Concatenation of the three types of embeddings, which is fed into the backbone model.

3.1 Column Embeddings

The TDTransformer framework uses distinct embedding processes for categorical, numerical, and binary columns. These processes are illustrated in Figure 1 (b) and discussed in detail below:

Embedding Categorical Columns For categorical columns, we concatenate column names and corresponding cell values to form natural language sentences. The concatenated sentence is tokenized and embedded to obtain hidden representations for the categorical columns denoted by $\mathbf{E}^{\text{cat}}$:

$$\mathbf{E}^{\text{cat}} = [\mathbf{E_1}, \ldots, \mathbf{E}_{c_{\text{cat}}}] \in \mathbb{R}^{k_{\text{cat}} \times d}, \quad \mathbf{E}_i = \left[e([T^{\text{col}}_{1:m_1(i)}]), e([T^{\text{cell}}_{1:m_2(i)}])\right], \tag{2}$$

where d is the dimension of hidden representations. $k_{\text{cat}} = \sum_{i=1}^{c_{\text{cat}}} \left(\left|[T^{\text{col}}_{1:m_1(i)}]\right| + \left|[T^{\text{cell}}_{1:m_2(i)}]\right| \right)$ is the total number of tokens. $m_1(i)$ is the number of tokens for i-th categorical column name. $m_2(i)$ is the number of tokens for cell values in i-th categorical column. After embedding, we use a linear transformation layer $\phi^{\text{cat}} : \mathbb{R}^d \to \mathbb{R}^d$.

Embedding Numerical Columns We adapt the PLE in Gorishniy et al. (2022) for transformer-based architectures. Specifically, we use the PLE function $f_{\text{ple}} : \mathbb{R} \to \mathbb{R}^{d_{\text{ple}}}$ to obtain the hidden representation for numerical columns:

$$f_{\text{ple}}(x) = [\xi_1, \ldots, \xi_n]^T \in \mathbb{R}^{d_{\text{ple}}}. \tag{3}$$

$$\xi_i = \begin{cases} -1, & x < b_{t-1} \\ 1, & x > b_t \\ \frac{2x-(b_t+b_{t-1})}{b_t-b_{t-1}}. & b_{t-1} \le x \le b_t \end{cases} \tag{4}$$

Here, the bins $\{b_t\}_{t=1}^{d_{\text{ple}}}$ are obtained based on the q-quantiles of the numerical value range while the original PLE work requires labels for fitting decision trees. d_{ple} is the number of quantiles. We summarize differences for the numerical value embedding in Table 1. The conventional tokenization and embedding in language models can map a scalar to a sequence of embeddings if there are multiple digits in the scalar. On the contrary, PLE always maps a scalar to one embedding. Non-Parametric fitting using PLE is capable of precisely determining the boundaries separating numerical values based on the dependence of numerical values on target labels (Mohanty & Fahnestock, 2021; Thielmann et al., 2025). Our method relies on the distribution of cell values and not conditions on labels. Besides, due to the layer normalization (LN) (Ba, 2016) within the embedding layer, co-domain of $[-1, 1]$ for our adapted PLE function is closer to the embedding than that of $[0, 1]$.

Table 1: Summary of the differences among methods obtaining embeddings of numerical values. Embedding of a numerical value is essentially a high-dimensional vector $\mathbf{v}$.

Method	v_i range	Not require labels	Fixed sequence length
Tokenization + Embedding	$(-\infty, \infty)$	✓	×
PLE (Gorishniy et al., 2022)	[0, 1]	×	✓
PLE (Ours)	[-1, 1]	✓	✓

We use a linear transformation layer $\phi^{\text{num}} : \mathbb{R}^{\text{ple}} \to \mathbb{R}^d$ to convert the high-dimensional representations $f_{\text{ple}}(x) \in \mathbb{R}^{\text{ple}}$ to the same embedding space as that of categorical column embeddings.

Our PLE function does not require training to convert numerical values to high dimensional vectors $\mathbb{R} \to \mathbb{R}^{d_{\text{ple}}}$. The hidden representation for numerical columns is obtained by the Hadamard product of the averaged column-name embedding and numerical-value embeddings:

$$\mathbf{E}^{\text{num}} = \mathbf{E}^{\text{num}}_{\text{col}} \odot \phi^{\text{num}}([f_{\text{ple}}(x_1), \dots, f_{\text{ple}}(x_{c_{\text{num}}})]) \in \mathbb{R}^{c_{\text{num}} \times d} \ . \tag{5}$$

$$\mathbf{E}^{\text{num}}_{\text{col}} = [\mathbf{E}_1, \dots, \mathbf{E}_{c_{\text{num}}}], \quad \mathbf{E}_i = \frac{1}{m_1(i)} \sum_{j=1}^{m_1(i)} e([T^{\text{col}}_{1:m_1(i)}]) \odot \mathcal{M} \ , \tag{6}$$

Here, $\mathcal{M}$ is the attention mask to exclude padding token embeddings, $\odot$ is the Hadamard product. For notational conciseness we ignore the notation of column types in the expressions of word tokens of column names and cell values. For example, we use the same notation $[T^{\text{col}}_{1:m_1(i)}]$ in Equations 2 and 6. The notation $[T^{\text{col}}_{1:m_1(i)}]$ denotes the word tokens for numerical column names in the former, whereas it denotes the word tokens for categorical column names in the latter.

Embedding Binary columns We convert cell values (*e.g.* True vs False and 0 vs 1) in binary columns to binary values $x_i \in \{0, 1\}$. Similar to numerical columns, the column-name embedding for each binary column is averaged. The hidden representation for binary columns is obtained by the Hadamard product of the averaged column name embedding and binary values as follows:

$$\mathbf{E}^{\text{bin}} = \mathbf{E}^{\text{bin}}_{\text{col}} \odot (\mathbf{x}(\mathbf{1}_d)^T) \in \mathbb{R}^{c_{\text{bin}} \times d}, \text{ where } \mathbf{x} = [x_1, \dots, x_{c_{\text{bin}}}]^T \ . \tag{7}$$

Similar to the embedding of column names for numerical columns $\mathbf{E}^{\text{num}}_{\text{col}}$, $\mathbf{E}^{\text{bin}}_{\text{col}}$ is averaged to ensure a constant sequence length c_{bin}. We use a linear transformation layer $\phi^{\text{bin}} : d \to d$ to ensure the embeddings of binary columns are the same as those of categorical columns and numerical columns.

3.2 Feature Combination

Figure 1 (c) shows the combination of features for three types of columns. We prepend `[CLS]` embedding to the concatenated hidden representations. As in the classic transformer model (Vaswani et al., 2017), we add the sinusoidal positional encoding $\mathbf{P}$ to the concatenated hidden representation:

$$\mathbf{E} = [e([\text{CLS}]), \mathbf{E}^{\text{bin}}, \mathbf{E}^{\text{num}}, \mathbf{E}^{\text{cat}}] + \mathbf{P} \ , \tag{8}$$

where

$$\mathbf{P}_{(j,2i)} = \sin(j/10000^{2i/d}) \ , \tag{9a}$$

$$\mathbf{P}_{(j,2i+1)} = \cos(j/10000^{2i/d}) \ . \tag{9b}$$

Here, j is the position index and i is the hidden dimension index. Only $\mathbf{E}^{\text{cat}}$ has a flexible sequence length. $\mathbf{E}^{\text{bin}}$ and $\mathbf{E}^{\text{num}}$ have a fixed sequence length. The sequence length for $\mathbf{E}^{\text{bin}}$ or $\mathbf{E}^{\text{num}}$ is equal to the number of binary columns or numerical columns. Given a fixed context length limit, TDTransformer can process larger tables (without truncation) as compared to language models that do tokenization and embedding for all types of columns.

In language models, positional encoding or positional embedding are added to the embedding in element-wise fashion. In tabular data domains, however, table columns have the permutation invariance property that prevents positinal encodings from improving performance (Huang et al., 2020). In TDTransformer, although the hidden representations for the binary columns ($\mathbf{E}_i^{\text{bin}} \in \mathbb{R}^d$) and that for the numerical columns ($\mathbf{E}_i^{\text{num}} \in \mathbb{R}^d$), there is indeed an ordering in $\mathbf{E}^{\text{cat}}$, because it is essentially the embedding of a natural language sentence. Therefore, we propose a column-type aware (CTA) position encoding for TDTransformer. CTA only adds positional encoding to $\mathbf{E}^{\text{cat}}$. The overall embedding $\mathbf{E}$ is computed as follows:

$$\mathbf{E} = [e([\text{CLS}]), \mathbf{E}^{\text{bin}}, \mathbf{E}^{\text{num}}, \mathbf{E}^{\text{cat}}] + [\mathbf{0}_{(c_{\text{bin}}+c_{\text{num}})\times d}, \mathbf{P}] \ . \tag{10}$$

3.3 Training Pipeline

After combining column embeddings, $\mathbf{E}$ is fed to the backbone model, which is constructed using the gated transformer proposed in Wang & Sun (2022). We also test the performance using RoBERTa (Liu, 2019) as the backbone model (see Appendix). `[CLS]` embedding is used for the prediction. The training pipeline, similar to the classic pre-training fine-tuning paradigm, consists of two steps: the first step is to pre-train the model. The second step is to fine-tune the model that is initialized with pre-trained weights. Pre-training is widely used in tabular deep learning to boost model performance (Yin et al., 2020; Iida et al., 2021; Somepalli et al., 2021; Rubachev et al., 2022; Wang & Sun, 2022; Müller et al., 2023; Zhu et al., 2023). Corruption is used to generate negative samples. The corruption method conditions on column types, because random permutation only occurs within the same type of column. We do not apply permutations for binary columns.

After the contextualization in LM encoder $\mathcal{F}(\cdot)$, we obtain the resulting embedding $\mathbf{E}^{'} = \mathcal{F}(\mathbf{E})$. We use the `[CLS]` embedding in $\mathbf{E}^{'}$ as shown in Figure 1 (c). The `[CLS]` embedding used for the prediction is denoted as $\mathbf{z}$. Given a table row $\mathbf{z}_i$, there is a hidden representation $\mathbf{z}_i$.

Before the pre-training process, the weights of the backbone model are randomly initialized. The pre-training process uses contrastive loss. Specifically, we examine two types of pre-training losses: self-supervised contrastive loss (e.g., Chen et al. (2020); Tian et al. (2020); Wang et al. (2021a)) and supervised contrastive loss (e.g., Khosla et al. (2020); Jaiswal et al. (2020); Le-Khac et al. (2020)). The contrastive loss function encourages the model to generate close embeddings for positive pairs. The self-supervised pre-training focuses on the category-level discrimination while self-supervised pre-training pays attention to the instance-level discrimination.

The self-supervised contrastive loss (SSCL) is computed as follows:

$$\mathcal{L}^{\text{SSCL}} = -\sum_{i\in\mathcal{I}} \log \frac{\exp(\mathbf{z}_i^T \tilde{\mathbf{z}}_i/\tau)}{\sum_{j\in\mathcal{I}} \exp(\mathbf{z}_i^T \tilde{\mathbf{z}}_j/\tau)} \ , \tag{11}$$

where τ is the temperature, $\mathcal{I} = \{i\}_{i=1}^n$, $\mathbf{z}_i$ is the hidden representation for i-th table row, and $\tilde{\mathbf{z}}_i$ is the hidden representation of the corrupted i-th table row.

The supervised contrastive loss (SCL) utilizes labels in the pre-training dataset and is computed as follows:

$$\mathcal{L}^{\text{SCL}} = \sum_{i\in\mathcal{I}} \frac{-1}{P(i)} \log \sum_{k\in P(i)} \frac{\exp(\mathbf{z}_i^T \tilde{\mathbf{z}}_k/\tau)}{\sum_{j\in\mathcal{I}} \exp(\mathbf{z}_i^T \tilde{\mathbf{z}}_j/\tau)} \ , \tag{12}$$

where $P(i) \coloneqq \{p|y_p = y_i\}$. SCL is found to be a powerful pre-training tool. For example, it can achieve in-context learning in decision-making problems (Lee et al., 2024) and learn data with long-tailed distributions (Li et al., 2022).

Table 2: Performance comparison for the binary classification task. In addition to the averaged performance, we select a subset of 76 tables for detailed comparison. $\mathcal{S} \cup \mathcal{S}_{\text{num}}$ contains tables including numerical columns. γ is the positive ratio.

Method	$\mathcal{S} \cup \mathcal{S}_{\text{num}}$		$\gamma \leq 0.2$		$0.2 < \gamma < 0.8$		$\gamma \geq 0.8$		Avg	
	Acc	Auc	Acc	Auc	Acc	Auc	Acc	Auc	Acc	Auc
XGBoost	85.06	0.83	91.88	**0.87**	78.44	0.82	95.10	0.73	84.97	0.83
CatBoost	86.27	0.86	**91.90**	**0.87**	80.66	0.87	94.51	0.87	86.12	0.87
SCARF	77.27	0.73	83.84	0.72	73.64	0.78	72.10	0.55	77.81	0.74
SwitchTab	74.32	0.78	79.67	0.80	69.89	0.78	89.05	0.74	75.03	0.78
SubTab	71.94	0.74	75.44	0.74	69.79	0.75	72.59	0.68	72.30	0.75
TransTab	84.83	0.81	91.20	0.83	79.74	0.82	95.45	0.83	85.39	0.82
Vanilla MLP	79.50	0.61	84.46	0.68	74.03	0.56	86.84	0.45	79.23	0.60
SAINT	86.16	0.85	84.24	0.85	**86.92**	0.86	86.25	0.79	85.77	0.85
FT-Transformer	85.63	0.84	90.50	**0.87**	80.96	0.83	95.49	0.80	85.74	0.85
TabM	86.71	0.81	86.67	0.78	86.23	0.83	91.03	0.90	86.69	0.81
Mambular	87.42	**0.88**	87.56	0.86	86.25	**0.88**	**97.81**	**1.00**	87.51	**0.88**
TDTransformer	**87.56**	0.87	91.67	**0.87**	83.94	**0.88**	95.40	0.96	**87.79**	**0.88**
TDTransformer (CTA Pos)	87.19	0.87	91.70	**0.87**	83.30	0.87	95.59	0.94	87.48	0.87

Table 3: Performance comparison for the multiclass classification task. In addition to the averaged performance, we select a subset of 76 tables for detailed comparison. $\mathcal{S} \cup \mathcal{S}_{\text{num}}$ contains tables including numerical columns. $|\mathcal{D}|$ is the dataset size, $\mathfrak{C}$ is the number of classes.

Method	$\mathcal{S} \cup \mathcal{S}_{\text{num}}$		$\|\mathcal{D}\| < 2000$		$\|\mathcal{D}\| \geq 2000$		$\mathfrak{C} < 10$		$\mathfrak{C} \geq 10$		Avg	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
XGBoost	72.56	0.60	65.77	0.56	**82.20**	**0.71**	79.32	0.64	71.12	0.69	76.45	0.66
CatBoost	73.03	0.59	66.68	0.56	81.97	0.70	79.32	0.63	71.59	0.69	76.61	0.65
SCARF	62.39	0.52	57.58	0.44	69.51	0.61	67.75	0.53	60.82	0.59	65.32	0.55
SwitchTab	56.92	0.45	57.56	0.45	62.29	0.52	64.93	0.50	52.65	0.49	60.63	0.50
SubTab	55.22	0.45	55.98	0.44	60.77	0.52	60.99	0.48	55.57	0.53	59.09	0.50
TransTab	70.22	0.53	70.38	0.53	69.96	0.52	71.79	0.49	66.98	0.63	70.11	0.54
Vanilla MLP	56.23	0.35	43.43	0.20	65.85	0.46	66.89	0.38	41.48	0.35	58.00	0.37
SAINT	63.63	0.61	64.95	0.60	70.97	**0.71**	65.55	0.64	75.03	0.74	68.87	0.67
FT-Transformer	73.20	0.61	73.71	0.65	78.91	0.68	81.62	0.67	68.67	0.67	77.09	0.67
TabM	76.53	0.62	77.34	0.63	80.53	0.69	80.84	0.63	76.77	0.75	79.41	0.67
Mambular	75.92	**0.63**	76.81	0.68	81.41	0.70	**81.73**	**0.67**	76.22	0.74	79.80	0.69
TDTransformer	76.30	**0.63**	78.68	**0.69**	81.06	0.70	80.89	0.65	79.00	**0.77**	80.23	**0.70**
TDTransformer (CTA Pos)	**76.70**	**0.63**	**78.94**	**0.69**	81.36	0.70	81.07	0.65	**79.47**	**0.77**	**80.51**	**0.70**

The model weight after the pre-training process is used as the initialized weight for the fine-tuning process. A prediction head is added to predict the probability as shown in Figure 1. The fine-tuning process is in a supervised fashion. For the binary classification task, we use the binary cross entropy loss. The multiclass classification task employs the cross entropy loss.

4 Experiments and Results

4.1 Experiments

Baseline methods XGboost (Chen & Guestrin, 2016) is an end-to-end tree boosting system. It uses a sparsity-aware algorithm and weighted quantile sketch. Compared to XGBoost, CatBoost (Prokhorenkova et al., 2018; Dorogush et al., 2018) has the inherent capability to process categorical features without relying on one-hot encoding. Besides, it introduces ordered boosting to avoid target leakage. SubTab (Ucar et al., 2021) divides input features into multiple subsets to perform multiview representation learning. Scarf

(Bahri et al., 2022) uses vanilla self-supervised contrastive learning to improve classification accuracy in the fully-supervised learning setting. SwitchTab (Wu et al., 2024) uses an asymmetric encoder-decoder framework to decouple mutual and salient features, which can address the issue of lacking dependencies between samples. FT-Transformer (Gorishniy et al., 2021) adapts the transformer architecture for the tabular domain. The embeddings of numerical columns are obtained by the linear transformation whereas categorical columns utilize the lookup table for embeddings. SAINT (Somepalli et al., 2021) uses row-wise attention and column-wise attention to enhance the embedding process. TabM (Gorishniy et al., 2024) improves tabular multilayer perceptron (MLP) by parameter-efficient ensembling. Mambular (Thielmann et al., 2024) adapts the Mamba architecture (Gu & Dao, 2023) for the tabular domain.

Datasets We use 76 real-world tabular classification datasets in the standard OpenML benchmark (which are manually curated for effective benchmarking). The train/validation/test splits is 72%/8%/20% for each OpenML dataset. We use accuracy as the metric to measure the performance for all classification data sets. Additionally, we use the area under the curve (AUC) to evaluate binary classification and the F1 score t evaluate multiclass classification. The details of the tables are given in Appendix Section A.4.

Experimental details TDTransformer uses pre-trained BERT tokenizer (Devlin, 2018) and Adam optimizer (Kingma, 2014) without weight decay. The hidden dimension is 512 and model depth is 12. The number of quantiles for PLE is 64. In both the pre-training and fine-tuning process, we use an early stopping strategy (Yao et al., 2007) with a patience of 10. The maximum number of training epochs is 200 with batch size of 128. The corruption parameter of pre-training process is set to 0.5. When there are empty cells in a column, we replace empty cells with the most common values in that column. We conducted all epxeriments using a single A40 Tensor Core GPU and EPYC 7232P CPU.

4.2 Results

Table 2 summarizes the performance comparison for the binary classification task. SSCL is used in the pre-training process. We denote categorical columns as $\mathcal{S}_{\text{cat}}$, binary columns as $\mathcal{S}_{\text{bin}}$, and numerical columns as $\mathcal{S}_{\text{num}}$. We use the notation $\mathcal{S}$ for generic table columns, $\mathcal{S} \subseteq (\mathcal{S}_{\text{num}} \cup \mathcal{S}_{\text{cat}} \cup \mathcal{S}_{\text{bin}})$. Note that $\mathcal{S}$ can be $\emptyset$. In addition to select subsets of tabular data based on column types, we use the positive ratio to make a selection. The positive ratio γ is the ratio of positive samples to the entire number of samples. The comparison of computational costs is reported in Appendix Section A.6. Overall, both TDTransformers exhibit better performance (with or without CTA positional encoding).

The performance comparison for the multiclass classification task is shown in Table 3. We use the dataset size $|\mathcal{D}|$ and the number of classes $\mathfrak{C}$ to select subsets of tabular data. For nearly all selected subsets, TDTransformer (with or without CTA positional encoding) shows a pronounced performance gain compared to baseline methods. For the subset of $|\mathcal{D}| \geq 2000$, XGBoost has the best performance. We examine datasets where our proposed framework has a relatively large performance gap compared to XGBoost. We find a remarkable gap appearing in the table Au4-2500 (Details regarding all tables are listed in Appendix). In this table, both column names and categorical columns lack semantics. Column names are `V1`, …, `V100`. Categorical columns contain cell values of `v1`, `v2`, … `Vk`, $k \in \mathbb{N}^{+}$. Lacking semantics is detrimental to the performance of language models. Hence, XGBoost outperforms TDTransformer by a relatively large margin.

Figure 2 shows the comparison between TDTransformer and baseline methods. Scatter points are the performance on individual dataset. Transformer-based baselines fall short remarkably compared to tree-based methods. Even though the TDTransformer model has a transformer-based architecture, it achieves better performance than all baselines.

4.3 Ablation Study

Pre-training We compare the performance of pre-training using SSCL and SCL. Both pre-training processes use the classic positional encoding as shown in Equation 8. The performance comparison is shown in Figure 3. Using SCL as shown in Equation 12, there is a small accuracy decrease in the binary classification task. The performance has a larger drop in the multiclass classification task. Overall, TDTransformer has better

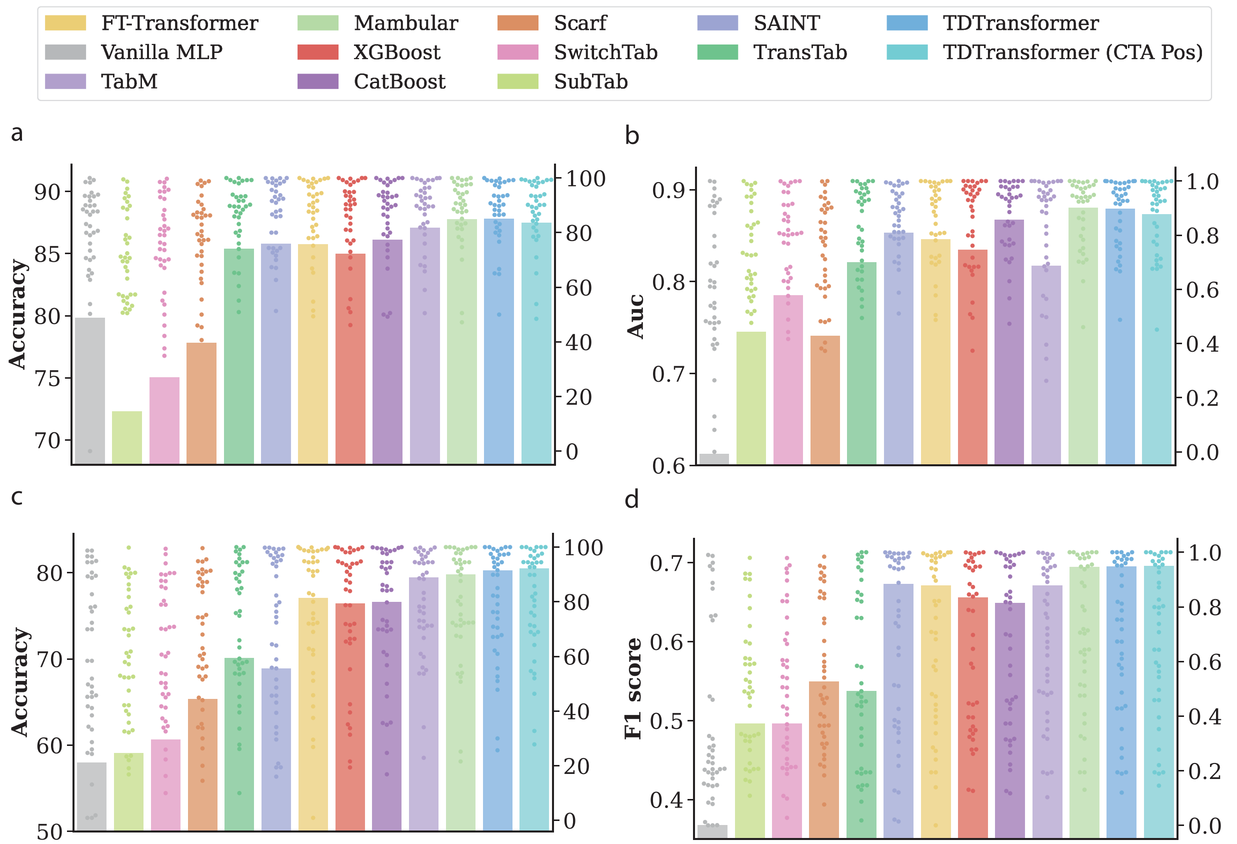

Figure 2: Performance comparison: The left axis shows the scale for (average) performance. The right axis shows the scale for the performance on individual datasets. (a) Test accuracy for the binary classification task. (b) AUC score for the binary classification task. (c) Test accuracy for the multiclass classification task. (d) F1 score for the multiclass classification task. TDTransformer outperforms baselines with greater improvements achieved for multiclass classification.

performance using SSCL compared to SCL. Out of the tabular data domain, a similar observation is reported that self-supervised pre-training without label information learns more effective representation than supervised pre-training when transferring to downstream tasks (Chen et al., 2020; He et al., 2020; Chen & He, 2021).

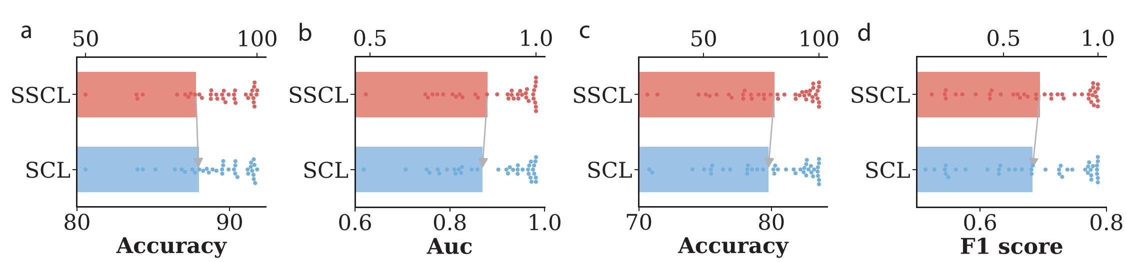

Figure 3: The performance comparison between SSCL and SCL pre-training. The upper axis shows the scale for the performance on individual datasets while the lower axis shows the scale for the averaged performance. (a) Test accuracy for the binary classification task. (b) Auc score for the binary classification task. (c) Test accuracy for the multiclass classification task. (d) F1 score for the multiclass classification task.

Positional encoding Attention mechanism (Vaswani et al., 2017) computes the pair-wise relation between the query and key. There is no inherent order of the sequence. Positional encoding or learnable positional embedding are added to help model track the order. However, tables have the inherent property of permutation invariance, which is contradictory to the order of the word token sequence. Huang et al. (2020) compares the transformer with positional encoding and without positional encoding. In their framework, no positional encoding leads to better performance. We compare the performance without positional encoding, with positional encoding and with CTA positional encoding.

Table 4: Performance comparison between different positional encoding methods. Positional encoding and CTA positional encoding have similar performance while no positional encoding can lead to a pronounced performance drop.

Task	Metric	w/o positional encoding	w/ positional encoding	w/ CTA positional encoding
Binary	Accuracy	88.07	87.79	87.48
	Auc	0.87	0.88	0.87
Multiclass	Accuracy	74.78	80.23	80.51
	F1	0.63	0.70	0.70

Batch size In SSCL, the number of negative pairs is related to the batch size. In SCL, batch size determines the number of negative and positive pairs. We use the same batch size in the pre-training and fine-tuning processes. Different batch sizes $\{128, 64, 32\}$ are examined to analyze the effect of batch size.

Table 5 shows the effect of batch size in the binary classification task. Overall, the effect of batch size is small. The average accuracy variation is within 0.2%. Table 6 exhibits the effect of batch size in the multiclass classification task. When decreasing the batch size, both accuracy and F1 score decrease.

Table 5: The effect of batch size N_{bs} on the performance of TDTransformer in the binary classification task. SSCL is used in the pre-training process. The fine-tuning process is in a supervised fashion.

Method	$\mathcal{S} \cup \mathcal{S}_{\text{num}}$		$\gamma \leq 0.2$		$0.2 < \gamma < 0.8$		$\gamma \geq 0.8$		Avg	
	Acc	Auc	Acc	Auc	Acc	Auc	Acc	Auc	Acc	Auc
TDTransformer ($N_{bs} = 128$)	87.56	0.87	91.67	0.87	83.94	0.88	95.40	0.96	87.79	0.88
TDTransformer ($N_{bs} = 64$)	87.61	0.82	91.22	0.79	84.44	0.85	95.54	0.94	87.88	0.83
TDTransformer ($N_{bs} = 32$)	87.70	0.86	91.56	0.86	84.37	0.87	95.54	0.94	87.99	0.88

Table 6: The effect of batch size N_{bs} on the performance of TDTransformer in the multiclass classification task. SSCL is used in the pre-training process. The fine-tuning process is in a supervised fashion.

Method	$\mathcal{S} \cup \mathcal{S}_{\text{num}}$		$\lvert\mathcal{D}\rvert < 2000$		$\lvert\mathcal{D}\rvert \geq 2000$		$\mathfrak{C} < 10$		$\mathfrak{C} \geq 10$		Avg	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1
TDTransformer ($N_{bs} = 128$)	76.30	0.63	78.68	0.69	81.06	0.70	80.89	0.65	79.00	0.77	80.23	0.70
TDTransformer ($N_{bs} = 64$)	75.78	0.62	78.54	0.68	80.58	0.69	80.33	0.64	79.00	0.77	79.86	0.69
TDTransformer ($N_{bs} = 32$)	76.16	0.62	78.97	0.68	79.40	0.66	79.43	0.61	78.90	0.77	79.24	0.67

Table 4 summarized the averaged performance for the binary and multiclass classification tasks. For the binary classification task, the performance difference among different encoding methods is small. There is a remarkable performance difference (5.45% drop in accuracy) for the multiclass classification task. Using no positional encoding pronouncedly degrades the performance. For tables that do not have numerical columns or binary columns, CTA positional encoding is the same as the traditional positional encoding. In the more challenging multiclass classification task, we observe the performance gain when using CTA positional encoding.

We examine the distribution of `[CLS]` embeddings by using t-SNE (Van der Maaten & Hinton, 2008) to compute the first two main components. Figure 4 shows the distribution of embeddings of table rows. Different

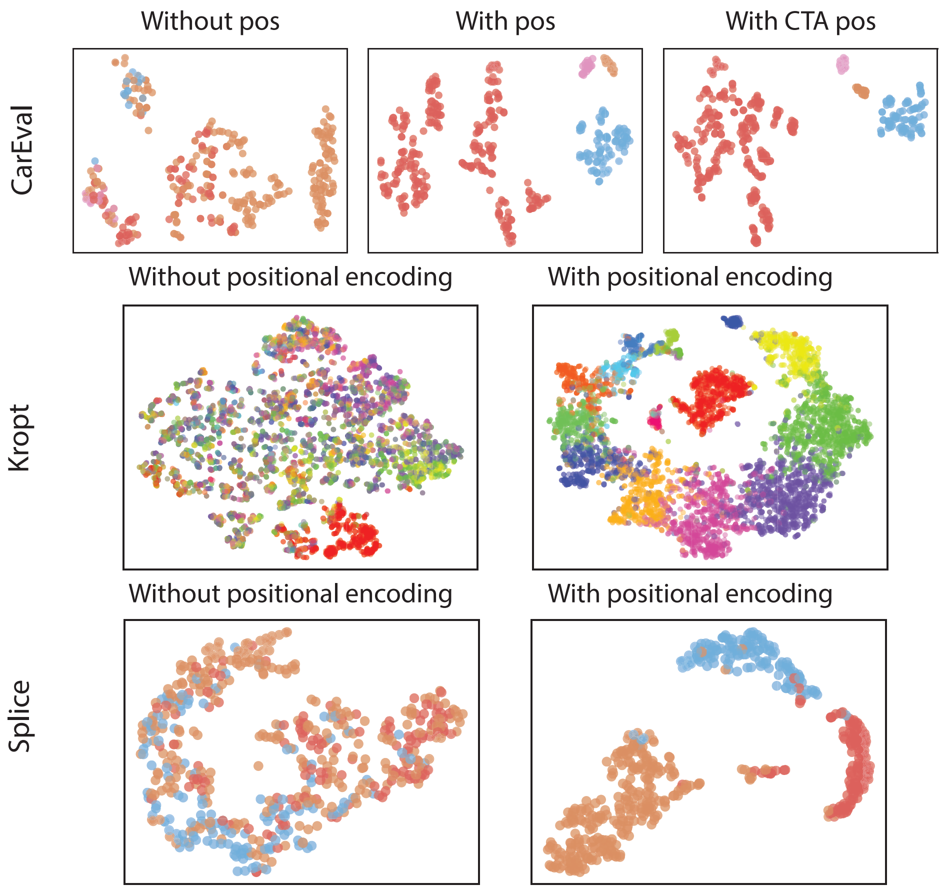

Figure 4: Effect of positional encoding on tabular representation learning. We assign the same color for instances with the same label. There are only categorical columns in Kropt and Splice tables, positional encoding and CTA positional encoding yield the same result. Using positional encoding greatly enhances representation learning.

classes are marked in distinct colors. Using positional encoding or CTA positional encoding pronouncedly improves the separation of different classes.

5 Discussion and Conclusion

Our results advocate a rethink of the power of language models in the tabular data domain. A direct way of applying language models to the tabular data domain is to represent tables using sequences of word tokens. However, the heterogeneity property of tables hinders models from learning effective representations (Shwartz-Ziv & Armon, 2022; Mathov et al., 2022; Borisov et al., 2023; Yan et al., 2023; Chen et al., 2024a). TDTransformer explicitly uses distinct embedding processes for different types of columns. Owing to the difference in embedding processes, the embedding spaces of different types of columns are different. Specifically, TDTransformer uses PLE to encode the statistical information of numerical columns in high-dimensional vectors while maintaining the continuity of numerical values in the co-domain of PLE function. Alignment layers are used to convert embeddings of different types of columns to a common embedding space. TDTransformer utilizes the good semantic understanding of language models. Some baseline methods with transformer-based architectures use one-hot encoded representation for categorical columns, which inherently loses semantic information. Those baselines lag behind tree-based methods. We find that language models might have unfavorable performance when a table has categorical columns that lack semantics. In addition, we find that positional encoding is important for the TDTransformer framework. The embeddings of numerical and binary columns are essentially column-wise, while those of categorical columns are token-wise. Based on this observation, we propose CTA positional encoding, which can boost the performance of TDTransformer.

The goal of the TDTransformer method is to convert tabular data to the embedding space of the traditional language models designed for NLP. Hence, this method can be readily generalized to various language encoders by replacing the language encoder in the TDTrasnformer framework with distinct encoders.

Overall, TDTransformer is able to to overcome the incapability of classical transformer-based architectures in interpreting heterogeneous data and to enhance the ability of the model to interpret numerical values.

We release our code in `https://github.com/Zhenhan-Huang/TDTransformer`.

6 Limitation

We use constant hyperparameters across 76 real-world tabular classification datasets. When obtaining the performance of baseline methods, we also use constant hyperparameters. Implementation details can be found in Appendix Section A.5. While our method demonstrates strong performance, it does not leverage the advantages of automated tuning techniques. As a result, the reported performance might not reflect the full potential of the proposed method. Existing works have shown that hyperparameter optimization (Kadra et al., 2021a;b) can greatly boost the performance of deep learning architecture on the tabular data domain. We leave the study of the effect of dataset-dependent hyperparameters on the performance of the TDTransformer method in our future work.

7 Broader Impact

In this work, we propose a way to help language models understand tabular data through the proposed embedding process, which paves the road for tabular operations through artificial intelligence automation. Different from tabular deep learning works using ordinal encoding or one-hot encoding, the TDTransformer framework utilizes word token embedding of both column names and cell contents to enhance the understanding of semantics within tables, which helps language models to understand tables based on the semantic level.

8 Acknowledgement

This work was supported by the USA National Science Foundation under grant #2047488, and IBM-RPI Future of Computing Research Collaboration. We acknowledge the computational resources provided by Artificial Intelligence Multiprocessing Optimized System in Rensselaer Polytechnic Institute.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*, 2024.

Jimmy Lei Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dara Bahri, Heinrich Jiang, Yi Tay, and Donald Metzler. Scarf: Self-supervised contrastive learning using random feature corruption. In *International Conference on Machine Learning*, pp. 1–24. PMLR, 2022.

Ege Beyazit, Jonathan Kozaczuk, Bo Li, Vanessa Wallace, and Bilal Fadlallah. An inductive bias for tabular deep learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 2022.

Vadim Borisov, Klaus Broelemann, Enkelejda Kasneci, and Gjergji Kasneci. Deeptlf: robust deep neural networks for heterogeneous tabular data. *International Journal of Data Science and Analytics*, 16(1): 85–100, 2023.

Jintai Chen, Jiahuan Yan, Qiyuan Chen, Danny Z Chen, Jian Wu, and Jimeng Sun. Can a deep learning model be a sure bet for tabular prediction? In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 288–296, 2024a.

Pei Chen, Soumajyoti Sarkar, Leonard Lausen, Balasubramaniam Srinivasan, Sheng Zha, Ruihong Huang, and George Karypis. Hytrel: Hypergraph-enhanced tabular data representation learning. *Advances in Neural Information Processing Systems*, 36, 2024b.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

Naihao Deng, Zhenjie Sun, Ruiqi He, Aman Sikka, Yulong Chen, Lin Ma, Yue Zhang, and Rada Mihalcea. Tables as texts or images: Evaluating the table reasoning ability of llms and mllms. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 407–426, 2024.

Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models. *arXiv preprint arXiv:2004.04487*, 2020.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35:24991–25004, 2022.

Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling. *arXiv preprint arXiv:2410.24210*, 2024.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9729–9738, 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16000–16009, 2022.

Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models. In *International Conference on Artificial Intelligence and Statistics*, pp. 5549–5581. PMLR, 2023.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. Tabbie: Pretrained representations of tabular data. *arXiv preprint arXiv:2105.02584*, 2021.

Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.

Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.

Chunheng Jiang, Zhenhan Huang, Tejaswini Pedapati, Pin-Yu Chen, Yizhou Sun, and Jianxi Gao. Network properties determine neural network performance. *Nature Communications*, 15(1):5718, 2024.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Regularization is all you need: Simple neural nets can excel on tabular data. *arXiv preprint arXiv:2106.11189*, 536, 2021a.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021b.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in neural information processing systems*, 33:18661–18673, 2020.

Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *Ieee Access*, 8:193907–193934, 2020.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*, 2023.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Tianhong Li, Peng Cao, Yuan Yuan, Lijie Fan, Yuzhe Yang, Rogerio S Feris, Piotr Indyk, and Dina Katabi. Targeted supervised contrastive learning for long-tailed recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6918–6928, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023.

Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*, 2022.

Yael Mathov, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous tabular data and adversarial examples. *Knowledge-Based Systems*, 242:108377, 2022.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When do neural nets outperform boosted trees on tabular data? *Advances in Neural Information Processing Systems*, 36, 2024.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, et al. Transformers can do arithmetic with the right embeddings. *arXiv preprint arXiv:2405.17399*, 2024.

Soumya D Mohanty and Ethan Fahnestock. Adaptive spline fitting with particle swarm optimization. *Computational Statistics*, 36(1):155–191, 2021.

Andreas Müller, Carlo Curino, and Raghu Ramakrishnan. Mothernet: A foundational hypernetwork for tabular classification. *arXiv preprint arXiv:2312.08598*, 2023.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International conference on machine learning*, pp. 5301–5310. PMLR, 2019.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

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

Camilo Ruiz, Hongyu Ren, Kexin Huang, and Jure Leskovec. High dimensional, tabular deep learning with an auxiliary knowledge graph. *Advances in Neural Information Processing Systems*, 36, 2024.

Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Alberto Testolin. Can neural networks do arithmetic? a survey on the elementary numerical skills of state-of-the-art deep learning models. *Applied Sciences*, 14(2):744, 2024.

Anton Thielmann, Thomas Kneib, and Benjamin Säfken. Enhancing adaptive spline regression: An evolutionary approach to optimal knot placement and smoothing parameter selection. *Journal of Computational and Graphical Statistics*, pp. 1–16, 2025.

Anton Frederik Thielmann, Manish Kumar, Christoph Weisser, Arik Reuter, Benjamin Säfken, and Soheila Samiee. Mambular: A sequential model for tabular deep learning. *arXiv preprint arXiv:2408.06291*, 2024.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pp. 776–794. Springer, 2020.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Talip Ucar, Ehsan Hajiramezanali, and Lindsay Edwards. Subtab: Subsetting features of tabular data for self-supervised representation learning. *Advances in Neural Information Processing Systems*, 34:18853–18865, 2021.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Geomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Xinlong Wang, Rufeng Zhang, Chunhua Shen, Tao Kong, and Lei Li. Dense contrastive learning for self-supervised visual pre-training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3024–3033, 2021a.

Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. Tuta: Tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1780–1790, 2021b.

Zifeng Wang and Jimeng Sun. Transtab: Learning transferable tabular transformers across tables. *Advances in Neural Information Processing Systems*, 35:2902–2915, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jing Wu, Suiyao Chen, Qi Zhao, Renat Sergazinov, Chen Li, Shengjie Liu, Chongchao Zhao, Tianpei Xie, Hanqing Guo, Cheng Ji, et al. Switchtab: Switched autoencoders are effective tabular learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 15924–15933, 2024.

Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.

Jiahuan Yan, Jintai Chen, Yixuan Wu, Danny Z Chen, and Jian Wu. T2g-former: organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 10720–10728, 2023.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, 2020.

Guri Zabërgja, Arlind Kadra, and Josif Grabocka. Tabular data: Is attention all you need? *arXiv preprint arXiv:2402.03970*, 2024.

Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206*, 2023.

Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. Xtab: Cross-table pretraining for tabular transformers. *arXiv preprint arXiv:2305.06090*, 2023.

Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat-Seng Chua. Tat-llm: A specialized language model for discrete reasoning over tabular and textual data. *arXiv preprint arXiv:2401.13223*, 2024.

Xueyan Zou, Jianwei Yang, Hao Zhang, Feng Li, Linjie Li, Jianfeng Wang, Lijuan Wang, Jianfeng Gao, and Yong Jae Lee. Segment everything everywhere all at once. *Advances in Neural Information Processing Systems*, 36, 2024.