TABGPT: TOWARD A TABULAR FOUNDATION MODEL

Felix Wick
Panasonic R&D Center Germany
felix.wick@eu.panasonic.com

Ulf Mertens
Panasonic R&D Center Germany
ulf.mertens@eu.panasonic.com

Naoki Setoguchi
Panasonic Connect
setoguchi.naoki@jp.panasonic.com

Jun Matsuzaki
Panasonic Connect
matsuzaki.jun001@jp.panasonic.com

Yoshito Tsuruda
Panasonic Connect
tsuruda.yoshito@jp.panasonic.com

Ryuji Noda
Panasonic Connect
noda.ryuji@jp.panasonic.com

Tawei Chang
Panasonic Connect
chang.ta@jp.panasonic.com

September 12, 2024

ABSTRACT

Transfer learning and foundation models have proven highly effective in domains involving homogeneous, unstructured data, such as computer vision and natural language processing. However, their application to structured or tabular data remains limited due to the inherent heterogeneity of such datasets. We propose a simple yet effective method for addressing the challenges associated with tabular data modeling by combining column embeddings generated with existing large language models and an encoder transformer architecture with a flexible prediction head. This approach integrates the vast knowledge of large language models with a robust numerical model backbone for pre-training across diverse datasets and tasks.

1 Introduction

Transfer learning and foundation models based on deep learning methods are ubiquitous in the homogeneous, unstructured data regime, such as computer vision and natural language processing [1, 2]. However, for structured or tabular data, its notorious heterogeneity hinders a widespread application [3].

There are several compelling reasons why an effective tabular foundation model would be highly desirable:

- **Strong predictions with limited data**: By leveraging a pre-trained model, fine-tuning or even few-shot learning can yield accurate predictions, even with small training datasets.
- **Leveraging LLM knowledge**: Such a model could benefit from the vast knowledge embedded in large language models (LLMs), enhancing its performance, especially in terms of data imputation.
- **Feature learning**: It would also facilitate advanced feature learning, improving the model's ability to capture relevant patterns in the data.
- **Data integration**: And arguably most importantly, it would solve the fundamental issue of data integration for tabular data sets, i.e., consolidating different data schemas and semantics, which requires quite some manual efforts in current models.

However, all current approaches struggle to fully address these requirements. In particular, the integration of extensive world knowledge from LLMs with a robust numerical model backbone for pre-training over various data sets and tasks remains a significant challenge.

Our Contributions We propose a simple approach, TABGPT[1], combining the following two ready-to-use components:

1. **Leveraging LLM for Column Embeddings**: We utilize an LLM to generate column embeddings, which facilitates consistent training across various datasets and tasks. This approach effectively addresses data integration challenges while incorporating the extensive knowledge and feature learning capabilities of LLMs.
2. **Transformer Backbone**: Our model employs a typical encoder-only transformer architecture with classification (or regression) head for making predictions. By omitting positional encoding, we preserve the permutation invariance of columns, enabling the model to directly input column embeddings into attention blocks. Additionally, we enhance our approach by incorporating a target/task description as an extra feature embedding, with positional encoding included for clear identification.

2 Related Work

Current approaches may be classified into two broad categories, the ones utilizing pre-trained LLMs and tabular transformers [4]. As we will describe in Section 3, our proposed approach can be interpreted as combination of these two classes.

2.1 Utilizing pre-trained LLMs

Approaches utilizing pre-trained LLMs typically transform each row of a table into text, what is referred to as serialization. The produced text is then tokenized and fed into a pre-trained LLM. Thereby, each row is processed independently and in principle, this approach requires no dedicated training. But usually there is some fine-tuning of the used LLM on tabular tasks.

Architecture-wise, both encoder-only [5] and decoder-only [6, 7, 8] LLMs are possible, where at least the encoder-only LLMs require a head with a numeric loss, that needs to be fine-tuned. An advantage of the conditional generative nature of decoder models is the possibility of in-context learning, potentially interesting here in the form of few-shot prompting.

The serialization can also be done with an LLM, which can be another one than for the subsequent prediction step. There are different serialization formats and strategies, but generally, the column values are combined with the respective column name or an augmenting description.

While these methods solve the data integration issue of tabular data, LLMs pre-trained on text have inherent limitations in capturing complex numerical dependencies.

2.2 Tabular Transformers

In order to dedicatedly learn numerical correlations, one often adapts transformer-like architectures [9, 10, 11] (or similarly graph attention networks [12]), because the permutation invariance of the self-attention mechanism reflects the arbitrary ordering of rows and columns in tabular data sets. It should be noted here that the causal masking used in transformer decoders partly destroy the permutation invariance of columns. So, encoder-only transformers without causal masking are preferable here.

The training across tables with different columns requires a mechanism to transform individual table rows into sequences of numerical representations. Usually, this is done with task-specific featurizers, which can be compared to embeddings in language models. A drawback of these approaches is that its featurizers need to be specified for each task, what makes it only partly practical as foundation model. Also, there is no mechanism to make use of the world knowledge of LLMs.

Another particular challenge of models for tabular data is how to treat numerical input values, especially continuous numbers, because its vector representations in a high-dimensional embedding space will hardly preserve the simple natural ordering. One possibility to overcome this issue is to use a single specific token for all numbers and multiply its learned embedding, i.e., vector representation, with the respective numerical value [13]. Another, more flexible,

[1]Code: `https://github.com/FelixWick/tabGPT`

possibility is to multiply the numerical values with the corresponding embedding of the column name or description [12]. However, this requires a tokenization that uses entire columns as individual tokens.

3 Method

To overcome the limitations of the existing methods described in Section 2, we propose TABGPT, which consists of two steps: First, we generate numerical vector representations from the individual columns of the table for each row by means of an LLM, then we train an encoder-only transformer model with these column embeddings as inputs. An overview of TABGPT can be seen in Figure 1.

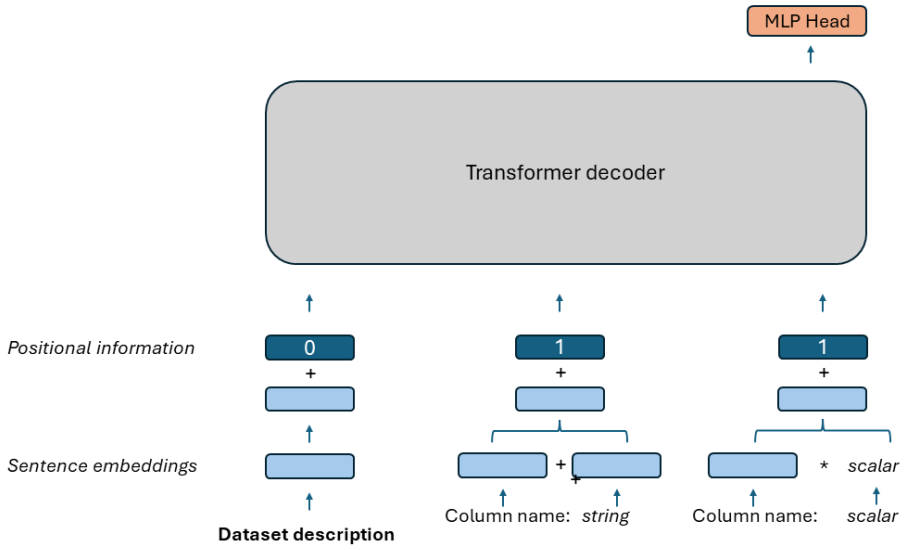

Figure 1: Sketch of the TABGPT method. Externally generated sentence embeddings for dataset or target descriptions, column names or descriptions, and row values for categorical columns are used as inputs to an encoder-only transformer with a simple multi-layer perceptron (MLP) head for numerical predictions. Hereby, embeddings of categorical values are added element-wise and scalar numerical values are multiplied to the corresponding column embeddings for each row. Also, simple positional encodings are added only to distinguish the target embedding from the other inputs.

3.1 Column Embeddings

For each column in the dataset, we individually generate sentence embeddings. This process involves using the column name or, if available, an additional description as input to a forward pass through a GPT-2 model (or any other text-generating LLM) pre-trained on language. We then extract the mean of the last hidden states for each token in the sequence, which forms the sentence embedding for that specific column.

For each row of the column at hand, we then combine the sentence embedding of the column name or description with the row value, where the exact combination depends on the nature of the values.

3.1.1 Categorical Values

For categorical inputs, we perform the same embedding procedure as for the column names (or descriptions) for each unique category. Then we element-wise add the value embeddings for each row and the column name embeddings, where the latter are the same for each row.

Table 1: Results of our approach for three different Kaggle competitions, compared to the respective best leaderboard results, where we have dropped reported leaderboard results that seem unreasonable (probably due to target leakage in the test data sets). The evaluation metric for house prices [14] and store sales [15] is root mean squared logarithmic error (RMSLE), the one for spaceship Titanic [16] accuracy.

	house prices	store sales	spaceship Titanic
TABGPT	0.138	0.450	79.9%
Kaggle leaderboard	0.113	0.379	85.3%

3.1.2 Numerical Values

For numerical inputs, we multiply each element of the column name embedding with the numerical value of the row at hand.

One difficulty lies in the treatment of zero values, because the multiplication with zero nullifies the column name embedding. A way to circumvent this is to use a dedicated category just for the zero value and follow the same procedure as for categorical values, i.e., embed the zero and add the resulting vector to the column embedding.

In order to generalize over different magnitudes of similar numerical columns in different data sets, it is beneficial to scale the numerical values to a given range. One possibility is to divide each value by the absolute value of the maximal positive or minimal negative value (whatever is larger) of the column.

3.1.3 Target Encoding

When training several data sets and tasks together, it is beneficial to include a target or task description in the model inputs to identify different target scales or model dynamics. For this, we perform the same embedding procedure as for the column names (or descriptions).

This can be interpreted as a way to kind of prompt the model at inference time to a specific task or target.

In order to distinguish the target embedding from the column embeddings, we add a simple positional encoding to the different vectors: 0 for the target and 1 for all the columns.

3.2 Transformer Model

The backbone for our proposed tabular foundation model is an encoder-only transformer, in our TABGPT implementation we use an architecture similar to GPT-2. But unlike typical language models, TABGPT operates without tokenization, embedding, and positional encoding. Instead it directly leverages the concatenation of the before created embeddings for each column along the sequence dimension.

To effectively generate predictions for classification or regression tasks, we replace the language model head with a classification or regression head and use a corresponding loss function (mean squared error for regression and cross-entropy for classification tasks). So far, we have trained different versions of TABGPT for regression and classification tasks, but in principle, one can use common model weights for all but the last layer.

4 Experiments

As feasibility check for our proposed method, we conducted several experiments with our TABGPT implementation. In the following, we show its ability to generally predict tabular tasks, to work in a cross-training setup, and to create decent predictions with limited training data.

Some details about our implementation can be found in Appendix A. All the models described in this section were trained from scratch, but it is possible to use language-pre-trained weights with our implementation.

4.1 General-Purpose Usage

Table 1 shows the results of tabGPT for three different Kaggle competitions [14, 15, 16]. Details of the models and data sets can be found in Appendices B.1, B.2, and B.3. The comparisons with the respective Kaggle leaderboard results show that tabGPT can produce predictions with an accuracy similar to prevalent methods.

Table 2: Results of our approach for the test data sets of four different tasks [14, 15, 17, 18], once from a common cross-training, once from individual trainings on the corresponding training data sets only, and once from a fine-tuning on the respective training data set following a pre-training on the other three. Evaluation metric for all is RMSLE.

	house prices	store sales	bicycles count	demand forecasting
cross-training	0.19	0.23	0.34	0.62
individual training	0.14	0.25	0.30	0.68
fine-tuning	...	...	...	...

4.2 Cross-Training

The most important requirement for a tabular foundation model is that one can train across different data sets and tasks, i.e., tables with different columns and targets. To test this, we have performed a common training on four different regression tasks [14, 15, 17, 18]. Details of the model and data sets can be found in Appendices B.1, B.2, B.4, B.5, and B.6.

The comparison of the results with the corresponding results of individual training, shown in Table 2, indicate that TABGPT can be trained across data sets and tasks without significantly loosing accuracy. It should be noted that there was no manual data integration needed to combine the different data sets in a common training.

4.3 Fine-Tuning

One of the key advantages of a tabular foundation model is the ability to produce good predictions with limited training data for a specific task, with the eventual goal of few-shot in-context learning, or even zero-shot learning. But this presumably requires lots of data sets and tasks with close similarities to be included in the training. On the other hand, a continuation of a training with a new data set or a split in a bigger pre-training on many tasks and data sets and a fine-tuning on a new and potentially small data set is feasible even with a small multi-task setup.

To show the effectiveness of the pre-train and fine-tune approach, we have repeated the cross-training experiment of Section 4.2 four times, while always leaving one task out of the pre-training and fine-tune on it afterwards. The comparison of the results with the corresponding results of a pure pre-training are shown in Table 2.

5 Conclusion and Outlook

We propose TABGPT, a method to be used as backbone for a tabular foundation model. It consists of two parts, column embeddings generated from an LLM and an encoder-only transformer adapted for numerical predictions.

Please note that this is merely a proposal for the model architecture of a future foundation model, including some non-extensive feasibility studies. We neither state to have already built such a foundation model nor claim state-of-the-art performance. There is lots of work to be done toward a full tabular foundation model, especially in terms of data sets to be used for pre-training, but we hope this study will serve as an additional step in that direction.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, 2012, Volume 1, Pages 1097–1105.

[2] Abhimanyu Dubey et al. The Llama 3 Herd of Models. In *arXiv preprint*, 2024, arXiv:2407.21783 [cs.AI].

[3] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS)*, 2022, Article No. 37, Pages 507–520.

[4] Maximilian Schambach. Towards Tabular Foundation Models: Status quo, challenges, and opportunities. In *HAL archive*, 2024, hal-04440710.

[5] Zifeng Wang, Chufan Gao, Cao Xiao, and Jimeng Sun. MediTab: Scaling Medical Tabular Data Predictors via Data Consolidation, Enrichment, and Refinement. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence (IJCAI)*, 2024.

[6] 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 *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, 2023, PMLR, Volume 206, Pages 5549–5581.

[7] Xumeng Wen, Han Zhang, Shun Zheng, Wei Xu, and Jiang Bian. From Supervised to Generative: A Novel Paradigm for Tabular Deep Learning with Large Language Models. In *arXiv preprint*, 2024, arXiv:2310.07338 [cs.LG].

[8] Josh Gardner, Juan C. Perdomo, and Ludwig Schmidt. Large Scale Transfer Learning for Tabular Data via Language Modeling. In *arXiv preprint*, 2024, arXiv:2406.12031 [cs.LG].

[9] Zifeng Wang and Jimeng Sun. TransTab: Learning Transferable Tabular Transformers Across Tables. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NIPS)*, 2022, Article No. 210, Pages 2902–2915.

[10] Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. XTab: Cross-table Pretraining for Tabular Transformers. In *Proceedings of The 40th International Conference on Machine Learning*, 2023, PMLR, Volume 202, Pages 43181–43204.

[11] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

[12] Myung Jun Kim, Léo Grinsztajn, and Gaël Varoquaux. CARTE: Pretraining and Transfer for Tabular Learning. In *arXiv preprint*, 2024, arXiv:2402.16785 [cs.LG].

[13] Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. xVal: A Continuous Number Encoding for Large Language Models. In *arXiv preprint*, 2023, arXiv:2310.02989 [stat.ML].

[14] Anna Montoya, DataCanary. House Prices - Advanced Regression Techniques. Kaggle, 2016, `https://kaggle.com/competitions/house-prices-advanced-regression-techniques`

[15] Alexis Cook, DanB, inversion, Ryan Holbrook. Store Sales - Time Series Forecasting. Kaggle, 2021, `https://kaggle.com/competitions/store-sales-time-series-forecasting`

[16] Addison Howard, Ashley Chow, Ryan Holbrook. Spaceship Titanic. Kaggle, 2022, `https://kaggle.com/competitions/spaceship-titanic`

[17] Department of Transportation (DOT). Bicycle Counts for East River Bridges. 2017, `https://data.cityofnewyork.us/Transportation/Bicycle-Counts-for-East-River-Bridges-Historical-/gua4-p9wg/about_data`

[18] Felix Wick. 2023, `https://github.com/FelixWick/demand_forecasting_simulation`

[19] Andrej Karpathy. minGPT. 2023, `https://github.com/karpathy/minGPT`

A TABGPT Implementation

For ease of comprehension, we fork our TABGPT implementation from MINGPT [19], a popular, publicly available re-implementation of GPT.

As embedding or model dimension, we use GPT-2 size of 768, reflecting our usage of GPT-2 for column embeddings. The used number of layers and attention heads can differ for different models and are described below.

Compared to the original MINGPT implementation, we made the following adaptions:

- no use of any tokenization, instead GPT-2 call to generate sentence embeddings for column names or descriptions, categorical values, and target or task descriptions
- dropped the causal masking in the self-attention mechanism to enable full permutation invariance of columns
- dropped positional encodings, instead added simple positional encoding to distinguish column from target description embeddings
- replaced language model head, loss function, and generation mechanism to reflect tabular prediction setup

In our implementation, there are different options for model configurations in terms of number of layers and number of self-attention heads. For all the experiments described here, we 4 layers and 4 self-attention heads. As embedding dimension, we use the one of GPT-2, which is 768.

B Data Set Details

All data sets used for this work are openly available. We have put emphasis on regression tasks, both because there are underrepresented in former studies and are arguably more important for many real-world applications.

For the three Kaggle data sets, described below in B.1, B.2, and B.3, we use the train data sets for training and the test data sets for the predictions stated in the results, respectively.

As task or target descriptions, we simply use a short name for the described experiments: house prices, store sales, bicycles count, retail demand forecasting, spaceship titanic. In the multi-task mode described in Section B.6, this helps to distinguish the different target scales and task dynamics. So, in a larger multi-task model, a more detailed description will be beneficial. In the single-task mode, it is completely irrelevant.

B.1 House Prices

As this is a regression task [14], we use a typical MSE loss in the training. But we take the logarithm (to be exact, plus 1) of the target for the training, in order to optimize for the RMSLE metric used for the evaluation of the results.

We use all available 79 features for our model, without any feature engineering. We simply decide for each feature by its type whether it is used as categorical (see Section 3.1.1) or numerical (see Section 3.1.2 one in our model. While the number of features is relatively large, the number of training samples is comparably low (1460 only), what makes it a challenging task.

Instead of the plain column names, we can alternatively use the column descriptions, as well as the descriptions of the different values for the categorical columns, which are provided on the web page, as input text for the column embeddings. While we anticipate this to be beneficial in a vast multi-task setup, it makes no significant difference in the single-task and small multi-task setup, especially when training from scratch without building on top of a language pre-training. So, we used the plain column and category names for our results here.

B.2 Store Sales

As this is a regression task [15], we use a typical MSE loss in the training. But we take the logarithm (to be exact, plus 1) of the target for the training, in order to optimize for the RMSLE metric used for the evaluation of the results.

We perform some feature engineering to extract seasonality information from the date (namely the weekday, day in the month, and day in the year), compute the days before or after the most important holiday, and calculate an exponentially weighted moving average (EWMA) of the past sales for each store-family-weekday combination.

We use the following features as inputs for our model:

- categorical: STORE_NBR (store), FAMILY (product group), weekday

- numerical: ONPROMOTION (items on promotion), DCOILWTICO (oil price), day in month, day in year, days around Primer Grito de Independencia, EWMA of past sales

For the training, we only use data from May 2017 onward, because using earlier data did not improve the validation score.

B.3 Spaceship Titanic

As this is a binary classification task [16], we use a typical cross-entropy loss in the training.

We need to do some light feature engineering for this data set to extract the group information from the PASSENGERID column, which we use to build features reflecting the group size and number of individuals in the group. Similarly, we decompose the information in the CABIN column in cabin size and DECK, NUM, SIDE.

We use the following features as inputs for our model:

- categorical: single group or not, DECK, SIDE, single cabin or not, HOMEPLANET, CRYOSLEEP, DESTINATION, VIP
- numerical: group size, number in group, cabin size, NUM, AGE, ROOMSERVICE, FOODCOURT, SHOPPING-MALL, SPA, VRDECK

B.4 Bicycles Count

As this is a regression task [17], we use a typical MSE loss in the training. But we take the logarithm (to be exact, plus 1) of the target for the training, in order to optimize for the RMSLE metric used for the evaluation of the results.

We perform some light feature engineering to extract the weekday from the date and use the following features as inputs for our model:

- categorical: weekday, bridge
- numerical: Precipitation, High Temp (F), Low Temp (F)

We use data from April 2017 until September 2017 for the training data set (732 rows), and from October 2017 for the test data set (124 rows).

B.5 Demand Forecasting

We have simulated [18] daily data points for 7 products and 3 stores from October 2021 until September 2022, where we use data until March 2022 for training (6374 rows) and data from April until September 2022 for testing (3302).

As this is a regression task, we use a typical MSE loss in the training. But we take the logarithm (to be exact, plus 1) of the target for the training, in order to optimize for the RMSLE metric used for the evaluation of the results.

We perform some light feature engineering to extract seasonality information from the date (namely the weekday, day in the month, and day in the year).

We use the following features as inputs for our model:

- categorical: product ID, product group ID, location ID, type of promotion, weekday
- numerical: normal price, sales area, sales price, day in month, day in year

B.6 Multi-Task Regression

For our cross-training experiments, we combine the data sets described in Sections B.1, B.2, B.4, and B.5, where we extend the data for B.2 to start from November 2016, but use only three stores (1, 2, and 3) and three product families (LIQUOR,WINE,BEER, EGGS, and MEATS), just to have a more balanced number of training samples for the different tasks. Also, in order to avoid lots of padding for the other data sets to get the same column dimension, we only use the 10 arguably most important features for B.1, which does not change the quality of the model significantly.