PORTAL: Scalable Tabular Foundation Models via Content-Specific Tokenization

Marco Spinaci
SAP France
marco.spinaci@sap.com

Marek Polewczyk
SAP SE
marek.polewczyk@sap.com

Johannes Hoffart
SAP SE
johannes.hoffart01@sap.com

Markus C. Kohler
SAP SE
markus.kohler01@sap.com

Sam Thelin
SAP SE
sam.thelin@sap.com

Tassilo Klein
SAP SE
tassilo.klein@sap.com

Abstract

Self-supervised learning on tabular data seeks to apply advances from natural language and image domains to the diverse domain of tables. However, current techniques often struggle with integrating multi-domain data and require data cleaning or specific structural requirements, limiting the scalability of pre-training datasets. We introduce PORTAL[1] (Pretraining One-Row-at-a-Time for All tabLes), a framework that handles various data modalities without the need for cleaning or preprocessing. This simple yet powerful approach can be effectively pre-trained on online-collected datasets and fine-tuned to match state-of-the-art methods on complex classification and regression tasks. This work offers a practical advancement in self-supervised learning for large-scale tabular data.

1 Introduction

The introduction of BERT Devlin et al. (2019), powered by the transformer architecture and enriched by extensive unstructured data from the internet, marked a transformative era in deep learning-based natural language processing. On the other hand, the impact of neural architectures on structured tabular data has been less significant. Tabular data, widely used in many enterprises, is a common target for machine learning applications but handling such data at scale presents challenges requiring scalable analytical solutions.

Currently, tree-based ensemble methods, particularly boosting techniques, excel in managing tabular data. Methods like XGBoost Chen and Guestrin (2016) and CatBoost Prokhorenkova et al. (2018) generally outperform deep learning models in both effectiveness and resource efficiency, maintaining a strong dominance Grinsztajn et al. (2022).

Despite successes in natural language, neural network methods for tabular data often underperforms compared to gradient boosting methods, possibly due to the limited possibilities for transfer learning, stemming from stringent preprocessing requirements. While there are attempts to adapt transformer models for tabular data Yang et al. (2024); Ye et al. (2024); Yan et al. (2024), few achieve results

[1] Code will be released at `https://github.com/SAP-samples/portal`

Table Representation Learning Workshop at NeurIPS 2024.

comparable to gradient boosting models, with successes generally confined to specific settings Hollmann et al. (2023); Kim et al. (2024).

To address these challenges, we propose PORTAL (**P**retraining **O**ne-**R**ow-at-a-**T**ime for **A**ll tab**L**es), a transformer encoder-based model pre-trained on diverse tabular data sources, with data types including text, numbers, and dates. PORTAL demonstrates superior performance (especially on text-heavy datasets) with respect to both boosting (XGBoost) and transformer (CM2) approaches, while being comparable to, and sometimes better than, state-of-the-art approaches (CatBoost, AutoGluon, CARTE). Our core contributions are threefold:

First, we introduce an adaptable encoding structure that eliminates the need for normalization or special handling of missing values and outliers.
Second, we implement a pre-training scheme using masked cell modeling that generalizes effectively to fine-tuning, reducing the gap between pre-training and fine-tuning stages.
Third, we provide a comparative analysis with robust classical baselines to highlight performance gaps between traditional tree-based methods and our transformer-based approach.

2 Related Work

Several approaches have been proposed to pre-train models on large amounts of tabular data. Here, we highlight those most relevant to our work.

Herzig et al. (2020) was among the first to adapt BERT's architecture to encode tables for question answering, yet it was constrained to smaller tables. Zhu et al. (2023) developed a transformer model divided into an encoder, backbone transformer, and decoder, but with limited scalability as only the transformer was shared across pre-training datasets. Despite these advancements, traditional tree-based models like CatBoost still exhibited superior performance.

Following these, Hollmann et al. (2023) pre-trained a transformer on synthetic datasets targeting high accuracy and swift inference, though its utility was limited to smaller tables and specific classification tasks, making it less relevant to broader applications. Yang et al. (2024) utilized TabUnit modules that treat table cells as key-value pairs, combining transformer and LSTM decoders, pre-training on a vast 13B-row dataset to slightly edge out XGBoost.

Ye et al. (2024) advanced a positional-encoding-free transformer using masked cell modeling on large datasets, with nuances in text and numerical data handling that, while innovative, led only to marginal performance gains. Meanwhile, Yak et al. (2023) introduced a "MapTransformer" architecture that converts table rows into feature embeddings sequences, albeit restricted to a select few large datasets for pretraining.

Yan et al. (2024) further explored transformer encoders pre-trained on discretized numerical features with the C4.5 algorithm but required labeled datasets, thus limiting its scalability. Finally, Kim et al. (2024) presented a graph-based encoder forming data into star-shaped graphlets for fine-tuning, achieving state-of-the-art results with an ensemble of multiple models. Our experiments indicate that while ensemble methods are necessary for peak performance, satisfactory results are achievable with a standalone model setup.

3 Method

Preliminaries:

In this study, we implement an encoder-only transformer as our primary architecture, optimally adapted for generating embeddings from heterogeneous data types. The model takes feature values and dataset descriptors, such as column names, as inputs. Initially, dedicated type-specific encoders convert each type of data into fixed-length embeddings. These embeddings are then combined with column metadata information and transformed into a unified latent space, which feeds directly into the transformer's backbone. The output from the transformer is then decoded for each feature individually, allowing specific processing based on the feature type.

A key characteristic of our model is its capacity to handle data on a row-by-row basis without needing context from the rest of the table that the row comes from (or even to know if there was any other data beyond this one row). The training process of our model is executed in two main stages. In the

first stage, the pre-training phase, the model sees randomly masked tokens, similarly to the process popularized by BERT pre-training Devlin et al. (2019). In this step, the model sees a version of the data where content is masked, but column names are left unmasked, to "prompt" the model for completion. This is needed, since no other positional encoding is used, and therefore the model could otherwise not determine which masked value to predict for which cell when more than one cell is masked. The model is then fine-tuned on downstream tasks, typically consisting of either regression or classification. All of the weights are shared between the pre-training stage and the fine-tuning one except, possibly, the last output layer.

The conceptual framework of our method is structured into three sequential segments: encoding, backbone, and decoding. Detailed descriptions of each segment are elaborated in the subsequent sections of this paper. For a graphical representation of the proposed PORTAL approach, readers are directed to refer to Fig. 1, which visually outlines the structure and flow of our methodology.

Throughout the discussion, we denote the hidden dimension of the backbone transformer model by d.

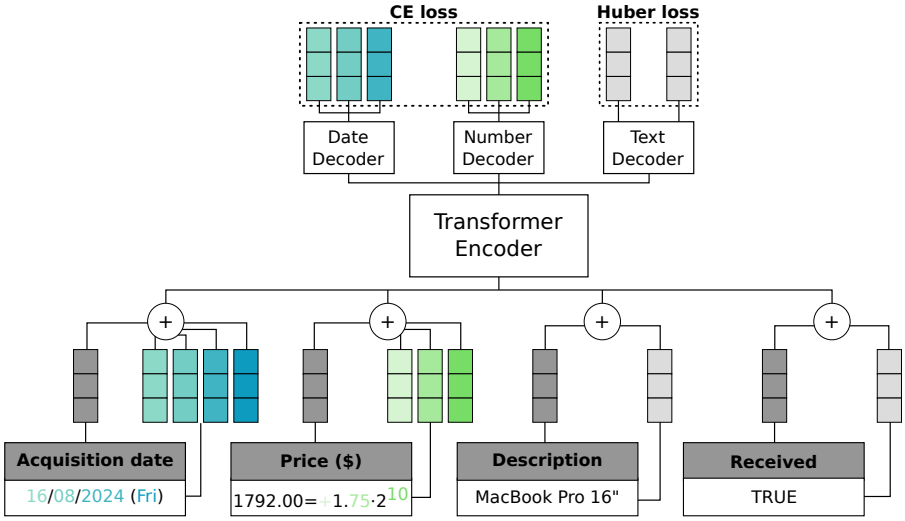

Figure 1: **Schematic illustration of PORTAL architecture:** Example on a row with 4 columns. Blue cells denote custom encodings/decodings for date (day, month, year, day of the week, and holidays). Green cells correspond to numbers (sign, fraction, and exponent), and gray cells correspond to text embeddings. Dark gray cells are column name embeddings. All values are processed via a trainable linear/embedding layer before being aggregated by sum. In the output layer, similar decoding layers are applied before feeding the outputs to cross-entropy or Huber loss in training (see 3.3 for details).

3.1 Encoding

Depending on the type of input data, distinct encoding mechanisms are employed. Specifically, we introduce three specialized encodings tailored for the primary atomic data types: text, numerical values, and dates. For each cell in a table row, only one type of embedding—date, numerical, or string—is generated and summed with the header embedding derived from the column name or meta information. At the end of the encoding process, each input row cell is represented by a single vector. Subsequently, all token embeddings are concatenated and passed to the backbone (in our case, transformer encoder).

Text: For cells that include text data, we employ a pre-trained large language model (LLM) for sentence embedding. Similar techniques were also used in other papers, such as Yak et al. (2023); Kim et al. (2024); Ye et al. (2024); however, the details of how information from cell and content are mixed together differ. In PORTAL, we separately embed text content and column name via the same LLM. Both types of embeddings then undergo processing via two distinct trainable linear layers configured to deliver output in dimension d. This alignment is necessary to allow the model to distinguish between column name embeddings and text content embeddings.

Date: For date cell values, the day, month, and year (clipped to the 1950-2050 range) are individually treated as integers and embedded in dimension d. As part of the automated feature engineering process, d-dimensional embeddings for the day of the week are incorporated, along with those of binary indicators that denote whether a date corresponds to a public holiday in any major countries and territories. These vectors are then summed up together with the embedding of the column name (followed by the same linear adaptation layer as above).

Numerical: Numerical values in our model are expressed using scientific notation: $x = \pm\alpha \cdot 2^{\beta}$, where α ranges within [1, 2) and $\beta \in \mathbb{Z}$ is an integer. Practically, we limit β to the range of $-127 \leq \beta \leq 127$, mirroring the constraints of single precision floating-point arithmetic. Both the sign and β, as discrete entities, are embedded in d dimensions. The fractional component is segmented into K uniformly distributed bins using a method of soft binning. These embeddings are then also summed up together with the column name embedding.

This numerical encoding enjoys multiple advantageous properties:

i) Adaptability to Varying Scales: This method is effective across diverse data scales. For datasets spanning multiple orders of magnitude, the most critical components are the exponent and sign. For more uniformly scaled data, the fractional part gains prominence. Unlike typical normalization methods, both orders of magnitude difference and small scale ones can be captured.

ii) Outlier Robustness: Unlike other scaling techniques such as min/max scaling, having one (or many) outliers within a dataset does not compromise the encoding of other rows.

iii) Versatility for Individual Observations: The encoding is practical for single data rows, making it suitable both for pre-training on rows provided as sets of key-value pairs and for zero-shot inference. This improves upon methods like quantile encoding, which do not make sense outside a reference dataset and typically require multiple unique values to be present.

3.2 Backbone

PORTAL employs a modified version of the standard transformer encoder Vaswani et al. (2017) as the structural backbone. This model, originally designed for natural language processing tasks, has been adapted to align with the nuances of tabular data. Specifically, with respect the BERT Devlin et al. (2019) implementation, we employ two important modifications. *First*, we have substituted the traditional positional encodings in BERT with column name embeddings. This adaptation allows our model to recognize and interpret the structural attributes of tabular data, where the importance and arrangement of data are dictated by the name of the corresponding columns rather than their sequential order. *Second*, we have omitted the initial `[CLS]` token that is standard in BERT's architecture. In BERT's typical applications, this token is needed during pre-training for the next sentence classification task. However, as our model does not engage in such tasks, the `[CLS]` token would never have any loss associated to it, rendering it superfluous.

3.3 Decoding

Decoding transforms the tokens that have gone through the backbone encoder into features similar to those in the encoding. For dates, we only predict day, month, and year. For numbers, we make a small but crucial change: in the fraction part, we do not predict the raw value $\alpha \in [1, 2)$ but predict either $\alpha - 1 \in [0, 1)$ or $2 - \alpha \in (0, 1]$, depending on the parity of β. This can be expressed as $\tilde{\alpha} = (-1)^{\beta}\left(\alpha - \frac{3}{2}\right) + \frac{1}{2}$. This ensures the target $\tilde{\alpha}$ is a continuous function of x, which has theoretical advantages for computing the derivative of the loss and has been empirically proven to perform better for regression tasks (see Tab. 4 for details).

3.4 Training

Our methodology relies on a multi-task learning objective, incorporating contextually selected loss components based on input type. Specifically, we integrate four distinct loss measures: binary cross-entropy, cross-entropy, Huber loss, and L^2 loss, to enhance the model's adaptability and accuracy across different tasks. We employ the following loss configuration:

i) Cross-entropy loss is used for discrete output fields, including day, month, year, sign, and exponent.
ii) Binary cross-entropy loss is designated for the fraction part of numerical values, following its

piecewise linear transformation detailed in Section 3.3.
iii) Huber loss is applied to predictions of text embeddings.
iv) L^2 **loss** is employed for tasks involving a single 1-dimensional head, specifically for regression during fine-tuning.

Pre-training: Pre-training involves masked prediction modeling. Mimicking the original BERT Devlin et al. (2019) and follow-up works (e.g. RoBERTa Liu et al. (2019)), given a row for pre-training, we select each cell with a fixed probability for masking. Following recent trends (e.g., Wettig et al. (2023)), we use a higher 30% probability of masking. For each cell selected for masking, we either zero out its content vector (with 80% probability), leave it unchanged (with 10% probability), or replace it with a random value (with 10% probability). Unlike text modeling, in this last case, we restrict the choice to other values sampled from the same column within the same table. This makes it more challenging for the model and solves the issue of selecting a potentially arbitrary, artificial random text or number.

Training the model entails optimizing the following objective:

$$\theta^* = \arg\min_{\theta} \sum_{i \in \{d,m,y,s,f,e,t\}} \omega_i \mathcal{L}_i(\theta), \tag{1}$$

where $\theta \in \mathbb{R}^k$ denotes the k trainable parameters, $\omega_i \in \mathbb{R}$ are the weighting terms, $\mathcal{L}_i$, for $i \in \{d, m, y, s, f, e\}$ denote the cross-entropy losses for **d**ay, **m**onth, **y**ear, **s**ign, **f**raction, and **e**xponent, and the Huber loss for **t**ext embeddings, respectively. Each of the 7 losses is the result of averaging across all masked tokens of the corresponding type (if any).

Fine-tuning: During fine-tuning, the decoding heads are removed and replaced by a single "pooling" head (e.g., selecting the first token from the transformer output), followed by two linear layers with a non-linearity and dropout in between. For classification, the final layer has as many output nodes as there are classes, trained with cross-entropy loss. For regression, two strategies can be employed. On the one hand, we employ the same shape as the number decoding head above (typically 257 nodes, of which one is used to predict the sign bit, one for $\tilde{\alpha} \in [0, 1]$, and the rest for the exponent in the $[-127, 127] \cap \mathbb{Z}$ range). In this case, the same losses are applied as for pre-training. Alternatively, a single value can be predicted, and the L^2 loss is used. In this last case, it is necessary to normalize the target variable to zero mean and unit variance beforehand.

4 Experiments & Results

4.1 Setup

For the text embeddings, we leverage the SentenceTransformers Reimers and Gurevych (2019) framework and a small open-source LLM for efficiency reasons (`all-MiniLM-L6-v2`)[2], which was trained on 1 billion sentences and produces embeddings $v \in \mathbb{R}^{384}$.

Additionally, we adopt the BERT nomenclature for model sizes (mini, small, medium, base, large), spanning from $N = 4$ layers and $d = 256$ for "mini" to $N = 24$ and $d = 1024$ for "large". Unless otherwise specified, results are reported for the "base" configuration of $N = 12$ layers and $d = 768$. Regarding the weights ω_i, we employ a cascading uniform approach, i.e. $\omega_t = \frac{1}{3}$ and $\omega_i = \frac{1}{9}$ for all other $i \in \{d, m, y, s, f, e\}$.

Our model is pre-trained using tabular data from English Wikipedia, similar to Herzig et al. (2020). This data consists of *infoboxes* (treated as single-row tables) and *wikidata* (multi-row tables found in article texts)—refer to Tab 1 for a breakdown of data statistics.

Table 1: **Data distribution:** Wikipedia pre-training data.

Name	# tables	Avg. rows	Avg. columns	Avg. cells	Total cells
Infoboxes	4.2M	1	8.5	8.5	35M
Wikidata	4.4M	11.8	5.6	73	320M

[2] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

4.2 Training

Pre-training: During pre-training, we randomly sample a single row from each table in the corpus each epoch, thus exposing the model to 8.6M rows per epoch across 30 epochs. This amounts to 1.8B tokens sampled from the overall 355M data points. Remark that not all *wikidata* rows are used in training, though *infobox* rows are reviewed 30 times. We use a triangular learning rate schedule peaking at $3 \cdot 10^{-4}$, starting with a warmup over 5% of the training data, followed by a linear decay to zero. A batch size of 4096 is used, and training is conducted on a single NVIDIA A10 GPU with gradient accumulation, with micro batch sizes ranging from 64 to 256 based on model size.
For validation, 28k *wikidata* tables were randomly selected, with one row each set aside. Validation involves sequentially masking each cell for a prediction task, totaling a validation set size of 155k. Performance is evaluated using cosine similarity for text and absolute difference for numeric and date fields; similarities between predictions and ground truth are compared to similarities between ground truth and unique values within the same column. The final metric is then the average of relative rankings; results are reported in Fig. 2.

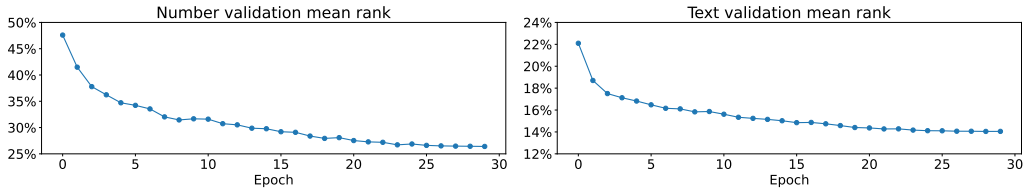

Figure 2: **Epoch-wise pre-training performance:** Validation metrics per epochs for the number and text head during pre-training.

Fine-tuning: For our study, we utilized a compilation of 51 datasets referenced in Kim et al. (2024), consisting of 11 binary classification and 40 regression datasets. Unlike Kim et al. (2024), we consistently split data into 80% training and 20% testing across all datasets, with results detailed in Tab. 2. We also evaluated our model on 45 predominantly numerical datasets from Grinsztajn et al. (2022), with results presented in Tab. 3. These datasets were cleaned without strictly adhering to Grinsztajn et al. (2022) protocols (e.g. multi-class classification was maintained) and limited to $50k$ rows through random sampling, then subject to the same $80\% - 20\%$ data split.

Fine-tuning was conducted for up to 100 epochs with early stopping (patience of 20 epochs), typically concluding between 25 and 70 epochs. Similar to Kim et al. (2024), we experimented with ensemble learning via bagging, using 10 models. For regression, we used here a single head and L^2 loss, predicting the target value, normalized to zero mean and unit variance. Different types of regression targets are analyzed in Section 4.4.

4.3 Results

We benchmarked our models against robust baselines, including tree boosting algorithms (XGBoost Chen and Guestrin (2016), CatBoost Prokhorenkova et al. (2018)) and transformer-based architectures (CM2 Ye et al. (2024), CARTE Kim et al. (2024)). We also compared against feature-engineered versions of boosting algorithms by providing them with text embeddings, reduced to 50 dimensions via PCA. Text columns with fewer than 50 unique values are treated as categorical and retain default encoding methods (one-hot for XGBoost, target encoding for CatBoost). AutoGluon was included as a baseline. Model performance was evaluated using accuracy for classification tasks and R^2 score for regression, capping the latter at zero. Note that our results differ from those in Kim et al. (2024) due to different data splits and baselines. Kim et al. (2024) used smaller datasets (up to 2048 points), different baselines (CatBoost without text embeddings, XGBoost with different text embeddings), and different metrics (AUC and normalized scores).
Analyzing datasets with categorical data from Kim et al. (2024), our model combined with bagging shows state-of-the-art performance in regression and comes close for classification. Results for PORTAL, CARTE, AutoGluon, and CatBoost with embeddings are statistically close, while XGBoost slightly trails. Without bagging, our model outperforms CARTE, CM2, as well as boosting methods which were not enriched via text embeddings.
For numerical datasets from Grinsztajn et al. (2022), both CatBoost and AutoGluon outperform

other models. However, PORTAL is competitive, surpassing XGBoost, CARTE, and CM2. This differentiation highlights our model's adaptability.

Table 2: **Evaluation of regression and classification tasks:** Performance on 51 benchmarks. **Top:** Standard approaches. **Bottom:** Ensemble models with embeddings. **Datasets:** Average scores on 51 datasets from Kim et al. (2024). Training on 80% of the data, testing on 20%.

Method	Acc. (%)	Cl. Rank	R^2	Reg. Rank
CARTE w/o bagging (Kim et al., 2024)	75.2	7.1	67.6	6.8
CatBoost (Prokhorenkova et al., 2018)	75.4	6.2	66.7	6.7
XGBoost (Chen and Guestrin, 2016)	71.8	9.5	59.0	8.7
CM2 (Ye et al., 2024)	76.3	6.6	4.9	10.0
PORTAL w/o bagging	**77.0**	**5.4**	**71.4**	**4.2**
CARTE 10 models bagging	78.3	3.7	72.3	3.2
CatBoost + Embeddings	**78.4**	**2.6**	72.3	3.5
XGBoost + Embeddings	76.5	6.5	67.5	6.8
AutoGluon (Erickson et al., 2020)	**78.4**	3.6	72.6	3.1
PORTAL 10 models bagging	77.8	3.7	**73.8**	**1.9**

Table 3: **Evaluation of regression and classification tasks:** Performance on 45 benchmarks. **Top:** Standard approaches. **Bottom:** Ensemble models with embeddings. **Datasets:** Average scores on 45 numerical datasets Grinsztajn et al. (2022). Training on 80% of the data, testing on 20%.

Method	Acc. (%)	Cl. Rank	R^2	Reg. Rank
CARTE w/o bagging (Kim et al., 2024)	71.9	6.5	67.1	5.7
CatBoost (Prokhorenkova et al., 2018)	**87.8**	**2.2**	**75.5**	**3.2**
XGBoost (Chen and Guestrin, 2016)	84.6	4.7	72.1	4.8
CM2 (Ye et al., 2024)	82.0	6.4	3.2	8.0
PORTAL w/o bagging	84.2	5.1	73.4	5.0
CARTE 10 models bagging	72.6	4.9	68.3	4.3
AutoGluon (Erickson et al., 2020)	**87.5**	**1.7**	**77.2**	**1.8**
PORTAL 10 models bagging	84.5	4.5	75.3	3.1

4.4 Ablation Study

Target number encoding: In Tab. 4, we present the outcomes of various experiments investigating different encoding strategies for the target variable, denoted as $y = \pm\alpha \cdot 2^{\beta}$, with $\tilde{\alpha} = (-1)^{\beta}(\alpha - 1.5) + 0.5$. Each model configuration predicts either y directly or a triplet consisting of the sign, β, and either α or $\tilde{\alpha}$. For direct y prediction, we explore binning into percentiles with cross-entropy loss or using non-binned values with L^2 loss. For triplet predictions, both α and $\tilde{\alpha}$ are tested as binned or non-binned. In non-binned scenarios, $\alpha - 1, \tilde{\alpha} \in [0, 1]$ are learned via binary cross-entropy loss. Furthermore, the results of applying different normalization techniques to y (no normalization, standard scaler, power transform) are investigated.

Bagging: In Fig. 3 (top), we examine the impact of varying the number of top-performing models (in terms of validation metrics) included in bagging on the overall score. Our analysis indicates that averaging results between 5 and 7 models yields the optimal results, increasing overall classification accuracy by $\sim$0.4% and regression R^2 score by $\sim$0.2%.

Note, although adopting this technique would have significantly improved our results, we refrained from including them in Tab. 2, as these ablation results were obtained on the test set directly. However, from an application point of view, it is interesting to note that this approach could maintain, and even surpass, the performance of a full bagging model, while halving the inference runtime.

Model size and pretraining: Finally, we test different model sizes, as well as the effect of pretraining. Results are reported in Figure 3 (bottom). While, on average, pretraining brings a benefit, and larger models do perform better, the relation is noisy. We suspect both effects might become more pronounced with a larger pretraining dataset.

Table 4: **Ablation of regression target encoding**. We report separately the average score of $\max(R^2, 0)$ and the number (out of 40 experiments) of "failures", i.e., when $R^2 < 0$ (or when power transforms resulted in infinity / NaN), to prevent negative scores from impacting the average too much. Numbers refer to fine-tuning a pre-trained "base" size model, without bagging, and with a shorter patience of 10 epochs.

Method	Targets	Binned?	Loss	Normalization	R^2 score (%)	# Failures
PORTAL L^2	y	No	L^2	Standard	**70.9**	0
PORTAL $\tilde{\alpha}$	$\pm, \tilde{\alpha}, \beta$	No	XE	None	67.3	0
Raw L^2	y	No	L^2	None	58.1	0
Percentile	y	Yes	XE	None	63.8	0
Not continuous	$\pm, \alpha - 1, \beta$	No	XE	None	57.8	4
Binned $\tilde{\alpha}$	$\pm, \tilde{\alpha}, \beta$	Yes	XE	None	64.6	1
Continuous L^2	$\pm, \tilde{\alpha}, \beta$	No	L^2	None	63.8	1
Standard $\tilde{\alpha}$	$\pm, \tilde{\alpha}, \beta$	No	XE	Standard	64.1	0
Power L^2	y	No	L^2	Power	67.2	2
Power $\tilde{\alpha}$	$\pm, \tilde{\alpha}, \beta$	No	XE	Power	58.0	4

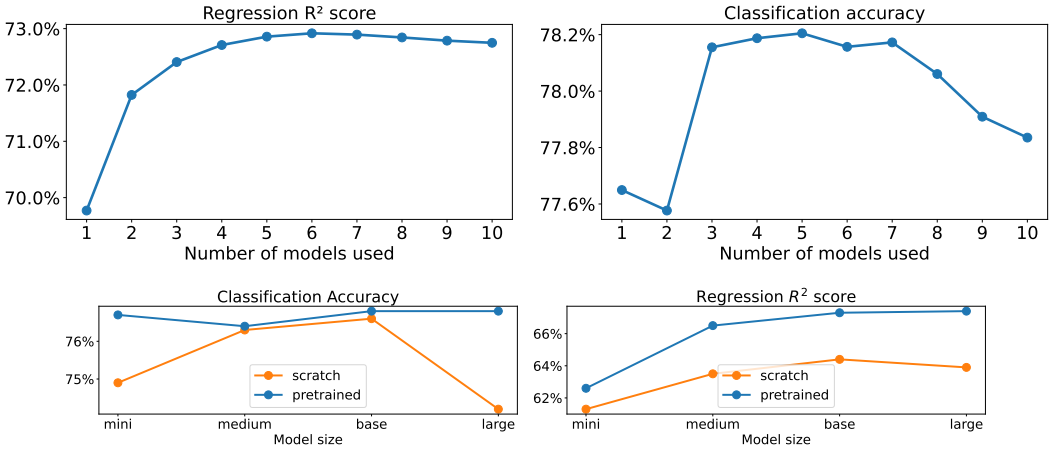

Figure 3: **Performance analysis by model size and count: Top:** Effect of bagging on R^2 score and accuracy: this experiment was conducted by selecting top n performing models (based on validation R^2 scores and accuracy) from a single batch of 10 runs. **Bottom:** Performance of models of different sizes, trained on the full training datasets (using patience = 10 epochs and, for regression, predicting $\tilde{\alpha}$ with binary cross-entropy loss)

5 Conclusion and Limitations

This study introduced a novel transformer encoder-based model, PORTAL, which excels on text-heavy datasets and competes strongly with, and occasionally surpasses, established models like XGBoost, CatBoost, and AutoGluon. The model features an innovative encoding structure and a pre-training scheme using masked cell modeling, removing the need for data normalization and enhancing adaptability during fine-tuning, showing potential benefits over traditional tree-based methods.

However, PORTAL faces challenges, possibly stemming from having used Wikipedia data for pre-training, which may not adequately represent diverse tabular datasets. Incorporating broader datasets such as OpenTabs Ye et al. (2024) or the CSV subset from The Stack Kocetkov et al. (2023) could increase the model's versatility. Additionally, enhancing the text embedding strategy and optimizing the model for numerical data handling could further improve performance. In particular, larger embedding models have shown better performance in various domains (e.g., Grinsztajn et al. (2023)), which is a route we have not yet examined in detail.

Moreover, the computational efficiency of PORTAL lags behind more optimized models. Future work should focus on optimizing pre-training strategies to boost efficiency, perhaps by refining larger models or using optimized ensemble configurations. Addressing these limitations is key to improving PORTAL's effectiveness and real-world applicability, offering promising directions for future research.

References

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*.

Léo Grinsztajn, Edouard Oyallon, Myung Jun Kim, and Gaël Varoquaux. 2023. Vectorizing string entries for data processing on tables: when are larger language models better? *arXiv preprint arXiv:2312.09634*.

Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? In *Advances in Neural Information Processing Systems*, volume 35, pages 507–520. Curran Associates, Inc.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

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

Myung Jun Kim, Leo Grinsztajn, and Gael Varoquaux. 2024. Carte: Pretraining and transfer for tabular learning. In *Forty-first International Conference on Machine Learning*.

Denis Kocetkov, Raymond Li, Loubna Ben allal, Jia LI, Chenghao Mou, Yacine Jernite, Margaret Mitchell, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro Von Werra, and Harm de Vries. 2023. The stack: 3 TB of permissively licensed source code. *Transactions on Machine Learning Research*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. In *NeurIPS*, pages 6639–6649.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.(nips), 2017. *arXiv preprint arXiv:1706.03762*, 10:S0140525X16001837.

Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. 2023. Should you mask 15% in masked language modeling? In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2985–3000, Dubrovnik, Croatia. Association for Computational Linguistics.

Scott Yak, Yihe Dong, Javier Gonzalvo, and Sercan Arik. 2023. Ingestables: Scalable and efficient training of llm-enabled tabular foundation models. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

Jiahuan Yan, Bo Zheng, Hongxia Xu, Yiheng Zhu, Danny Chen, Jimeng Sun, Jian Wu, and Jintai Chen. 2024. Making pre-trained language models great on tabular prediction. In *The Twelfth International Conference on Learning Representations*.

Yazheng Yang, Yuqi Wang, Guang Liu, Ledell Wu, and Qi Liu. 2024. Unitabe: A universal pretraining protocol for tabular foundation model in data science. In *The Twelfth International Conference on Learning Representations*.

Chao Ye, Guoshan Lu, Haobo Wang, Liyao Li, Sai Wu, Gang Chen, and Junbo Zhao. 2024. Towards cross-table masked pretraining for web data mining. In *Proceedings of the ACM on Web Conference 2024*, pages 4449–4459.

Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. 2023. Xtab: cross-table pretraining for tabular transformers. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.