UniTabE: Pretraining a Unified Tabular Encoder for Heterogeneous Tabular Data

Yazheng Yang†, YuQi Wang†, Guang Liu‡, Ledell Wu‡, Qi Liu†
The University of Hong Kong†, Beijing Academy of Artificial Intelligence‡
Pokfulam Road, Hong Kong, China†, Zhongguancun East Road, Beijing, China‡
{yangyazh,wangyuqi}@connect.hku.hk, {liuguang,wuyu}@baai.ac.cn, liuqi@cs.hku.hk

Abstract

Recent advancements in Natural Language Processing (NLP) have witnessed the groundbreaking impact of pretrained models, yielding impressive outcomes across various tasks. This study seeks to extend the power of pretraining methodologies to tabular data, a domain traditionally overlooked, yet inherently challenging due to the plethora of table schemas intrinsic to different tasks. The primary research questions underpinning this work revolve around the adaptation to heterogeneous table structures, the establishment of a universal pretraining protocol for tabular data, the generalizability and transferability of learned knowledge across tasks, the adaptation to diverse downstream applications, and the incorporation of incremental columns over time. In response to these challenges, we introduce UniTabE, a pioneering method designed to process tables in a uniform manner, devoid of constraints imposed by specific table structures. UniTabE's core concept relies on representing each basic table element with a module, termed TabUnit. This is subsequently followed by a Transformer encoder to refine the representation. Moreover, our model is designed to facilitate pretraining and finetuning through the utilization of free-form prompts. In order to implement the pretraining phase, we curated an expansive tabular dataset comprising approximately 13 billion samples, meticulously gathered from the Kaggle platform. Rigorous experimental testing and analyses were performed under a myriad of scenarios to validate the effectiveness of our methodology. The experimental results demonstrate UniTabE's superior performance against several baseline models across a multitude of benchmark datasets. This, therefore, underscores UniTabE's potential to significantly enhance the semantic representation of tabular data, thereby marking a significant stride in the field of tabular data analysis.[1]

1 Introduction

Tabular data is extensively utilized to present and organize information in diverse contexts, including webpages, spreadsheets, and database systems, etc. Consequently, it has garnered significant attention from the research community due to its numerous practical applications, such as table lookup [Wang et al., 2021a; Ye et al., 2022], table question answering (Table QA) [Herzig et al., 2020; Yin et al., 2020; Katsis et al., 2022; Cheng et al., 2022], formulas prediction [Cheng et al., 2021], and so on. In this work, we dedicate to pretrain a tabular encoder serving various downstream tasks.

Pretraining over voluminous tables necessitates a model that is adaptable to diverse tabular structures. Consequently, we need to reconsider what constitutes the basic element of a table. In practical applications, a row or record in a table typically represents an instance's information, with each

[1]We will release our pretrained models.

Preprint. Under review.

column in a row seen as a feature or attribute of the instance. A cell, being the intersection of a row and a column in a table, is the most granular unit. Generally, columns are independent of each other in terms of their order. This allows each instance in the table to be converted into an unordered sequence of cells, contributing to the model's flexibility in accepting any fixed-structure table as input and facilitating the addition of extra columns. Moreover, each cell in the table provides the finest level of granularity we can manipulate for different data types separately. Thus, in the context of this work, each cell is treated as the basic element of the table. In alignment with previous studies [Yin et al., 2020; Liu et al., 2022; Wang and Sun, 2022], we concentrate on the most common data types: numerical, textual, and categorical values. Specifically, we consider each digit of a numerical value as a token. For categorical values, we map the index to its meaning. For instance, we convert "1" to "True" and "0" to "False" for a column like "is_foreign_worker". Tokens of each cell are processed by the embedding layer containing the positional embedding to preserve the data structure, such as the order of tokens in text and the digital structure of numerical values.

Motivated by the recent advancements in pretraining techniques for natural language processing, our objective is to develop a unified tabular encoder that can serve as a versatile module for obtaining semantic representations in various tabular tasks. However, the transferability of knowledge across tables remains uncertain. Existing approaches typically rely on consistent table structures in both training and testing data [Huang et al., 2020; Wang and Sun, 2022; Jiang et al., 2022]. However, table structures often differ across tasks, rendering direct learning across tables impractical. One naive approach to address this issue involves merging distinct tables into a larger one, where the columns represent the union set of the original columns. Nevertheless, this approach introduces data sparsity and poses challenges when additional columns need to be appended, which is a common occurrence in practical scenarios. For example, in clinical trials, incremental columns are collected across different phases. In addition to adapting to different structures, the conversion of tabular data into the model input is crucial for neural network performance. Existing methods often propose strategies to transform a table into text format and subsequently apply pretraining similar to Masked Language Model (MLM) [Devlin et al., 2018; Salazar et al., 2019; Lewis et al., 2020] techniques in natural language processing. In this self-supervised training, the model is trained to predict the masked span within the text. However, tables contain various data formats, including natural language text, numerical values, categorical data, etc. Since numerical values make up a significant portion of tabular data, treating them as text without distinguishing their nature undermines the inherent structure and numerical meaning [Eisenschlos et al., 2021; Herzig et al., 2020]. Furthermore, in practice, it is more common for entire cell values to be missing rather than just a portion of the value. However, the dynamic masking employed in MLM struggles to ensure complete cell coverage. Traditionally, previous approaches failed to inform the model about the correspondence between column names and their values during the transformation of tabular data into sequences of tokens [Herzig et al., 2020; Arik and Pfister, 2021; Liu et al., 2022]. The literature mostly emphasizes the enhancement of powerful backbones, often at the expense of relatively simple and less sophisticated design of embedding modules tailored for different data types. This skewed focus often poses challenges to the model's capacity to effectively learn from such data, thereby underscoring a potential lacuna in the current methodological approach.

To address these identified challenges, we introduce *UniTabE*, an efficient and straightforward framework designed to process tables uniformly, eliminating the need for fixed table structures. Within this module, we adopt the data type embedding to aid the model in dealing with different data types, taking into account the divergent meanings associated with different data formats while forming the sequence of value tokens. For example, values derived from natural texts reflect grammatical and syntactic structures, while numerical values denote a distinct recording structure. The data type embedding thus conveys the differences between various data types. In order to constrain the relationship between the column name and its corresponding value, we design a *linking layer* that integrates a portion of the column name information into its value. Furthermore, we present a versatile model architecture that is tailored to accommodate a wide range of tabular tasks, facilitated by the use of prompts. Within this framework, we employ a shallow decoder to ensure the majority of the learned knowledge is stored into the semantic representation generated by the encoder. This vanilla decoder performs reasoning based on the high-level semantic representation provided by the encoder, serving as an adaptation module that can adjust to different tasks by simply employing task-specific prompts. For the training of our UniTabE, we have constructed an expansive tabular dataset amassed

from Kaggle.[2] This pretraining dataset comprises a considerable 13 billion examples sourced from approximately 300 distinct domains, with an average of 36.7 columns per table. To comprehensively evaluate the effectiveness of our method, we conduct extensive experiments including its application in predominant downstream tasks, tasks involving filling in missing values, zero-shot prediction, adaptation to incremental tables, and integration of the neural semantic representation with eXtreme Gradient Boosting (XGBoost), etc. This work makes three significant contributions:

- We delve into the realm of large-scale pretraining over tables and propose a novel framework, UniTabE, to address the inherent challenges. Our model is capable of accepting tables with heterogeneous structures as input and learning the semantic representation. Moreover, by integrating prompts within our framework, we enhance its scalability, enabling it to accommodate an extensive array of pretraining and finetuning tasks for downstream applications.
- We have crawled 7TB tables from Kaggle to construct a large-scale pretraining dataset that spans numerous domains. With this dataset, it substantiates the feasibility of conducting large-scale pretraining on tables.
- We have carried out comprehensive experiments to evaluate the performance of our pretrained UniTabE on a variety of benchmark datasets. We have also employed UniTabE in several Kaggle tasks, applied our model into the scenario of incremental columns, and explored equipping the learned semantic representation with XGBoost. The comparative results highlight the superiority of our approach.

2 Related Work

2.1 Table Data Featurization

Tables encompass various data types, and to process this tabular data with neural models, conventional methods typically convert each data format into a continuous space using individual strategies. For instance, texts are processed with tokenization and word embedding, while images are handled using image patch embedding. However, prior research tends to simplify this process by treating numerical values as text and then applying the same embedding strategy, which invariably disrupts the original recording structure and numerical meanings [Eisenschlos et al., 2021; Herzig et al., 2020]. Given the prevalence of numerical values in tables, this area has garnered increasing attention [Gorishniy et al., 2022]. To enhance the representation of numerical values, MATE [Eisenschlos et al., 2021] and TaPas [Herzig et al., 2020] introduced a ranking embedding based on numeric rank, which relies on comparisons. Further, TUTA [Wang et al., 2021b] applied additional numerical features, such as magnitude, precision, the first digit, and the last digit, to distinguish numbers from text. Gorishniy et al. [2022] attempted to train the embedding vector for numbers. Wang and Sun [2022] suggested categorizing tabular data into three distinct types: textual, categorical, and numerical. Their model, TransTab, concatenates columns of the same type into a text sequence, with column names, column values, and different columns separated by a space. After the concatenation, these three text sequences are fed into the embedding layer individually.

2.2 Pretraining Table Models

In recent years, the pretraining of language models (LMs) over vast text corpora has led to noteworthy enhancements in performance for a variety of downstream tasks. This success has stimulated a growing body of work that focuses on pretraining and adapting LMs specifically for tabular data. The prevailing method employed by these studies is to fine-tune LMs that have been pretrained on NLP datasets, such as BERT [Devlin et al., 2018], BART [Lewis et al., 2019], etc. Typically, this training utilizes the Masked Language Model (MLM) objective, as evidenced by models like Tabtransformer [Huang et al., 2020], TABERT [Yin et al., 2020], Tabnet [Arik and Pfister, 2021], Saint [Somepalli et al., 2021], and so on. Liu et al. [2022] proposed a modality transformation that converts tabular data into textual data using a basic lexicon and ordered syntax before feeding it into the pretrained LMs. They then finetuned their model using the same MLM objective. However, LMs pretrained on natural texts do not perform optimally as the textualized tabular data differs

[2]https://www.kaggle.com

fundamentally from natural language texts. The efficacy of finetuned LMs without architectural modifications has largely been confined to text data so far. But making modifications might lead to undesired outcomes like catastrophic forgetting of knowledge learned from natural language corpora [Chen et al., 2020; Bavarian et al., 2022].

In this work, our focus lies on pretraining a large model from scratch on tabular data. In line with previous work, we construct our model upon the renowned Transformer encoder, utilizing a self-supervised objective of MLM, which masks parts of the model input and then predicts the masked content. Contrary to previous work, we abstain from textualizing the tabular data using a simplistic strategy. Instead, we introduce a TabUnit module designed to process the basic element of a table independently, leading to an improved modeling of tabular data.

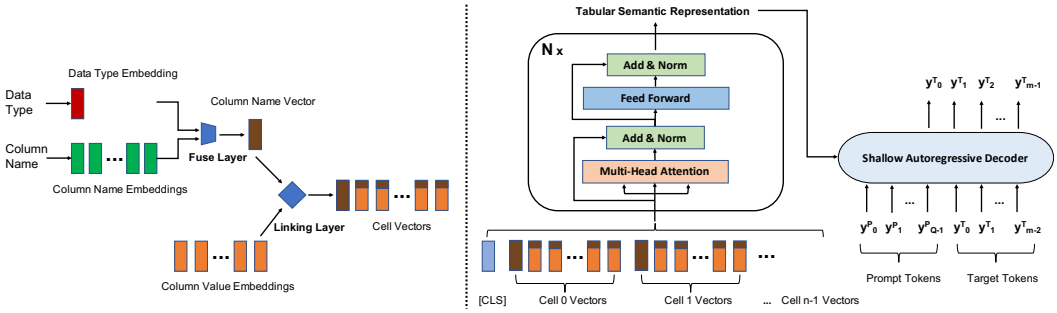


Figure 1: The left part shows the procedure of processing each cell, the basic element of tabular data in this work. The right part illustrates the overall architecture of our UniTabE. n denotes the number of cells in each example, Q denotes the length of prompt, while T here represents the length of target. The UniTabE takes the concatenation of [CLS] embedding and embeddings of cells as input. A shallow decoder is applied to guarantee the capability that stores most of the learned knowledge as well as the scalability that adapts to different downstream tasks.

3 UniTabE Architecture

In this section, we present the architecture of our model, which is composed of three primary components: the *TabUnit*, the *Encoding Layer*, and a *Shallow Decoder*. Our model employs the foundational TabUnit module to handle data of varying types and then utilizes the Transformer's encoder for further encoding. By leveraging the setting of prompts and integrating a decoder, our model becomes adaptable to a wide range of tasks.

TabUnit Module As depicted on the left side of Figure 1, we propose the use of an unified module, named TabUnit, for modeling the basic element of tabular data. To mitigate the influence of table structure, we treat each cell in a table as a key-value pair representing the column name and the column value respectively. The tokens of the column name are passed into the embedding module:

$$\mathbf{v}_{cn} = f_{fl}(Emb_{DT}(t_d), Avg(Emb(X_{cn}))) \tag{1}$$

where Emb represents the embeddings consisting of word embedding and positional embedding. $\mathbf{x}_{cn} = Avg(Emb(X_{cn}))$ is the vector after mean pooling across the dimension of token sequence. Here, we adopt the data type embedding Emb_{DT}, specifically designed to help the model adeptly handle diverse data formats, particularly in instances where columns share a name but contain values in different formats. For example, the values in the "salary" column of a table in a downstream task might be numerical, while those in the corresponding column of another table could be textual (e.g., high income, medium income, and low income, etc.). Our model integrates this information into column name embeddings to get column name vector via a fuse layer:

$$\begin{aligned} g_{dt} &= \text{Sigmoid}(\mathbf{v}_{fl}\text{ReLU}(\mathbf{w}_{fl}^{\top}\mathbf{x}_{dt} + \mathbf{b}_{fl})) \\ \mathbf{v}_{cn} &= (1 - g_{dt})\mathbf{x}_{cn} + g_{dt} * \mathbf{x}_{dt} \end{aligned} \tag{2}$$

where $\mathbf{w}_{fl}$, $\mathbf{b}_{fl}$ and $\mathbf{v}_{fl}$ are trainable parameters. $\mathbf{x}_{dt}$ is the data type embedding. Theoretically, integrating an equal amount of data type information into the column name representation across columns of the same data format is reasonable, as opposed to computing the fusing ratio g_{dt} based on both $\mathbf{x}_{cn}$ and $\mathbf{x}_{dt}$. Consequently, we compute g_{dt} solely based on the type embedding.

Tokens in each cell are also passed into the embedding module:

$$\{\mathbf{x}_{cv}^0, \mathbf{x}_{cv}^1, \mathbf{x}_{cv}^2, ..., \mathbf{x}_{cv}^{q-1}\} = Emb(\{x_{cv}^0, x_{cv}^1, x_{cv}^2, ..., x_{cv}^{q-1}\}) \tag{3}$$

Where the embedding module $Emb(.)$ is shared to carry out embedding for column names and column values, q here denotes the length of the cell value.

Given the orderless nature of self-attention within the Transformer encoder, it becomes challenging for the model to learn the connection between the column name and its value when all cells are concatenated into a sequence. As a result, we introduce a *Linking Layer* to establish the relationship within each name-value pair. We employ a gated function to weave the information from the column name into its corresponding value:

$$\begin{aligned} \alpha &= \text{Sigmoid}(\mathbf{v}_{lk}\text{ReLU}(\mathbf{w}_{lk}^\top \mathbf{v}_{cn} + \mathbf{b}_{lk})) \\ \mathbf{v}_{cv}^i &= \mathbf{x}_{cv}^i + \alpha * \mathbf{v}_{cn} \end{aligned} \tag{4}$$

where $\mathbf{w}_{lk}$, $\mathbf{b}_{lk}$ and $\mathbf{v}_{lk}$ are learnable parameters. We employ α to ensure that an equal amount of column name information is integrated into the value vectors. This allows the model to recognize which parts of vectors are values corresponding to specific column. To avoid having the value information overshadowed by the column name information, we only apply the multiplication operation to α and $\mathbf{v}_{cn}$. Overall, the TabUnit can be briefly formulated as:

$$\mathbf{X}_{TU} = \{\mathbf{v}_{cn}, \mathbf{v}_{cv}^0, \mathbf{v}_{cv}^1, ..., \mathbf{v}_{cv}^{q-1}\} = f_{TabUnit}(t_d, X_{cn}, X_{cv}) \tag{5}$$

where t_d, X_{cn} and X_{cv} denote the data type indicator, tokens of column name and tokens of column value, respectively. The concatenation of column name vector and value vectors is treated as the inner representation of a tabular cell. In our implementation, all cells are parallel processed.

Encoding Layer We concatenate the representations of all cells, and attach a trainable [CLS] vector to the head of such sequence. We leverage the Transformer encoder as the encoding layer:

$$\{\mathbf{h}_{cls}, \mathbf{h}^0, \mathbf{h}^1, ..., \mathbf{h}^{N-1}\} = f_{Enc}(\mathbf{v}_{cls}, \mathbf{X}_{TU}^0, \mathbf{X}_{TU}^1, ..., \mathbf{X}_{TU}^{n-1}) \tag{6}$$

where n is the number of columns, and N is the length after concatenating all cells' representations.

Shallow Decoder During pretraining, we want to encourage the encoder to store most of the learned knowledge. Hence, we adopt a Long Short-Term Memory network (LSTM) [Hochreiter and Schmidhuber, 1997] as the weak decoder. Specifically, the hidden state of [CLS] token and the prompt are passed to the decoder to compute the initial state of our decoder:

$$\{\mathbf{y}_0^P, \mathbf{y}_1^P, ..., \mathbf{y}_{Q-1}^P\} = f_{attn}(\mathbf{W}_1^\top Emb(\{y_0^P, y_1^P, ..., y_{Q-1}^P\}), \mathbf{W}_2^\top \{\mathbf{X}_{TU}^0, \mathbf{X}_{TU}^1, ..., \mathbf{X}_{TU}^{n-1}\}) \tag{7}$$

$$\mathbf{v}_p = \sum_i \frac{\exp(\mathbf{v}_1^\top \mathbf{y}_i^P)}{\sum_j \exp(\mathbf{v}_1^\top \mathbf{y}_j^P)} \mathbf{y}_i^P \tag{8}$$

$$\mathbf{v}_{state} = \mathbf{W}_s^\top \{\mathbf{v}_p, \mathbf{h}_{cls}\} + \mathbf{b}_s \tag{9}$$

where the f_{attn} denotes the dot-attention. $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{v}_1$, $\mathbf{W}_s$ and $\mathbf{b}_s$ are trainable parameters. Here $\mathbf{v}_p$ represents the weighted average of attention states of the prompt. The embedding layer of decoder also share the same parameter as that one on TabUnit. The target sequence of tokens are generated by the decoder step by step conditioned on the initial state and previously produced token.

4 Pretraining & Finetuning

4.1 Pretraining Objective

Previous research in NLP pretraining utilized self-supervised tasks within datasets to provide supervised training signals, such as predicting next token, generating masked spans of texts, and determining the subsequent sentence, etc. These studies have shown that unlabeled data can aid in the learning of semantic representation. In this work, we also adopt the mask-then-predict approach to facilitate self-supervised training. We treat each cell as the basic masked unit, as opposed to the token level in NLP pretraining. In practical applications, filling in the entire content of a cell is more useful than merely filling in part of the cell content. As such, we randomly replace the content of the

Table 1: Overall statistics of our pretraining dataset. "**Avg# NC**", "**Avg# CC**" and "**Avg# TC**" means the average number of numerical columns, categorical column and textual column in each table respectively. The bottom part demonstrates the Top-5 domains in our dataset.

Domains	**# Domains**	**# Tables**	**# Examples**	**Avg# NC**	**Avg# CC**	**Avg# TC**
ALL	303	283K	13B	28.7	0.4	7.7
Investing	1	71K	1B	29.33	0.02	1.58
Time Series	1	65K	1B	6.47	0.02	2.27
Finance	1	52K	773M	37.57	0.04	1.46
Economics	1	47K	488M	40.34	0.01	1.27
Games	1	32K	430M	23.37	0.66	3.66

masked cells with a special token, [MASK]. We also use [MASK] as the default content for cells whose values are missing.

For each example, we arbitrarily mask columns and train the model to predict the masked content. Often, there may be missing values in downstream applications. Therefore, we also train the model under conditions where several values are missing, to familiarize the model with such situations. The number of masked cells varies and follows a standard normal distribution, with single-cell masking being the most likely outcome. The prompt template for pretraining is set to "fill in missing value, <column name> :", which specifies the precise masked column to predict. The details regarding the number of randomly masked cells and their corresponding probabilities will be elaborated in § 6.1. Our model is trained with the optimization objective of the maximum log likelihood estimation.

4.2 Finetuning Formulation

Filling in Missing Value as Prediction. When finetuning our model for downstream tasks, we can consider the target as an additional column of the table. The model is then tasked with predicting the masked values of this target column using the same prompt as during pretraining. Thanks to the decoder, UniTabE is capable of generating textual and numerical targets. For example, the trained model is suited for classification and regression tasks, as well as predicting missing values in tables. In our implementation, we also support constrained generation. This feature is particularly beneficial for classification tasks, as the model only needs to predict from a small subset of the vocabulary.

Finetuning with Task-specific Prompt. Apart from those tasks where we treat the target as the masked column of the table, there are tasks that require the model to perform reasoning over the table and other inputs. For instance, in table question answering (TableQA) tasks, the model needs to produce an answer conditioned on the provided table and question. The prompt used in these tasks may include the task description and the question, appropriately formatted.

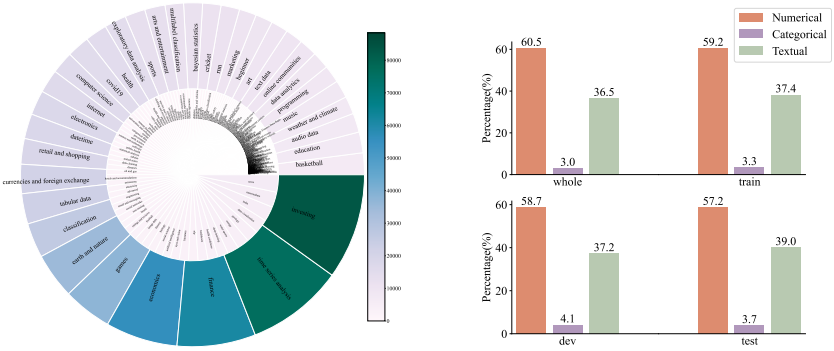
5 Tabular Dataset

As there are no large-scale, high-quality tabular datasets available for pretraining, we have collected our own dataset by crawling from Kaggle. We downloaded CSV tables, omitting empty columns. Specifically, we constructed initial keywords for each domain and extended the set of keywords using WordNet under the same topic. We then searched and downloaded tables using these keywords. For tables originating from the same Kaggle dataset, we attempted to join tables using primary and foreign keys. This process resulted in a 7TB dataset containing 13 billion tabular examples. Statistics about the pretraining data are presented in Table 1. We also present the statistics of the Top-5 domains in this table. Figure 2 shows the distribution of domains and the proportion of cells of different data types for each split of our dataset. We aimed to split our dataset in such a way as to maintain a similar distribution of data types and domains.

6 Experiments and Analyses

6.1 Implementation Details

We train our UniTabE with 32 A100 GPUs in the distributed way. Our model is implemented with the PyTorch v1.12. The Transformer encoder used as the backbone of our model is borrowed from the huggingface "transformers" module. During training, the learning rate is set to be 1e-



(a) Domains Distribution (b) Data Type Distribution

Figure 2: Distribution visualization. The left part (a) demonstrates the distribution of domains and the number of tables in each domain. The right part shows the proportion (cell level) of different data types in train/dev/test splits.

5, the batch size is 64. We use Adam [Kingma and Ba, 2015] as the optimizer with β_1=0.9, β_2=0.999 and ϵ =10^{-8}. We train three variants of our model, $\mathrm{UniTabE}_{base}$, $\mathrm{UniTabE}_{large}$ and $\mathrm{UniTabE}_{xlarge}$. The hidden size and embedding size for $\mathrm{UniTabE}_{base}$ are both set as 768. Its encoder is the 12 layers of self-attention stack with 12 heads. $\mathrm{UniTabE}_{large}$ consists of 24 layers encoder, 16 attention heads. Its hidden size and embedding size are both 1024. $\mathrm{UniTabE}_{xlarge}$ is the 48 layers' version of $\mathrm{UniTabE}_{large}$. We randomly sample a number p from a standard normal distribution to determine the number of randomly masked cells: (1 cell, abs(p) $\leq$ 1.0 or abs(p) $>$ 2.5) , (2 cells, 1.0 $<$ abs(p) $\leq$ 1.25), (3 cells, 1.25 $<$ abs(p) $\leq$ 1.5), (4 cells, 1.5 $<$ abs(p) $\leq$ 1.75), (5 cells, 1.75 $<$ abs(p) $\leq$ 2.0), (7 cells, 2.0 $<$ abs(p) $\leq$ 2.5). Masking single cell is used as the backup option in some cases, such as there are no sufficient column number to mask.

6.2 Benchmarks & Baselines

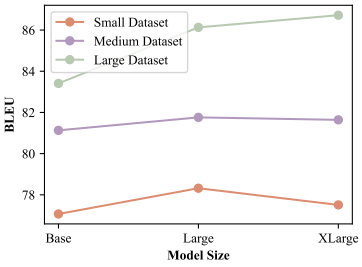


Figure 3: BLEU score of generating textual values by different model size on different dataset size.

Many practical tabular tasks fall into the categories of classification or regression. As such, we evaluate the effectiveness of our pretrained model on these types of tasks. To do this, we have selected several baseline models for comparison. These include: **XGBoost** Despite a large number of proposed neural network architectures for tabular data, the performance gap between them and the "shallow" ensembles of decision trees, like XGBoost, often remains significant in industry. We use it as baseline for classification tasks. Implemented based on the XGBoost package.[3] **TransTab** textualizes columns of the same data type into a text by inserting a space among columns, column names and column values. **TransTab-LSTM** is equipped with the same shallow decoder as our UniTabE. **Linear Regress** is also used to compare the performance for regression tasks. We adopt the Scikit-learn implementation of linear regression in the experiments.

6.3 Results & Analyses

Overall Results. We selected a diverse set of tasks comprising 6 representative classification assignments and 6 regression tasks from Kaggle. To ensure a fair and unbiased evaluation, we deliberately excluded these specific tabular datasets from our pretraining corpus. This precaution was taken to prevent the pretrained model from gaining undue familiarity with the target columns, thereby avoiding any potential bias in the results. The classification tasks and their abbreviated names in this work are Pima Indians Diabetes (**PID**, url), Red Wine Quality (**RWQ**, url), Heart Failure Prediction (**HFP**, url), Health Insurance Cross Sell (**HIC**, url), Eligibility Prediction for Loan (**EPL**, url), and Loan Default Prediction (**LDP**, url). The regression tasks are Medical Insurance Payout

[3] xgboost.sklearn

Table 2: Performance comparison in 12 Kaggle tasks coming from different domains. The dataset name is abbreviated referring to § 6.3. "UniTabE scratch" means the model is trained from scratch without loading pretrained parameters. Bold results are the best.

Method/Dataset	Classification (AUC ↑)						Regression (R2 ↑)					
	PID	RWQ	HFP	HIC	EPL	LDP	MIP	GPP	RSP	CCL	MMP	HPA
XGBoost	0.79	**0.67**	**0.89**	0.85	0.73	0.50	-	-	-	-	-	-
TransTab-LSTM	0.81	0.56	0.74	0.85	0.73	0.50	0.50	0.55	0.64	0.65	0.51	0.47
UniTabE scratch	0.76	0.57	0.61	0.85	0.73	0.52	0.53	0.76	**0.99**	0.72	0.82	0.54
UniTabE finetune	**0.83**	0.66	0.81	**0.86**	**0.78**	**0.53**	**0.75**	**0.99**	**0.99**	**0.96**	**0.87**	**0.58**

Table 3: Evaluation results with AUC on public tabular datasets. Bold results are the best.

Method/Dataset	CG	CA	DS	AD	CB	BL	IO
XGBoost	0.78	0.93	0.54	**0.91**	0.87	0.82	0.71
TransTab	0.73	0.86	0.52	0.90	0.80	0.71	0.73
TransTab-LSTM	0.70	0.85	0.56	0.90	0.72	0.83	0.73
UniTabE scratch	0.76	0.93	0.62	**0.91**	0.85	**0.84**	0.74
UniTabE + XGBoost	**0.79**	**0.93**	0.60	**0.91**	**0.88**	0.83	0.74
UniTabE finetune	**0.79**	**0.94**	**0.66**	**0.91**	**0.88**	**0.84**	**0.76**

(**MIP**, url), Gold Price Prediction (**GPP**, url), Reliance Stock Price (**RSP**, url), Credit Card Limit Prediction (**CCL**, url), Miami Housing Prediction (**MHP**, url), and House Prices - Advanced (**HPA**, url), respectively. The comparison of results among methods is presented in Table 2. As a whole, our UniTabE outperforms the other contenders, particularly excelling in regression tasks. The marked superiority of UniTabE over TransTab-LSTM offers clear evidence that a fine-grained processing of tabular data is more beneficial than straightforward textualization. This is indicative of the efficacy of our model's approach to handling and understanding the nuanced structure of tabular data.

Model Size Analysis. We want to see the impact of the size of UniTabE on performance across different size of dataset. 50 tables are reserved and not included in the pretraining dataset. The results presented in Figure 3 indicate that for larger finetuning datasets, larger models (like $\mathrm{UniTabE}_{xlarge}$) tend to perform better. However, for smaller datasets, the performance of larger models tends to decrease. We notice that $\mathrm{UniTabE}_{large}$ is a balance size for different datasets. Hence, we apply the $\mathrm{UniTabE}_{large}$ to compare the performance with baselines in subsequent experiments.

Out-of Domain Benchmark Datasets. Since those 12 tasks coming from Kaggle might have the similar domains to our pretraining data, we also use other public tabular datasets to further evaluate the efficacy of our method. These datasets are all binary classfication tasks. We also simplify the dataset name and attach the hyperlink for each dataset: credit-g (**CG**, url), credit-approval (**CA**, url), dress-sales (**DS**, url), adult (**AD**, url), cylinder-bands (**CB**, url), blastchar (**BL**, url), insurance-co (**IO**, url). Table 3 presents experimental results on these datasets. Similar to those results in 12 Kaggle tasks, our method also achieves impressive results indicating its superiority in learning the intrinsic semantic information of tabular data.

Ablation Analysis. As seen in Table 2, when compared to the model without pretraining, "UniTabE scratch", our finetuned UniTabE demonstrates a substantial improvement, achieving a total gain of +0.43 and +0.78 for classification and regression tasks respectively. This trend is also mirrored in Table 3. These results underscore the significant advantages brought about by the pretraining phase for handling a diverse range of tabular tasks.

Fill in Missing Value. We evaluate the model's capability to fill in missing values, utilizing the Mean Absolute Error (MAE) metric for numerical columns and the BLEU score for textual predictions. For comparative analysis, we also fine-tune and test the GPT-3 model. The results, outlined in Table 4, indicate a significant improvement over other baseline models. This substantial gain in performance further corroborates the efficacy of our pretrained model in handling missing values, demonstrating its potential for applications in data completion and recovery tasks.

Zero-shot Prediction. Table 5 presents the results of zero-shot prediction. The "Random Initial" approach, which does not load the pretrained parameters, exhibits inferior performance. In contrast, our pretrained model demonstrates strong performance, even without fine-tuning on the target datasets, indicating its promising generalizability. These results also suggest that UniTabE acquires a

Table 4: Performance comparison in filling in missing value. "mean/mode" uses the average in numerical column as prediction, and takes the most common text as prediction for textual column.

Method/Dataset	Regression (MAE ↓)						Text Generation (BLEU ↑)					
	CG	CA	DS	AD	CB	BL	CG	CA	DS	AD	CB	BL
mean/mode	0.81	0.61	0.84	0.78	0.73	0.84	48	62	46	62	65	43
Linear Regress	0.64	0.58	0.91	0.45	0.60	0.51	-	-	-	-	-	-
GPT-3	0.58	0.57	0.69	0.64	0.61	0.46	48	66	39	73	71	82
UniTabE scratch	0.49	0.56	0.83	0.69	0.72	0.38	35	63	28	67	18	75
UniTabE finetune	**0.40**	**0.51**	**0.61**	**0.43**	**0.51**	**0.22**	**59**	**76**	**51**	**80**	**85**	**92**

Table 5: The accuracy of different methods for zero-shot classification.

Method/Dataset	CG	CA	DS	AD	CB	BL	IO
UniTabE finetune	0.75	0.89	0.62	0.86	0.78	0.80	0.94
Random Initial	0.30	0.41	0.50	0.54	0.41	0.26	0.06
Zero-Shot	**0.70**	**0.56**	**0.58**	**0.76**	**0.57**	**0.73**	**0.94**

certain degree of high-level reasoning capabilities through extensive large-scale pretraining, which contributes to its robust performance in zero-shot settings.

Adaption of Incremental Columns. We want to investigate the scalability of various models when confronted with an increasing number of columns. To simulate this scenario, we remove k columns from the original training set and train models using the modified data. Subsequently, we perform inference using the unaltered test set. The results of this experiment are displayed in Table 6. Thanks to the flexibility afforded by the TabUnit component of UniTabE, our model exhibits adaptability to the introduction of new columns with a relatively minor performance deterioration.

Table 6: The percentage(%) of AUC drop facing with incremental columns.

Method/Drop k	1	2
TransTab-LSTM	7.5	11.7
UniTabE scratch	5.3	8.4
UniTabE finetune	**3.5**	**5.8**

Learned Feature + XGBoost. We are interested in examining the synergistic effects of combining the semantic representation derived from UniTabE with traditional machine learning algorithms, such as XGBoost. We integrate the original features with the generated representation vector, and use this composite data as input to XGBoost. The outcomes are displayed in Table 3. These results indicate that the neural feature acts as a beneficial supplement, leading to a slight performance enhancement in most instances.

7 Conclusions

In this research, we investigate the challenge of pretraining large-scale models specifically on tabular data. To address the inherent difficulties in pretraining over tabular data, we introduce UniTabE, a flexible approach capable of modelling arbitrary tables using a single neural network architecture. We have also collected a substantial dataset, comprising 13B examples spanning various domains, for pretraining purposes. To ascertain the effectiveness of our methodology, we carry out extensive experiments across 19 downstream tasks, with the outcomes confirming the superiority of our approach. Additionally, we explore issues related to the practical application of our method, including zero-shot prediction, adaptation to incrementally added columns, and combination of learned representation and XGBoost.

8 Limitations

In this study, our primary objective is to learn the semantic representation of tabular data via a pretraining process. To maximize the knowledge retention within the encoder, we integrate our model with a comparatively weak decoder. In theory, the shallow decoder may constrain performance in downstream applications as it serves as a high-level reasoning module that adjusts to various tasks. Nevertheless, the experimental results presented earlier illustrate that our model yields substantial performance improvements even with this weak decoder. This outcome, in turn, substantiates the success of our method. It should be noted that our focus is solely on certain data types common to most scenarios, such as textual, numerical, and categorical values, without considering other modalities like images, videos, or audio.

References

S. Ö. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

M. Bavarian, H. Jun, N. Tezak, J. Schulman, C. McLeavey, J. Tworek, and M. Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

S. Chen, Y. Hou, Y. Cui, W. Che, T. Liu, and X. Yu. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. *arXiv preprint arXiv:2004.12651*, 2020.

Z. Cheng, H. Dong, F. Cheng, R. Jia, P. Wu, S. Han, and D. Zhang. Fortap: Using formulae for numerical-reasoning-aware table pretraining. *arXiv preprint arXiv:2109.07323*, 2021.

Z. Cheng, H. Dong, Z. Wang, R. Jia, J. Guo, Y. Gao, S. Han, J.-G. Lou, and D. Zhang. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.78. URL `https://aclanthology.org/2022.acl-long.78`.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

J. M. Eisenschlos, M. Gor, T. Müller, and W. W. Cohen. Mate: Multi-view attention for table transformer efficiency. *arXiv preprint arXiv:2109.04312*, 2021.

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

J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

Z. Jiang, Y. Mao, P. He, G. Neubig, and W. Chen. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 932–942, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.68. URL `https://aclanthology.org/2022.naacl-main.68`.

Y. Katsis, S. Chemmengath, V. Kumar, S. Bharadwaj, M. Canim, M. Glass, A. Gliozzo, F. Pan, J. Sen, K. Sankaranarayanan, and S. Chakrabarti. AIT-QA: Question answering dataset over complex tables in the airline industry. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 305–314, Hybrid: Seattle, Washington + Online, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-industry.34. URL `https://aclanthology.org/2022.naacl-industry.34`.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

G. Liu, J. Yang, and L. Wu. Ptab: Using the pre-trained language model for modeling tabular data. *arXiv preprint arXiv:2209.08060*, 2022.

J. Salazar, D. Liang, T. Q. Nguyen, and K. Kirchhoff. Masked language model scoring. *arXiv preprint arXiv:1910.14659*, 2019.

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

F. Wang, K. Sun, M. Chen, J. Pujara, and P. Szekely. Retrieving complex tables with multi-granular graph representation learning. *arXiv preprint arXiv:2105.01736*, 2021a.

Z. Wang and J. Sun. Transtab: Learning transferable tabular transformers across tables. *arXiv preprint arXiv:2205.09328*, 2022.

Z. Wang, H. Dong, R. Jia, J. Li, Z. Fu, S. Han, and D. Zhang. Tuta: tree-based transformers for generally structured table pre-training. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1780–1790, 2021b.

D. Ye, Y. Lin, P. Li, M. Sun, and Z. Liu. A simple but effective pluggable entity lookup table for pre-trained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 523–529, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.57. URL https://aclanthology.org/2022.acl-short.57.

P. Yin, G. Neubig, W.-t. Yih, and S. Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, 2020.