Machine Learning (2025) 114:145
https://doi.org/10.1007/s10994-025-06778-1

One transformer for all time series: representing and training with time-dependent heterogeneous tabular data

Simone Luetto[1] · Fabrizio Garuti[2,3] · Enver Sangineto[3] · Lorenzo Forni[2,4] · Rita Cucchiara[3,5]

Received: 12 July 2023 / Revised: 21 March 2025 / Accepted: 30 March 2025 /
Published online: 28 April 2025

Abstract
There is a recent growing interest in applying Deep Learning techniques to tabular data in order to replicate the success of other Artificial Intelligence areas in this structured domain. Particularly interesting is the case in which tabular data have a time dependence, such as, for instance, financial transactions. However, the heterogeneity of the tabular values, in which categorical elements are mixed with numerical features, makes this adaptation difficult. In this paper we propose UniTTab, a Transformer based architecture whose goal is to uniformly represent heterogeneous time-dependent tabular data, in which both numerical and categorical features are described using continuous embedding vectors. Moreover, differently from common approaches, which use a combination of different loss functions for training with both numerical and categorical targets, UniTTab is uniformly trained with a unique Masked Token pretext task. Finally, UniTTab can also represent time series in which the individual row components have a variable internal structure with a variable number of fields, which is a common situation in many application domains, such as in real world transactional data. Using extensive experiments with five datasets of variable size and complexity, we empirically show that UniTTab consistently and significantly improves the prediction accuracy over several downstream tasks and with respect to both Deep Learning and more standard Machine Learning approaches. Our code and our models are available at: https://github.com/fabriziogaruti/UniTTab.

Keywords Tabular data · Time series · Heterogeneous structural data · Deep learning for finance

Editor: Gustavo Batista.

Simone Luetto and Fabrizio Garuti have contributed equally to this work.

Extended author information available on the last page of the article

1 Introduction

Despite the success of Deep Learning methods in different areas of Artificial Intelligence (AI), such as, for instance, Natural Language Processing, Computer Vision, Audio Processing, Robotics, etc., the use of deep networks to represent tabular data is so far largely underexplored. However, tabular data have a large application interest, since many public institutions or commercial/industrial companies represent their structured knowledge using datasets of "tables" (Kotios et al., 2022). For instance, bank data, clinical data, commercial data, etc., are often provided as a list of attributes (field names) and corresponding values (field values) for each represented entity (sample). As reported in Benjelloun et al. (2020), over 65% of the datasets in the Google Dataset Search platform contain tabular files in either CSV or XLS formats. Particularly interesting is the case of financial transactions, which, for instance, describe the sequence of (time dependent) transactions of a given bank client on their bank account. However, most of the Machine Learning approaches for predictive tasks in these scenarios are often still based on pre-Deep Learning techniques, such as Gradient Boosted Decision Trees (Ye et al., 2024; Borisov et al., 2023; Chen et al., 2016; Kotios et al., 2022).

One of the reasons why Deep Learning is still underexplored with tabular data is the lack of large scale publicly available datasets, often due to commercial or privacy constraints (e.g., in the financial domain). However, another important reason is the heterogeneity of these data, which are different from, e.g., texts or images and make their uniform representation and training challenging. In fact, while an image is composed of a set of (numerical) pixel values and a textual sentence is a list of (categorical) words, tabular data are usually composed of heterogeneous attributes, in which numerical features (e.g., the transaction amount) are mixed with categorical features (e.g., the category of the transaction receiver or the transaction type). Thus, it is not clear how different types of features should be jointly represented, and what kind of objective function should be optimized during training to cover all the data types. Moreover, a second source of heterogeneity comes from the necessity to represent time sequences composed of rows with a variable internal structure and a variable number of fields. For instance, a "Point Of Sale" (POS) transaction includes a field describing the locality of the payment, which is not included in other transaction types made from *the same* bank account and included in *the same* time series (e.g., an ATM transaction).

In this paper, we propose a unified architecture based on modern Transformers (Vaswani et al., 2017) to simultaneously and uniformly deal with all these representation problems. Specifically, inspired from TabBERT (Padhi et al., 2021), we use a hierarchical architecture to represent the time series dynamics. The hierarchy is composed of two levels, the first of which represents a single tabular row (e.g., a specific transaction), while the second represents a sequence of temporally dependent rows. This two-level hierarchy is a common solutions in, e.g., video processing, where each single frame is independently embedded using a still image Transformer and the sequence of frames is then passed to a second Transformer (Arnab et al., 2021). We empirically show that this solution makes it possible to represent time series of more than one hundred rows. However, differently from TabBERT and other similar hierarchical approaches, we introduce the following methodological novelties.

First, we modify the hierarchical network proposed in Padhi et al. (2021) to uniformly and efficiently represent tabular rows with different *types*. Taking the financial transaction case as an example, a POS transaction corresponds to a table row with a set of fields different from an

ATM transaction. We introduce a type-dependent embedding interface between the two levels of the hierarchy which projects each row type into a fixed-dimension vector fed to the second level. This interface extends the common look-up table of initial token embeddings used in Transformer networks (Vaswani et al., 2017) and it addresses a problem not yet considered in literature, i.e., how to deal with time series composed of structurally different types of consecutive rows.

Second, inspired by the numerical representation adopted, for instance, in NeRFs (Mildenhall et al., 2020) for 3D synthesis, we propose to represent each numerical value as a feature vector obtained using a set of sinusoidal functions at different frequencies. The motivation for this choice arises from the observation that deep networks are biased towards learning low frequency functions (Rahaman et al., 2019), while the scalar value of a specific tabular field undergoes to a high frequency variation. Thus, similarly to the coordinate embedding adopted in NeRFs, we smooth this variation using a set of sinusoidal transformations defined over a range of different frequencies. Note that this encoding is used to represent a specific *field value* and not to represent the *field position* as with the standard Positional Encoding used in Transformers (Vaswani et al., 2017).

Third, we adopt a unified objective function for all the feature types based on the standard Masked Token pretext task (Devlin et al., 2019), which avoids the need to tune loss specific weights. However, since, in our framework, the numerical features are *not* represented as discrete tokens, in order to use a Masked Token task for a numerical input, we propose to decouple the numerical feature representation from the target prediction used during the self-supervised training of the network. Specifically, inspired by BEiT (Bao et al., 2022), where a discrete VAE (Ramesh et al., 2021) is used to quantize image patches and extract a discrete token for each of them, we similarly quantize the numerical features using a predefined set of bins, and then we use this quantization as the target token. Note that the discretized value is the *target* label associated with a numerical value, but it is *not* used as input to the network.

Finally, we use a standard Label Smoothing (Szegedy et al., 2016) for the categorical feature tokens and we propose Neighborhood Label Smoothing for the numerical feature targets. The latter is based on the observation that numerical feature values, once discretized, still preserve a total order relation over the elements of the quantized vocabulary, thus, differently from pure categorical features, a small neighborhood of the original (numerical) value can be computed and used to restrict Label Smoothing on the most informative value range.

We call our network Unified Transformer for Time-Dependent Heterogeneous Tabular Data (UniTTab), and we show that it consistently outperforms state-of-the-art approaches based on both Deep Learning and more standard Machine Learning techniques for time series of tabular data, often with a large margin. In our experiments, we use different common benchmarks and we additionally use a private dataset we collected in collaboration with a private financial institute. It is a dataset composed of millions of real bank account transactions spanning over 2 years, and it represents a real life scenario, in which the transaction history of a given account is highly variable and different transactions of the same time series can have a different internal row structure.

2 Related work

Recent different works focus on using deep networks for tabular data (Ye et al., 2024) and time series (Trirat et al., 2024). For instance, (Lyu et al., 2022) propose an architecture which combines different modules (meta-embeddings, automatic discretization and

aggregation) and can represent both numerical and categorical features. (Borisov et al., 2023) use a distillation approach to map decision trees, trained on heterogeneous tabular data, onto homogeneous vectors which are fed to a deep network. Shankaranarayana et al. (2021) represent the time stamp as a sequence of different fields (e.g., the year, the month, etc.), each of which separately embedded and then summed together. We follow a similar paradigm when representing time, but we do not sum the corresponding embeddings. Huang et al. (2020) represent categorical features using an attribute specific embedding which is used as a prefix, concatenated with the actual field value. Schäfl et al. (2022) use a non-parametric representation of the training data, which reminds the use of external networks in Transformers (Wu et al., 2022). Beyazit et al. (2023) represent a dataset of *numerical* tabular data using the Generalized Fourier Transform. Zhang et al. (2024) and Suh et al. (2024) use a Variational AutoEncoder (Kingma et al., 2014) to represent heterogeneous tabular data in a latent space and then they adopt a Diffusion Model (Ho et al., 2020) for data generation. Other recent works use Transformer architectures to represent tabular data (Kossen et al., 2021; Gorishniy et al., 2021; Huang et al., 2020; Somepalli et al., 2021). However, these approaches still underperform standard, non Deep Learning based methods (Huang et al., 2020; Gorishniy et al., 2021), such as Gradient Boosted Decision Trees (Chen et al., 2016). Moreover, they usually represent numerical/categorical features in a relatively simple way, e.g., using a linear embedding layer or a quantization method, which we show in Sect. 5.1 to be suboptimal. Finally, most of previous work dealing with tabular data does not model the temporal *dynamics*: each row in the table is an individual sample. Conversely, in this paper we focus on the more general case in which the rows are concatenated into a sequence (a time series) representing the temporal evolution of the data, similarly to frames in a video.

Another interesting line of work is based on directly or indirectly using natural language-based "interfaces" between the tabular data and a Transformer. For instance, in LUNA (Han et al., 2022), numerical values are represented as an atomic natural language string. Similarly, (Solatorio et al., 2023) represent numerical features using textual tokens. Other works create a bridge between tabular data and Large Language Models (LLMs) and use a zero/few-shot learning paradigm based on textual prompts. For instance, (Li et al., 2024) use an LLM-based encoder initialized with the pre-trained weights of the textual encoder of CLIP (Radford et al., 2021) to represent *numerical* time series of different domains. Narayan et al. (2022) transform numerical and categorical field values and attribute names in a natural-language prompt, and then they adopt a pre-trained GPT-3 language model (Brown et al., 2020) as a few/zero shot learner. However, the type of data used is relatively simple and should be described as attribute-value pairs (without any temporal dynamics). In the context of synthetic tabular data generation, a similar strategy is adopted in Borisov et al. (2022), where syntactically correct natural language sentences are created using the feature names and the row values. Similarly, (Jiang et al., 2023) develop a set of interfaces which are used to query a tabular dataset. The retrieved column names and values are then transformed in a textual prompt which is fed to a pre-trained LLM. However, apart from the lack of representation of temporal dynamics, there are other important problems in using a prompt to represent tabular data in natural language for an LLM. The first is that the LLM maximum context length limits the number of training samples which can be fed to the LLM. Thus, when the training dataset is composed of, e.g., thousands or millions of rows, training a specialized Transformer usually leads to a performance largely superior to an in-context learning paradigm (Narayan et al., 2022). The second is that, as observed by Narayan et al. (2022), tabular attributes and field values frequently contain domain-specific abbreviations or jargon terms not commonly found in textual documents, and, in these

cases, an LLM lacks a strong semantic understanding of the input because these data are out of its training distribution.

In this work, we train a specialized Transformer on large datasets of tabular data, and we use it to represent both the data heterogeneity and the dynamics of the time series (Wen et al., 2022). Specifically, the work which is the closest to our proposal is TabBERT (Padhi et al., 2021), in which the authors propose a hierarchical network to efficiently represent tabular time series (Sects. 1 and 3). However, our proposal differs from TabBERT in different aspects, the main of which are: (1) While in TabBERT numerical features are discretized in bins *when input to the network*, we propose a frequency-based embedding representation; (2) When training the network, we decouple the input and the output representations similarly to BEiT (Bao et al., 2022), and we propose a Neighborhood-based Label Smoothing; (3) Our network architecture can represent time series in which each row is an ensemble of possibly different attributes, in this way dealing with real life complex datasets. In our experiments, we use TabBERT as the main baseline and we show that UniTTab significantly outperforms TabBERT, as well as all the other tested state-of-the-art models, in all the tested scenarios.

3 Preliminaries

Problem Statement. Tabular data are represented as a set of attributes (field names) $A = \{a_1, \ldots, a_k\}$, where each $a_j \in A$ can be either categorical or numerical, and a set of table rows $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N$, which specify a value for each field: $\boldsymbol{r}_i = [v_1, \ldots, v_k]$. If a_j is numerical, then $v_j \in \mathbb{R}$, otherwise $v_j \in V_j$, where V_j is an unordered attribute-specific vocabulary of categories. A time series is a (variable length) sequence of rows $\boldsymbol{s} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t]$ which are related to each other by a temporal dynamics. For instance, in financial transactional data, $\boldsymbol{s}$ can represent the last t bank account transactions of a given client (see Fig. 1). When adopting a Deep Learning method, a common paradigm is to use a large dataset of time series to pre-train a network with self-supervised learning, and then fine-tune the model for a specific downstream task using task-specific labeled data and a possible smaller (supervised) dataset.

In this paper, we further generalize the previous scenario introducing time series composed of different row types. As mentioned in Sect. 1, this generalization is particularly useful in real life datasets, in which, for instance, a transaction time series is composed of different transaction types (e.g., POS type, ATM type, etc.). Formally, we describe this situation using a function which associates each row in $\boldsymbol{s}$ to a predefined set of row types: $type(\boldsymbol{r}_i) = h \in T = \{1, \ldots, n\}$ and using a type-dependent set of attributes

Card	Timestamp	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?
4	06/10/2013 04:45	$31.32	Online Transaction	Frontier Communications	ONLINE			4784	Bad CVV,
4	11/10/2013 10:13	$209.43	Swipe Transaction	Applebees	Strasburg	OH	44680	5211	
4	11/10/2013 23:31	$1.72	Swipe Transaction	Chevron	West Covina	CA	91792	5499	Technical Glitch,
4	12/10/2013 04:03	$49.38	Online Transaction	Frontier Communications	ONLINE			5300	
4	12/10/2013 04:28	$244.67	Online Transaction	Frontier Communications	ONLINE			5816	
4	12/10/2013 06:52	$12.29	Swipe Transaction	Barnes & Noble	Spring Valley	CA	91977	5942	
4	13/10/2013 07:53	$5.32	Swipe Transaction	Anwar Grocery	Alhambra	CA	91801	5411	
4	13/10/2013 15:56	$40.00	Swipe Transaction	Green Wholesale	Chula Vista	CA	91910	4829	
4	14/10/2013 04:40	$41.56	Online Transaction	Frontier Communications	ONLINE			4784	Bad CVV,
4	14/10/2013 04:48	$53.31	Online Transaction	Frontier Communications	ONLINE			4784	

Fig. 1 An example of time series taken from the Transaction Dataset (Padhi et al., 2021). Each row is a bank transaction and it is composed of $k = 10$ attributes. This sequence of $t = 10$ temporally consecutive transactions (rows) of the same client is a time series

$A_h = \{a_1, \ldots, a_{k_h}\}$ to specify the fields of $\boldsymbol{r}_i$. Note that the cardinality of the attributes (k_h) varies depending on h. Moreover, we assume that $type(\boldsymbol{r}_i)$ is always defined for each $\boldsymbol{r}_i \in \boldsymbol{s}$: for example, each transaction in a time series of a bank account can be of only one type (e.g., POS, or ATM, etc.).

TabBERT architecture. TabBERT is a hierarchical architecture composed of two different Transformers, trained end-to-end (Fig. 2a). The first Transformer (called "Field Transformer") takes as input the k field values of a single table row $\boldsymbol{r}_i$. Note that k is constant for all the rows, as TabBERT implicitly assumes that there is only one row type (in our notation: $|T| = n = 1$). Numerical features are discretized using an attribute specific set of bins. In this way, both numerical and categorical features can be associated to a specific discrete token. The tokens are transformed in embedding vectors using a standard look-up table of learned embeddings (Vaswani et al., 2017). In Fig. 2a, this is indicated as a set of field embeddings $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_k$. The Field Transformer transforms these vectors in k final embeddings of dimension d, which are concatened in a single vector $\boldsymbol{g}$ of dimensions $d \cdot k$. Then, $\boldsymbol{g}$ is fed to the second Transformer ("Sequence Transformer"), jointly with the representations of all the other rows in the input time series $\boldsymbol{s}$. Note that the dimension of each $\boldsymbol{g}$ should be constant because $\boldsymbol{g}$ is a fixed-size initial embedding vector for the second Transformer. The Sequence Transformer outputs a sequence of t final embedding vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t$. Finally, each $\boldsymbol{z}_i$ is split in k vectors, on top of which a shallow MLP is used to output a posterior distribution over the attribute-specific vocabulary. This makes it possible to apply a Masked Token pretext task (Devlin et al., 2019) during pre-training, in which a few tokens are randomly masked and the network is asked to predict the masked tokens.

4 Method

In this section we present UniTTab, showing the architecture of the network, the way in which heterogeneous features are represented and the uniform pre-training strategy.

Row-type dependent embedding. We first extend the hierarchical architecture of TabBERT to deal with a variable number of row types ($n > 1$). The main problem we need to solve is that k_h depends on $type(\boldsymbol{r}_i)$ for each $\boldsymbol{r}_i \in \boldsymbol{s}$, while the dimension of $\boldsymbol{g}$ should be fixed (Sect. 3). We solve this problem using a linear projection layer (Fig. 2b) which

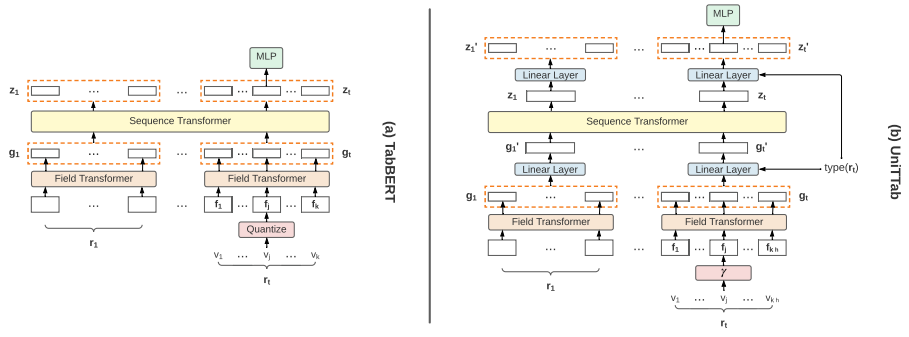

Fig. 2 A schematic comparison between the architectures of TabBERT (**a**) and UniTTab (**b**). In both figures, v_j is a numerical value. Note that in **b** the number of attributes of each row (k_h) is variable

takes $h = type(\boldsymbol{r}_i)$ as input and transforms $\boldsymbol{g} \in \mathbb{R}^{d \cdot k_h}$ in $\boldsymbol{g}' \in \mathbb{R}^m$, where m is fixed and the transformation depends on a row-type specific linear matrix W_h:

$$\boldsymbol{g}' = W_h \boldsymbol{g} \quad (W_h \in \mathbb{R}^{m \times (d \cdot k_h)}). \tag{1}$$

The set of learnable projection matrices $W_1, \ldots, W_n$, one for each row type, constitutes a look-up table of embeddings for the initial layer of the second Transformer, and naturally extends the common initial embedding look-up table used in Transformer networks. Analogously, each final row embedding $\boldsymbol{z}$ (Sect. 3) is transformed in $\boldsymbol{z}' \in \mathbb{R}^{d \cdot k_h}$ using a specific weight matrix S_h ($S_h \in \mathbb{R}^{(d \cdot k_h) \times m}$) before being fed to the final MLP (Fig. 2b).

Feature representation. We represent each categorical feature using a standard linear embedding based on its attribute-specific vocabulary. However, for numerical features, we extract a frequency-based representation as follows. Let v be a scalar value corresponding to a numerical attribute. Similarly to (Mildenhall et al., 2020), we transform v using:

$$\gamma(v) = (\sin(2^0 \pi v), \cos(2^0 \pi v), \ldots, \sin(2^{L-1} \pi v), \cos(2^{L-1} \pi v)), \tag{2}$$

where $L = 8$ is the number of sine/cosine pairs used (see Sect. 5.1). The vector $\gamma(v)$ so obtained is then fed to a linear (learnable) layer whose output is the initial embedding vector for v. Finally, similarly to (Shankaranarayana et al., 2021), we represent the numerical value of a time stamp attribute (e.g, measured in minutes) using a combination of different time fields (i.e, the year, the month, the day, and, if necessary, the hour). Each such basic value is represented as a categorical feature (e.g., with 12 elements for the month, etc.). In preliminary experiments, we also tried to represent each basic time field of a date as a numerical attribute (i.e. by means of our frequency-based representation), but this led to slightly worse results, presumably because the periodicity of some time series (e.g., in transactional data) can better be represented by the network as a categorical value. Note, that, differently from (Shankaranarayana et al., 2021), we do not add the corresponding field embeddings but treat them as additional fields simply by increasing the number of attributes of each single row.

Training. During the unsupervised pre-train stage, we train UniTTab using *only* a Masked Token pretext task. Specifically, for an input sample $\boldsymbol{s} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_t]$, we randomly replace a field value $v \in \boldsymbol{r}_i$ with the special symbol `[MASK]`. We use a standard replacement probability value $p_f = 0.15$ (Devlin et al., 2019; Padhi et al., 2021). Moreover, with probability $p_r = 0.1$, we also mask *all* the values in a row $\boldsymbol{r}_i$, while, for the fields representing the time stamp, they are always either jointly masked or jointly unmasked. We call these additional masking strategies "Row masking" and "Time stamp masking", inspired by the block masking of adjacent image patches used in BEiT (Bao et al., 2022), and we use them to make the pretext task more challenging for the network.

Note that we can replace (the initial embedding vector of) v with `[MASK]` independently on whether v is numerical or categorical. However, the problem is what the network should predict in correspondence of v if this is not an element of a discrete vocabulary. A possible solution could be to adopt a regression loss function and directly ask to the network to reconstruct the original numerical value v. The disadvantage of this hybrid solution is that we would need two different loss functions: one for the categorical features (e.g., the Cross Entropy), and another for the numerical values (e.g., an MSE loss), which then need to be suitably weighted. Conversely, we propose a different solution: inspired by BEiT, we quantize v and we use its categorical representation as the target label. Specifically, if a_j is a numerical attribute, we define a vocabulary of bin values $V_j = \{b_1, \ldots, b_q\}$ spanning the whole range of possible values for a_j. Then, when a given value v for this attribute is

masked, we quantize v ($b = quantize(v)$) and we use $b \in V_j$ as a pseudo label for v, which is used as the token to be predicted. Note that b is *not* input to the network.

When we compute the loss function, we use Label Smoothing (Szegedy et al., 2016), which replaces the one-hot vector representation of a categorical label $v \in V_j$ with a vector of smoothed probability values $\boldsymbol{p}(v)$. In more detail, if $V_j = \{v_1, \ldots, v_{q_j}\}$ is the vocabulary corresponding to a_j, and q_j is its cardinality, then a ground truth value $v \in V_j$ is associated with the vector $\boldsymbol{p}(v)$ defined as follows:

$$[\boldsymbol{p}(v)]_l = \begin{cases} 1-\epsilon, & \text{if } l = v \\ \frac{\epsilon}{q_j - 1} & \text{otherwise,} \end{cases} \tag{3}$$

where $[\boldsymbol{p}(v)]_l$ is the l-th element of $\boldsymbol{p}(v)$ and ϵ a small constant. Furthermore, in case a_j is a numerical attribute, we propose Neighborhood Label Smoothing, in which $b = quantize(v)$ is smoothed using only a small neighborhood centered in b. Specifically, we use the range $R = \{b-5, \ldots, b+5\}$, and we replace Eq. 3 with:

$$[\boldsymbol{p}(v)]_l = \begin{cases} 1-\epsilon, & \text{if } l = b \\ \frac{\epsilon}{10} & \text{if } l \in R, l \neq b \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Note that this is possible because, if a_j is numerical, then V_j, obtained with quantization, is an ordered set.

Finally, if $\boldsymbol{s}'$ is the perturbed version of $\boldsymbol{s}$, in which some random field values have been masked as explained above, then our Masked Token pretext task can be formulated as minimizing the following Cross Entropy loss:

$$\min_{\theta} - \sum_{v \in \boldsymbol{s}} \mathbb{1}_{Masked(v)} \boldsymbol{p}(v) \log p_\theta(quantize(v) | \boldsymbol{s}'), \tag{5}$$

where: θ are the parameters of the network, $p_\theta(y|\boldsymbol{x})$ is the probability of the network to predict y given the sequence $\boldsymbol{x}$ as input, $\mathbb{1}_{Masked(v)}$ is 1 only when v was masked (otherwise is 0), $quantize(v) = v$ if v is categorical, and, with a slight abuse of notation, $v \in \boldsymbol{s}$ indicates a generic field value of one of the rows in $\boldsymbol{s}$.

5 Experiments

In this section, we evaluate UniTTab using different datasets and downstream tasks. We use the same architecture for all the datasets, with the only difference being the number of fields (k) and the length (t) of the time series, which depend on the specific dataset and the adopted evaluation protocol. Specifically, independently of the dataset, we always use only one self-attention layer with 8 heads in the Field Transformer, and 12 self-attention layers with 12 heads each in the Sequence Transformer. For a fair comparison, the total number of parameters of UniTTab was kept approximately the same as in TabBERT (see Sect. C for more details). We use five different-size datasets of time series of heterogeneous tabular data: the Pollution Dataset (Liang et al., 2015), the Transaction Dataset (Padhi et al., 2021) (both adopted by TabBERT), the PKDD'99 Financial Dataset (Berka, 1999), the Age2 dataset (Fursov et al., 2021), and our Real Bank Account Transaction Dataset (in short,

RBAT Dataset). In the latter, each time series is composed of three different row types (i.e., $n = |T| = 3$, see Sect. 3). The other four datasets have only one row type ($n = |T| = 1$), thus, in all the experiments but those on the RBAT Dataset, we use only one projection matrix in Eq. 1. All the five datasets are composed of tabular data with both numerical and categorical fields (see Sect. B for more details). In all the experiments, the Deep Learning based models are first pre-trained using self-supervision (Sect. 4) and then evaluated using a dataset specific (supervised) downstream task, while standard Machine Learning methods are directly trained on the downstream task training set. Moreover, to enable a fair comparison with other methods tested on the Pollution Dataset and the Transaction Dataset, we follow the protocol defined in Padhi et al. (2021) for the downstream task definition and the training/testing partitions. However, since the stride used in Padhi et al. (2021) to extract the time series from the Pollution Dataset leads to a partial information leak between the training and the testing data of the downstream task, we additionally use a different training/testing partition for this dataset, which we call "our partition", while we use the term "original partition" to refer to the initial split adopted in Padhi et al. (2021) and in other works (see Sect. A for more details). Finally, for the other three datasets, we define the details of the corresponding downstream tasks and the training/testing partitions in Sect. A.

5.1 Ablation study

In this section, we analyze the contribution of each component of our method. We use the Pollution Dataset (Liang et al., 2015) and a random subset of the Transaction Dataset (Padhi et al., 2021) (400 K time series, see Sect. A for more details). The former dataset is composed of time series where each row contains $k = 11$ fields, 10 of which are numerical attributes and one is categorical (Sect. B). Hence, this dataset is particularly suitable for investigating the influence of different numerical feature representations (Sect. 4). The downstream task is a regression task, and the performance is measured using RMSE (the lower the better). In the Transaction Dataset, the rows of the time series are composed of 2 numerical and 8 categorical attributes, and its associated downstream task is a binary classification task. In this case, we use the F1 score as the performance metric (the higher the better). Both datasets contain a time stamp attribute, which we represent using 4 time attributes as explained in Sect. 4. For both datasets, following (Padhi et al., 2021), after self-supervised pre-training, the final embeddings of the Sequence Transformer $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t)$ are used as input to an LSTM which is (separately) trained to solve a supervised regression or binary classification task using the corresponding task labels. Specifically, we first train UniTTab using Eq. 5 and then we train an LSTM (using the same LSTM architecture adopted in Padhi et al. (2021)) on top of the final embeddings of our Sequence Transformer. In Sect. 5.2 we also show the results obtained by directly fine-tuning our model, without using an LSTM, as well as comparative results obtained using the full Transaction Dataset and both the original and our partition of the Pollution Dataset.

Table 1 analyses the impact of the main components of our method. The first row is the baseline, which corresponds to our implementation of TabBERT, starting from its publicly available code and keeping fixed all the main hyperparameters (e.g., the number of layers of the two Transformers, the number of heads, the emdedding size, etc.). In this baseline, the numerical features are quantized into discrete bins and treated as categorical *when input to the network*. Note that there is no Row or Time stamp masking nor Label Smoothing. The the Cross Entropy loss is the only loss used because the tokens of the quantized numerical

features can be masked and predicted just like categorical tokens. The second row of the table shows the improvement obtained when the time stamp is split in 4 different fields, which corresponds to a significant performance improvement in both datasets and tasks. In the third row, we replace the discrete representation of the numerical features with our frequency-based representation (Sect. 4). In this case, for each masked numerical field value v, we use a regression function, which consists in predicting the original scalar value of v (before the frequency-based embedding). We use the squared difference between the predicted and the ground truth value as the loss function (MSE loss) for these numerical features, which is summed to the Cross Entropy loss computed with the categorical features. Note that, as mentioned in Sect. 4, one of the problems when using two different loss functions is the necessity to weight their relative importance. In this ablation experiment, we set the MSE loss weight using a simple heuristic in which we compute the average MSE loss value and the average Cross Entropy loss value and we impose these two values to be equivalent using a relative weight (more details in Sect. C). Comparing this row with the baseline (first row), the improvement obtained when using a frequency-based numerical representation jointly with a regression loss is comparable with the introduction of the combinatorial time stamp. In row 4, the regression loss is replaced by the Cross Entropy loss, which is applied to numerical features using pseudo target labels (see Eq. 5). Comparing this row with the baseline, we obtain the largest relative improvement, which shows that using frequency-based numerical representations jointly with pseudo labels for the Cross Entropy can significantly boost the network performance. In row 5, the combinatorial time stamp is added to the setting of row 4, bringing an additional benefit. Finally, row 6 and row 7 show the results corresponding to the introduction of standard Label Smoothing and Neighborhood Label Smoothing, respectively.

In Table 2, we analyze the impact of using Row masking and Time stamp masking, where the first strategy is evaluated with two different selection probability values p_r. The best result corresponds to using both Row masking (with probability $p_r = 0.1$) and Time stamp masking. In Table 3 we empirically evaluate the influence of the number of frequency function pairs L (Sect. 4) used for our frequency-based numerical feature representation. In the rest of this paper, we use the best value ($L = 8$) for all the other datasets and tasks.

Finally, we postpone the ablation of the Row-type dependent embedding (Sect. 4) to Sect. 5.2, where we introduce our RBAT Dataset composed of variable row types. In the same section, we sow the impact of different amount of pre-training data on the downstream task performance.

5.2 Main results

In this section, we compare UniTTab with different state-of-the-art approaches using different datasets and downstream tasks.

Pollution prediction task. This is the regression task used in Sect. 5.1 (and presented with more details in Sect. A). We show results based on two main paradigms: based on LSTM training and on directly fine-tuning the pre-trained model. The former is based on the protocol proposed by Padhi et al. (2021), and trains a separate LSTM using the vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_t$ as input and the labels of the downstream task as the supervision (see Sect. 5.1). However, the latter, proposed here, is likely a much more natural choice, and it is coherent with most of the recent AI literature, where the backbone network, after pre-training, is directly fine-tuned for a specific downstream task without training a separate network. Specifically, when we fine-tune either UniTTab or TabBERT, we include a `[CLS]` token in the input sequence of the Sequence Transformer and we use the corresponding $\boldsymbol{z}_{\texttt{[MASK]}}$ final

Table 1 Ablation study using the Pollution Dataset (our partition) and a subset of the Transaction Dataset

	Quantized numerical features	Combina. time stamp	Frequency based num. features	Regression loss	CE loss (only)	Standard Label Smoothing	Neighbor Label Smoothing	Pollution Dataset (RMSE ↓)	Transaction Dataset (F1-score ↑)
1	✓				✓			34.30	0.829
2	✓	✓			✓			32.23	0.840
3			✓	✓				32.47	0.844
4			✓		✓			31.52	0.846
5		✓	✓		✓			29.63	0.847
6		✓	✓		✓	✓		29.47	0.848
7		✓	✓		✓	✓	✓	**29.05**	**0.850**

Bold indicates best results

Table 2 An analysis of different masking strategies using the Pollution Dataset (our partition) and a subset of the Transaction Dataset

	Row masking $p_r = 0.05$	Row masking $p_r = 0.1$	Time stamp masking	Pollution Dataset (RMSE ↓)	Transaction Dataset (F1-score ↑)
1				29.05	0.850
2	✓			29.01	0.851
3		✓		28.91	0.854
4			✓	28.82	0.855
5		✓	✓	**27.99**	**0.858**

Bold indicates best results

Table 3 The influence of the number of frequency function pairs (L) using the Pollution Dataset (our partition) and a subset of the Transaction Dataset

L	Pollution Dataset (RMSE ↓)	Transaction Dataset (F1-score ↑)
4	28.63	0.828
6	28.38	0.830
8	**27.99**	**0.858**
10	28.11	0.808
12	28.20	0.804

Bold indicates best results

embedding vector as input to a final linear layer dedicated to the specific task (and trained from scratch). In case of LUNA (Han et al., 2022), we report the results we obtained using its public code and only the LSTM training paradigm (adopted also in Han et al. (2022)), because fine-tuning this method requires non-trivial modifications of its architecture.

To enable a comparison with results published in previous work, in the experiments of Table 4 we strictly follow (Padhi et al., 2021) and we use the original partition of the Pollution Dataset (see Sect. 5 and Sect. A) jointly with a time series length $t = 10$. Conversely, in Table 5 we use our partition (where $t = 10$ but also the time series extraction stride is 10, see Sect. A for more details) and we repeat all the experiments 5 times with different random seeds. Note that the random seed is used both to initialize the models' parameters and to randomly split the time series in training and testing splits. Finally, in Table 6 we extend these experiments to longer time series with $t = 50$ (Sect. A). In all cases (Tables 4, 5 and 6), UniTTab outperforms both TabBERT and LUNA by a large margin, and, as expected, direct fine-tuning significantly improves over the separate LSTM training.

In the same tables, we also report the results obtained using both XGBoost (Chen et al., 2016) and CatBoost (Prokhorenkova et al., 2018), which are the state-of-the-art non Deep Learning based methods for tabular data (Sect. 1 and 2). Specifically, since both XGBoost and CatBoost cannot directly work on (variable-length) time series, we used a standard library (Christ et al., 2018) to extract features from a time series. These "engineered" features include field-specific statistics (e.g., the mean or the variance for numerical features and the mode for the categorical ones, etc.), autocorrelation with

Table 4 Pollution prediction task (original partition)

Downstream task training	Model	RMSE
Fine-tuning	TabBERT†	24.10
	UniTTab (ours)	**20.05**
LSTM	TabBERT (Padhi et al., 2021)‡	32.80
	LUNA†	37.73
	UniTTab (ours)	**25.42**
Training from scratch	XGBoost†	34.14
	CatBoost†	34.15
	VAR†	83.83

Bold indicates best results

† Our reproduction. ‡ Results reported in the corresponding paper

Table 5 Pollution prediction task (our partition): average and standard deviation results obtained with 5 random seeds

Downstream task training	Model	RMSE
Fine-tuning	TabBERT†	31.41 (± 1.74)
	UniTTab (ours)	**25.37** (± 1.59)
LSTM	TabBERT†	37.20 (± 1.67)
	LUNA†	45.17 (± 0.62)
	UniTTab (ours)	**30.88** (± 1.70)
Training from scratch	XGBoost†	48.89 (± 0.73)
	CatBoost†	47.09 (± 0.89)
	VAR†	84.05 (± 1.71)

Bold indicates best results

† Our reproduction using the official code

different "lags", the number of local minima and maxima, etc. Moreover, the features are automatically pre-selected using a Filter feature selection method on a validation set extracted and separated from the training data. Finally, we used grid search on the validation set to set the optimal values of the XGBoost and the CatBoost hyperparameters (e.g., the number of trees, the max depth of each tree, etc.). After feature selection and hyperparameter tuning, the validation set is merged with the training set and XGBoost/CatBoost are re-trained using the same data adopted for training the LSTMs or fine-tuning the UniTTab/TabBERT models on the supervised downstream task. We will use this procedure (task and dataset dependent feature selection and hyperparameter tuning) for XGBoost CatBoost in all the other downstream tasks of this paper. Additionally, we use Vector AutoRegression (VAR) (Lütkepohl et al., 2005), which is a common (non Deep Learning based) autoregressive model for multivariate time series, where we transform categorical features in numerical by assigning a numerical value to each element in the vocubaluary V_j. In case of VAR, the field values of all the rows of the time series are directly input to the model without intermediate engineered features. The results in Tables 4, 5 and 6 show that UniTTab largely outperforms "standard" Machine Learning approaches, and the gap is particularly significant with longer sequences (Table 6), where the hierarchical Transformer architecture of both UniTTab and TabBERT can likely better represent long inter-row dependencies.

Table 6 Pollution prediction task (our partition) with longer time series ($t = 50$)

Downstream task training	Model	RMSE
Fine-tuning	TabBERT†	30.95 (± 0.53)
	UniTTab (ours)	**25.11** (± 0.48)
LSTM	TabBERT Padhi et al. (2021)‡	36.67 (± 0.81)
	LUNA†	43.93 (± 2.83)
	UniTTab (ours)	**28.17** (± 0.51)
Training from scratch	XGBoost†	56.37 (± 0.92)
	CatBoost†	56.26 (± 0.99)
	VAR†	83.37 (± 0.95)

Bold indicates best results

Average and standard deviation results obtained with 5 random seeds.
† Our reproduction using the official code

Fraud detection task. This is the classification task used in Sect. 5.1 (and presented in more detail in Sect. A). Differently from Sect. 5.1, in the experiments of this section we use the full dataset for pre-training ($\sim 4,9$M time series), which leads to a higher absolute performance in the results reported in Table 7. Specifically, we use the original time series length $t = 10$ in Table 7 and we extend the experiments to $t = 50$ in Table 8. These tables shows that UniTTab outperforms all the compared methods, independently on whether an LSTM is used or not and the gain is particularly significant with longer time series (Table 8). Similarly to the Pollution prediction task, in Tables 7 and 8 we also report the results obtained using VAR, XGBoost and CatBoost, with task and dataset specific features and hyperparameters for XGBoost and CatBoost (see above). Also in the Fraud detection task, UniTTab significantly outperforms all the non Deep Learning based methods, and the improvement is particularly significative with longer time series (Table 8).

Loan default prediction task. For this task, we use the PKDD'99 Financial Dataset (Berka, 1999), which is smaller than the other datasets (see Sect. A and B for more details), thus we report the average results obtained with 5 random splits. The dataset consists of the complete (real) transaction history of several bank customers, the average length of which is 232. To increase the number of time series available for pre-training, we set a maximum length value t_{max}, and we use time series with variable length t, where $t \leq t_{max}$. Specifically, for each client, assuming that t_{all} is her/his total number of transactions, if $t_{all} < t_{max}$, then we use $t = t_{all}$. Otherwise, *at each pre-training iteration* we randomly select a time series $\boldsymbol{s}$ of $t = t_{max}$ consecutive transactions over the sequence of all possible t_{all} rows of that client. In Table 9 we present results with t_{max} ranging from 50 to 150. This experiment is important to test whether a model dealing with time series can operate in real life scenarios where the actual row sequence length is variable and it can be very long ($t \geq 50$).

We fine-tune the models similarly to the Fraud detection task (i.e., using a `[CLS]` token, etc.). Both in the fine-tuning and in the testing stage, the length t of a transaction is variable. However, differently from the pre-training stage, if $t_{all} > t_{max}$, then we select the *last* t_{max} transactions.

Table 9 shows that UniTTab outperforms both TabBERT and LUNA, especially when using very long sequences ($t_{max} = 150$). The bottom of that table shows the results

Table 7 Fraud detection task ($t = 10$)

Downstream task training	Model	F1 score
Fine-tuning	TabBERT†	0.910
	TabAConvBERT (Shankaranarayana et al., 2021)‡	0.896
	UniTTab (ours)	**0.915**
LSTM	TabBERT‡(Padhi et al., 2021)	0.860
	LUNA ‡(Han et al., 2022)	0.862
	UniTTab (ours)	**0.914**
Training from scratch	XGBoost†	0.779
	CatBoost†	0.499
	VAR†	0.495

Bold indicates best results

† Our reproduction.‡ Results reported in the corresponding paper

Table 8 Fraud detection task ($t = 50$)

Downstream task training	Model	F1 score
Fine-tuning	TabBERT†	0.777
	UniTTab (ours)	**0.812**
LSTM	TabBERT†Padhi et al. (2021)	0.883
	LUNA†Han et al. (2022)	0.884
	UniTTab (ours)	**0.931**
Training from scratch	XGBoost†	0.349
	CatBoost†	0.337
	VAR†	0.497

Bold indicates best results

† Our reproduction

of VAR, XGBoost and CatBoost, with the usual dataset and task specific features and hyperparameters selection for XGBoost and CatBoost. Additionally, we also report the result obtained by Xu et al. (2020) using Random Forests with 18 features, some of which are computed extracting aggregated information from the time series, while others are socio-demographic information about the client which all the other methods do *not* use. The results in Table 9 show that also in this small dataset UniTTab outperforms the standard Machine Learning methods and, similarly to the other tasks, the benefit is larger with longer time series.

Age prediction task. These experiments are based on the Age2 Dataset and its associated Age prediction downstream task, both described in Sect. A. Similarly to the PKDD'99 Financial Dataset, Age2 is composed of the bank transaction history of different bank customers. Each row (transaction) is composed of $k = 3$ fields (5 after splitting the time stamp in 3 fields). For data augmentation, we follow the same protocol adopted when pre-training on the PKDD'99 Financial Dataset, i.e., we use time series of variable length $t \leq t_{max}$, extracted at random at each iteration from the whole history of each bank account. We use $t_{max} = 50$. Similarly to the Loan default prediction task, during both fine-tuning and

Table 9 Loan default prediction task: average and standard deviation results obtained with 5 random seeds

Pre-training (t_{max})	Model	F1 score
50	TabBERT†	0.611 (± 0.032)
	LUNA†	0.604 (± 0.048)
	UniTTab (ours)	**0.619** (± 0.011)
100	TabBERT†	0.636 (± 0.024)
	LUNA†	0.624 (± 0.075)
	UniTTab (ours)	**0.654** (± 0.032)
150	TabBERT†	0.620 (± 0.024)
	LUNA†	0.637 (± 0.043)
	UniTTab (ours)	**0.673** (± 0.038)
	Random Forest (Xu et al., 2020) ‡	0.2667
	XGBoost†	0.608 (± 0.079)
	CatBoost†	0.527 (± 0.065)
	VAR†	0.474 (± 0.007)

Bold indicates best results

† Our reproduction. ‡ Results reported in the corresponding paper

testing, we use time series with variable length t and, if $t_{all} > t_{max}$, then we select the *last* t_{max} transactions.

Churn prediction task. In this last battery of experiments, we use the large RBAT Dataset and its associated downstream task (both described in Sect. A). Similarly to the PKDD'99 Financial Dataset and the Age2 Dataset, also this dataset is composed of the bank transaction history of different real users. However, RBAT is much more complex than most public datasets, and each transaction (row) of its time series is associated with a specific type. To be precise, there are $n = 3$ types of mutually exclusive transactions (see Sect. 3): (1) generic transactions, with $k_g = 5$ fields, (2) POS transactions, with $k_p = 8$ fields and (3) ATM transactions, with $k_a = 7$ fields. Since the generic transaction fields are shared also by the other two types of transactions, the total number of different fields is 10, which become 12 when the time stamp is split in 3 different fields (day, month, year).

For data augmentation, we follow the same protocol adopted when pre-training on both the PKDD'99 Financial Dataset and the Age2 Dataset, i.e., we use time series of variable length $t \leq t_{max}$, extracted at random at each iteration from the whole history of each account. In this case, we use $t_{max} = 150$. Since the baseline TabBERT cannot deal with different types of transactions and needs a fixed number of fields for each row, when training TabBERT we concatenate all the $k = 10$ fields of all the row types. In a given row $\boldsymbol{r}$, the field values of those attributes which are not included in the type of $\boldsymbol{r}$, are represented using a `[MISSING]` token. Note that this solution is computationally more expensive, especially in the Field Transformer, where the computation costs grow quadratically with k, and the computational benefit of our proposal is larger when n is bigger. For ablation reasons (see Sect. 5.1), we also train a modified version of TabBERT in which we introduce the Row-type dependent embedding (Eq. 1) *without any other changes*. This baseline corresponds to the baseline used in Table 1 (first row), on top of which we add the Row-type dependent embedding described in Sect. 4, thus we call it "TabBERT + Variable Row Types" (TabBERT + VRT).

Table 10 Age prediction task on the Age2 dataset

Downstream task training	Model	F1 score
Fine-tuning	TabBERT†	0.662
	UniTTab (ours)	**0.678**
LSTM	TabBERT‡ Padhi et al. (2021)	0.645
	LUNA ‡ Han et al. (2022)	0.648
	UniTTab (ours)	**0.664**
Training from scratch	XGBoost†	0.542
	CatBoost†	0.622
	VAR†	0.354

Bold indicates best results

† Our reproduction

Table 11 Churn prediction task on the RBAT Dataset

Pre-training (t_{max})	Model	F1 score
150	TabBERT†	0.526
	TabBERT + VRT (ours)	0.536
	UniTTab (ours)	**0.604**
	XGBoost†	0.485
	CatBoost†	0.483
	VAR†	0.472

Bold indicates best results

† Our reproduction

We define the Churn prediction task in Sect. A. Similarly to the Loan default prediction task and the Age prediction task, during both fine-tuning and testing, we use time series with variable length t and, if $t_{all} > t_{max}$, then we select the *last* t_{max} transactions (which are the closest to a possible bank account closure). Since this dataset is much larger than the others, for computational reasons we have not included LUNA in this comparison. However, similarly to the other tasks, we also used VAR, XGBoost and CatBoost with the usual dataset and task specific features and hyperparameter selection for the last two (Table 10).

Table 11 shows that TabBERT + VRT significantly improves the baseline TabBERT, showing that the Row-type dependent embedding has an accuracy benefit on the downstream task which goes beyond its computational advantages. Moreover, also in this task our full method (UniTTab) outperforms TabBERT, VAR, XGBoost and CatBoost with a large margin.

5.3 Effect of pre-training

One of the main advantages of using Deep Learning methods over more traditional Machine Learning approaches, is the possibility to pre-train a large network using self-supervision and a large unsupervised dataset, and then fine-tune the same network on the

available supervised data of a downstream task. For instance, in case of UniTTab, we first pre-train the network using a Masked Token pretext task (Sect. 4) using all the available training data. These data do not need to be annotated, since predicting a masked token is a self-supervised task. Then, we use downstream task-specific annotated training data (which are usually much sparser than the unsupervised data) to fine-tune the network. This is not possible with techniques like VAR, Gradient Boosted Decision Trees or Random Forests, which need labeled data and thus cannot exploit the knowledge contained in the unlabeled samples.

In order to quantify the contribution of the pre-training phase, and to show that this is useful also when the unlabeled dataset is not huge, we use the two smallest datasets, i.e., the Pollution Dataset and the PKDD'99 Financial Dataset, and we pre-train the models with different portions of the pre-traing dataset. Specifically, in both Tables 12 and 13 we indicate the fraction of the pre-training dataset used for each experiment, where zero corresponds to training the models from scratch directly on the (labeled) downstream task data. The results in these tables show that both TabBERT and UniTTab significanltly benefit from the pre-training phase, despite both datasets have a small-medium size, and even when only a small portion of the unlabeled data (e.g., 0.25) is used for pre-training. Moreover, Table 12 shows that UniTTab can drastically outperform the non Deep Learning based methods (VAR, XGBoost and CatBoost) even with no pre-training. Conversely, in the smallest dataset (PKDD'99), XGBoost beats all the other methods with a large margin when no pre-training is used (Table 13). Note that this dataset is very small, with only 478 labeled samples (Sect. B), and training from scratch networks with 100 M parameters (Sect. C) with these scarce data is very difficult. However, when pre-training is used, UniTTab gets a significantly higher F1 score than XGBoost.0

Table 12 Pollution prediction task (our partition): impact of different portions of the pre-training dataset

Pre-training portion	Model	RMSE
0	TabBERT†	33.37
	UniTTab (ours)	**27.10**
	XGBoost†	48.89
	CatBoost†	47.09
	VAR†	84.05
0.25	TabBERT†	30.78
	UniTTab (ours)	**24.55**
0.5	TabBERT†	30.11
	UniTTab (ours)	**23.73**
0.75	TabBERT†	29.51
	UniTTab (ours)	**23.32**
1	TabBERT†	29.13
	UniTTab (ours)	**23.29**

Bold indicates best results

† Our reproduction

Table 13 Loan default prediction task: impact of different portions of the pre-training dataset

Pre-training portion	Model	F1 score
0	TabBERT†	0.526 (± 0.009)
	UniTTab (ours)	0.548 (± 0.015)
	Random Forest (Xu et al., 2020) ‡	0.2667
	XGBoost†	**0.608** (± 0.079)
	CatBoost†	0.527 (± 0.065)
	VAR†	03474 (± 0.007)
0.25	TabBERT†	0.586 (± 0.023)
	UniTTab (ours)	**0.593** (± 0.022)
0.5	TabBERT†	0.577 (± 0.022)
	UniTTab (ours)	**0.607** (± 0.029)
0.75	TabBERT†	**0.628** (± 0.019)
	UniTTab (ours)	0.620 (± 0.027)
1	TabBERT†	0.620 (± 0.024)
	UniTTab (ours)	**0.673** (± 0.038)

Bold indicates best results

We report average and standard deviation results obtained with 5 random seeds. For both TabBERT and UniTTab, we keep fixed $t_{max} = 150$.† Our reproduction.‡ Results reported in the corresponding paper

6 Conclusions

We proposed UniTTab, a hierarchical Transformer architecture which can uniformly process highly heterogeneous time series of tabular data with variable lengths, including categorical and numerical values, as well as rows with different internal structure and type. UniTTab can be pre-trained using a uniform Masked Token task (independently of the feature type input), and fine-tuned for different tasks. Our experiments show that the proposed method consistently outperforms the state-of-the-art tabular time series approaches, usually with a large margin. We believe that our architecture and our unified Masked Token pre-training can pave the way to large scale pre-trained foundation models in the domain of tabular data with heterogeneous attributes.

Appendix A: Experimental setting

In this section, we present in more detail the datasets and the associated downstream tasks we used for our evaluations. Generally speaking, the five datasets we used are very different from each other in terms of size, attributes, complexity and downstream tasks. Moreover, we use two main protocols to extract the time series and split the data in training and testing partitions, the first proposed in Padhi et al. (2021), and the second proposed in this paper. Specifically, for both the Pollution Dataset and the Transaction Dataset we follow the protocol proposed in Padhi et al. (2021), in which time series are extracted from longer sequences using a temporal sliding window (whose length corresponds to the time series length t) and a stride. These time series are then randomly

split in training and testing sequences using a fixed training/testing ratio and used for the corresponding downstream task. Note that the *pre-training* data include the downstream task testing samples, however, *no downstream task label is used in pre-training*, since this phase is completely unsupervised. Even if a clearer separation between the pre-training dataset and the downstream task testing dataset would be desirable, we followed the protocol proposed in Padhi et al. (2021) to make possible a comparison with the methods that have been evaluated with those benchmarks using the same approach (Padhi et al., 2021; Han et al., 2022; Shankaranarayana et al., 2021). On the other hand, for the other three datasets adopted in this paper, the PKDD'99 Financial Dataset, the Age2 Dataset and the RBAT Dataset (all bank transaction datasets), we split the time series based on the bank client, which leads to a more realistic scenario and a sharper training–testing separation. Specifically, we create two data partitions (training and testing) based on the bank customer identity, and in each partition we collect the entire transaction history of the corresponding clients. Therefore, the history of a given customer can either belong (entirely) to the training data or (entirely) to the testing data. The training split is used both in the pre-training phase and in the downstream task training phase, while testing data are observed by the models only at the time of evaluating the downstream task. Finally, as explained in Sect. 5.2, for each dataset we define a t_{max} length and we extract time series with variable length t, where $t \leq t_{max}$. We provide below more details on each dataset/task, while in Sect. B we show the statistics of each dataset.

The Pollution Dataset (Liang et al., 2015) is a public UCI dataset based on pollution data collected from 12 monitoring sites. Every row is composed of $k = 11$ fields ($k = 14$ using our time stamp representation with 4 fields) and it was adopted in Padhi et al. (2021) to extract time series using the concatenation of t time-dependent, consecutive rows, obtained with a t long sliding window and a stride of 5 (see below). Following (Padhi et al., 2021), in Tables 4 and 5 we *pre-train* all the models using 76 K time series with length $t = 10$.

For evaluation, we adopt the Pollution prediction (supervised) downstream task (Padhi et al., 2021), consisting in predicting the air pollution concentration. For this task, (Padhi et al., 2021) use 45 K (supervised) time series samples for training and 15 K (supervised) samples for testing, in which the time series are obtained using a sliding window and a stride of 5. However, using this stride, there is an overlapping between adjacent sequences, and, since the training–testing splits of the downstream task are obtained by random sampling of the time series, the two splits may have a partial information leak. To avoid this, we use a stride of 10 (with no overlapping), which leads to a downstream task training–testing partition different from the one used in Padhi et al. (2021), with 23 K training and 7.6 K testing sequences. As mentioned in Sect. 5, we call "our partition" the training–testing random splits obtained using a stride of 10, and "original partition" the training–testing random splits used in Padhi et al. (2021) and based on a stride of 5 (see Sect. B for more details).

Finally, in Table 6 we use time series with length $t = 50$ and a stride of 50, which leads to ~7.6 K unsupervised time series used for pre-training, ~6.1 K used for the supervised downstream task training and ~1.5 K testing sequences.

The Transaction Dataset (Padhi et al., 2021) is a synthetic dataset created with heuristic rules to generate realistic credit card transactions (see Fig. 1). The dataset is composed of 24 M transactions from 20,000 users. There are $k = 10$ attributes, which become $k = 13$ when the time stamp is split in 4 different fields (Sect. 4). Following (Padhi et al., 2021), a

time series is composed of $t = 10$ transactions (i.e., rows), and, for pre-training, the stride of the sliding window is 5, which leads to a total number of 4.87M pre-training time series.

The associated Fraud detection downstream task is a binary classification task based on the prediction of the fraudulent label, associated with a subset of samples. Following (Padhi et al., 2021), we use 1.9M labeled samples (with $t = 10$) for the supervised training stage. Note that, for the downstream task, (Padhi et al., 2021) extract time series using a stride of 10, thus, differently from the Pollution prediction downstream task, there is no overlapping between two adjacent time series and no information leak in the splits defined in Padhi et al. (2021). However, since the positive and the negative classes are highly unbalanced, the positive samples are upsampled (for more details, we refer to (Padhi et al., 2021)). The testing set is composed of 487 K time series.

For computational reasons, in the ablation experiments presented in Sect. 5.1 (Tables 1, 2 and 3) we use only 400 K randomly selected time-series for pre-training, of which 360 K are used for the downstream task training and 40 K for testing. Finally, in Table 8, we use $t = 50$ and a stride of 50, which leads to ~475 K unsupervised time series for pre-training, ~428 K supervised time series for the downstream task training and ~47 K supervised downstream task testing time series.

The PKDD'99 Financial Dataset (Berka, 1999) is a collection of real anonymized financial data of a Czech bank. Besides other client specific information (which we do not use), this dataset contains bank account transactions of 4,500 clients. We use $k = 6$ fields to represent each transaction (row), which become $k = 8$ after splitting the time stamp in 3 different fields. On average, the total length of transaction history for a given customer is 232.

The associated Loan default prediction downstream task is a binary classification task consisting in predicting how likely a client is going to default the loan. For this task, we use 682 clients, jointly with their loan ground truth label, which, following (Xu et al., 2020), are split in 478 clients for training and 204 for testing, and we report the average results obtained with 5 random splits. Note that the testing clients are not used for pre-training and we removed (from the pre-training, the fine-tuning and the testing data) all those transactions directly related to the loan payment. Moreover, both at fine-tuning and at testing time, we cut the transaction history of each client before the loan started.

The Age2 Dataset (Fursov et al., 2021) is a public dataset composed of real bank transactions of different clients. Specifically, there are 43 K clients, which we split in 39 K for training (used both in the pre-training stage and in the downstream task training phase) and 4 K for testing (used only for evaluation). The average length of the transaction history of each client is ~84 rows. As shown in Sect. 5.2, time series of variable length are extracted from each client with a maximum length of $t_{max} = 50$.

The downstream task requires the model to predict the age of the client given a time series. Specifically, we formulate this task as a binary classification task, where the models should predict whether the customer is over 30 years old.

The RBAT Dataset is a proprietary dataset provided by an international bank,[1] which is composed of several hundred thousands real bank account transactions of private clients. From this datasets, we have randomly selected 100 K bank accounts, corresponding to about 32.5M transactions (i.e., rows), which we used for both pre-training and fine-tuning on the donwstream task, and another 20 K accounts used only for the downstream task testing. Overall, the time series transactions span about 2 years, from 2021

[1] For both privacy and commercial reasons, this dataset cannot be released.

to 2022. For a given account, the average transaction history length is 325. There are $n = 3$ types of transactions (see Sect. 5.2), and, in the future, we plan to extend our experiments with larger portions of this dataset, including other types of transactions ($n > 3$).

The Churn prediction downstream task consists in predicting a bank account closure after a period of one month from the last transaction considered. In more detail, at inference time, given a time series $\boldsymbol{s}$, extracted from a given bank account, the model should predict a possible closure of that account which can happen any day of the month following the last transaction contained in $\boldsymbol{s}$ (which is a supervised information provided at fine-tuning time).

Appendix B: Dataset statistics

In Tables 14 and 15 we report the main characteristics of the datasets used in our experiments, including our RBAT Dataset. Specifically, both "Pre-training samples" and "Downstream training samples" refer to the original number of time series samples *before* any upsampling or data-augmentation process. The reported number of attributes include the time stamp counted as a single field. For the Pollution Dataset, we also report the statistics of our partition, obtained using a non-overlapping stride which guarantees that there is no information leak between the training and the testing split (Sect. A). For both the Pollution and Transaction datasets, we also report the statistics corresponding to time series of length $t = 50$, used in Tables 6 and 8, respectively. Finally, the last column of Table 14 refers to the random subset of the Transaction Dataset used in the ablation experiments of Sect. 5.1.

Table 14 Dataset statistics: Pollution Dataset and Transaction Dataset

	Pollution Dataset			Transaction Dataset		
	Our partition	Original partition	$t = 50$	Original	$t = 50$	Ablation
Total dataset rows	382,168	382,168	382,168	24,386,900	24,386,900	4,000,000
Number of attributes (k)	11	11	11	10	10	10
Number of categorical fields	1	1	1	8	8	8
Number of numerical fields	10	10	10	2	2	2
Time series length (t)	10	10	50	10	50	10
Pre-training samples	76,414	76,414	7638	4,874,597	475,066	400,000
Downstream training samples	22,927	45,850	6114	1,950,224	427,559	360,000
Testing samples	7642	15,282	1524	487,556	47,507	40,000

Table 15 Dataset statistics: PKDD'99 Financial Dataset, Age2 Dataset and RBAT Dataset

	PKDD'99 Financial Dataset	Age2 Dataset	RBAT Dataset
Total dataset rows	1,042,740	3,652,757	39,040,010
Number of attributes (k)	6	11	10
Number of categorical fields	3	1	7
Number of numerical fields	3	10	3
Time series length (t)	Variable	Variable	Variable
Pre-training samples	4500	38,961	100,000
Downstream training samples	478	38,961	100,000
Testing samples	204	4328	20,000

Appendix C: Implementation details

Following TabBERT, we use Positional Encoding only in the Sequence Transformer, being the attribute specific embeddings of the Field Transformer sufficient to the network to distinguish the specific field (intuitively, the attribute order in a row does not matter, once the network can distinguish an attribute from the others). Table 16 shows all the hyperparameter values used in UniTTab, most of which are shared with TabBERT.

We used Pytorch 1.11 and we trained all the models on 4 GPUs NVIDIA RTX A6000 (48 G memory). Note that the model sizes reported in Table 16 vary across datasets almost only because of the difference in the size of the corresponding attribute vocabularies, which is an inherently task-dependent hyperparameter.

Regression loss. In the ablation experiments of Table 1 (Sect. 5.1) we have indicated with "Regression loss" the use of the MSE loss function for the numerical features, which is summed with the Cross Entropy loss used for the categorical attributes. We provide more details below. When using "Regression loss", Eq. 5 is replaced by the following objective function:

$$\min_{\theta} - \sum_{v \in \boldsymbol{s}, Cat(v)} \mathbb{1}_{Masked(v)} \log p_\theta(quantize(v)|\boldsymbol{s}') + \lambda \sum_{v \in \boldsymbol{s}, \neg Cat(v)} \mathbb{1}_{Masked(v)} (\hat{v} - f_\theta(\boldsymbol{s}'))^2. \tag{6}$$

Note that we do not use $\boldsymbol{p}(v)$ because, in this ablation experiment, Label Smoothing is not used (Sect. 5.1). In Eq. 6, $Cat(v)$ is true when v is a categorical attribute. In that case, the standard Cross Entropy loss is used, where the posterior $p_\theta(\cdot)$ is obtained with a softmax layer on top of the final MLP network (Sect. 3). On the other hand, for numerical features ($\neg Cat(v)$), we use a linear regression layer (indicated by $f_\theta(\cdot)$ in Eq. 6). Moreover, $\hat{v}$ is the normalized v value. In more detail, following (Padhi et al., 2021), numerical feature values v are pre-processed by applying a log-scale and a standardization using the mean and the standard deviation of the training sample values.

The value of λ is obtained using the following heuristic. We train the entire network for one epoch using $\lambda = 1$. Then, we use a few batches ($Batch$) to separately compute the average magnitude of the two losses:

Table 16 UniTTab hyperparameter values

	Pollution Dataset	Transaction Dataset	PKDD'99 Financial Dataset	RBAT Dataset	Age2 Dataset
Optimizer	AdamW	AdamW	AdamW	AdamW	AdamW
Learning rate	5e-05	5e-05	5e-05	5e-05	5e-05
Dropout	0.1	0.1	0.1	0.1	0.1
Label Smoothing (ϵ)	0.1	0.1	0.1	0.1	0.1
Batch size	120	120	120	120	120
Model size (parameters)	137 M	300 M	100 M	90 M	44 M
Field Transformer layers	1	1	1	1	1
Field Transformer heads	8	8	8	8	8
Field Transformer embedding size (d)	72	72	72	72	72
Sequence Transformer layers	12	12	12	12	12
Sequence Transformer heads	12	12	12	12	12
Sequence Transformer embedding size (m)	1080	1080	1080	1080	1080
Pre-training epochs	12	5	20	20	20
Pre-training iterations	7.6k	203k	0.76k	17k	21k
Fine-tuning epochs	10	10	30	30	20
Fine-tuning iterations	5.7k	485k	0.21k	30k	21k
Total training time	1.5 h	4 days	15 min	8 h	2 h

$$L_C = \sum_{\boldsymbol{s} \in Batch} \sum_{v \in \boldsymbol{s}, Cat(v)} \mathbb{1}_{Masked(v)} \log p_\theta(quantize(v)|\boldsymbol{s}'), \tag{7}$$

$$L_R = \sum_{\boldsymbol{s} \in Batch} \sum_{v \in \boldsymbol{s}, \neg Cat(v)} \mathbb{1}_{Masked(v)} (\hat{v} - f_\theta(\boldsymbol{s}'))^2. \tag{8}$$

Finally, we impose that L_C must be equal to λL_R and we solve for λ:

$$L_C = \lambda L_R; \quad \lambda = \frac{L_C}{L_R}. \tag{9}$$

Using this heuristic, for the Pollution Dataset, we get $\lambda = 50$ and, for the Transaction Dataset, we get $\lambda = 25$. As mentioned in Sect. 1, the need to set the relative weighting factor (λ) for each dataset makes difficult to simultaneously use heterogeneous loss functions. On the other hand, our proposed uniform Masked Token pretext task (Eq. 5) besides being more effective from a performance point of view (Sect. 5.1) is also much more easy to use.

Author contributions Simone Luetto and Fabrizio Garuti wrote the code and conducted all the experiments. Enver Sangineto and Rita Cucchiara, jointly with Luetto and Garuti, proposed the ideas the method is based on and conducted the research. Lorenzo Forni provided the funding and contributed to the research with his expertise in financial topics. All authors wrote and reviewed the paper.

Funding Open access funding provided by Università degli Studi di Modena e Reggio Emilia within the CRUI-CARE Agreement. Partial financial support was received from Prometeia Spa, Bologna, Italy, and from Prometeia Associazione, Bologna, Italy.

Data availibility All the dataset are public and available online, except our private RBAT Dataset, which, for both privacy and commercial reasons, cannot be released.

Declarations

Conflict of interest One of the authors, specifically Enver Sangineto, is member of the Editorial Board of the Machine Learning Journal.

References

Arnab, A., Dehghani, M., Heigold, G., Sun, C., Lucic, M., & Schmid, C. (2021). Vivit: A video vision transformer. In *ICCV*.

Bao, H., Dong, L., & Wei, F. (2022). BEiT: BERT pre-training of image transformers. In *ICLR*.

Benjelloun, O., Chen, S., spsampsps Noy, N. (2020). Google dataset search by the numbers. In *International semantic web conference* Springer.

Berka, P. (1999). Workshop notes on Discovery Challenge PKDD'99. https://sorry.vse.cz/~berka/challenge/pkdd1999/berka.htm

Beyazit, E., Kozaczuk, J., Li, B., Wallace, V., & Fadlallah, B. H. (2023). An inductive bias for tabular deep learning. In *NeurIPS*.

Borisov, V., Seßler, K., Leemann, T., Pawelczyk, M., & Kasneci, G. (2022). Language models are realistic tabular data generators. arXiv:2210.06280

Borisov, V., Broelemann, K., Kasneci, E., & Kasneci, G. (2023). DeepTLF: Robust deep neural networks for heterogeneous tabular data. *International Journal of Data Science and Analytics, 16*(1), 85–100.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language models are few-shot learners. arXiv:2005.14165

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh - a python package). *Neurocomputing, 307*, 72–77.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Fursov, I., Morozov, M., Kaploukhaya, N., Kovtun, E., Rivera-Castro, R., Gusev, G., Babaev, D., Kireev, I., Zaytsev, A., & Burnaev, E. (2021). Adversarial attacks on deep models for financial transaction records. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining (KDD)*

Gorishniy, Y., Rubachev, I., Khrulkov, V., & Babenko, A. (2021). Revisiting deep learning models for tabular data. In *NeurIPS*.

Han, H., Xu, J., Zhou, M., Shao, Y., Han, S., & Zhang, D. (2022). LUNA: language Lunderstanding with number augmentations on transformers via number plugins and pre-training. arXiv:2212.02691
Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. In *NeurIPS*.
Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. S. (2020). Tabtransformer: Tabular data modeling using contextual embeddings. arXiv:2012.06678
Jiang, J., Zhou, K., Dong, Z., Ye, K., Zhao, W. X., & Wen, J.-R. (2023). StructGPT: A general framework for large language model to reason on structured data. arXiv:2305.09645
Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. In *ICLR*.
Kossen, J., Band, N., Lyle, C., Gomez, A. N., Rainforth, T., & Gal, Y. (2021). Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In *NeurIPS*.
Kotios, D., Makridis, G., Fatouros, G., & Kyriazis, D. (2022). Deep learning enhancing banking services: A hybrid transaction classification and cash flow prediction approach. *Journal of Big Data, 9*, 100.
Li, J., Peng, J., Li, H., & Chen, L. (2024). UniCL: A universal contrastive learning framework for large time series models. arXiv:2405.10597
Liang, X., Zou, T., Guo, B., Li, S., Zhang, H., Zhang, S., Huang, H., & Chen, S. (2015). Assessing Beijing's PM2.5 pollution: Severity, weather impact, APEC and winter heating. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 471*(2182), 20150257.
Lütkepohl, H. (2005). New Introduction to Multiple Time Series Analysis. Springer Books (978-3-540-27752-1).
Lyu, F., Tang, X., Zhu, H., Guo, H., Zhang, Y., Tang, R., & Liu, X. (2022). OptEmbed: learning optimal embedding table for click-through rate prediction. In M. A. Hasan, L. Xiong, L. (Eds.) *Proceedings of the 31st ACM international conference on information & knowledge management*.
Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., spsampsps Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.
Narayan, A., Chami, I., Orr, L. J., & Ré, C. (2022). Can foundation models wrangle your data? *Proceedings of the VLDB Endowment, 16*(4), 738–746.
Padhi, I., Schiff, Y., Melnyk, I., Rigotti, M., Mroueh, Y., Dognin, P.L., Ross, J., Nair, R., & Altman, E. (2021). Tabular transformers for modeling multivariate time series. In *IEEE international conference on acoustics, speech and signal processing, ICASSP*.
Prokhorenkova, L. O., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). Catboost: unbiased boosting with categorical features. In *NeurIPS*.
Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., & Clark, J. (2021). Learning transferable visual models from natural language supervision. In *ICML*.
Rahaman, N., Baratin, A., Arpit, D., Draxler, F., Lin, M., Hamprecht, F. A., Bengio, Y., & Courville, A. C. (2019). On the spectral bias of neural networks. In *ICML*.
Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., & Sutskever, I. (2021). Zero-shot text-to-image generation. In *ICML*.
Schäfl, B., Gruber, L., Bitto-Nemling, A., & Hochreiter, S. (2022). Hopular: Modern hopfield networks for tabular data. arXiv:2206.00664
Shankaranarayana, S. M., & Runje, D. (2021). Attention augmented convolutional transformer for tabular time-series. In *2021 international conference on data mining, ICDM 2021—Workshops*.
Solatorio, A. V., & Dupriez, O. (2023). REaLTabFormer: Generating realistic relational and tabular data using transformers. arXiv:2302.02041
Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C. B., & Goldstein, T. (2021). SAINT: improved neural networks for tabular data via row attention and contrastive pre-training. arXiv:2106.01342
Suh, N., Yang, Y., Hsieh, D.-Y., Luan, Q., Xu, S., Zhu, S., & Cheng, G. (2024). TimeAutoDiff: Combining Autoencoder and Diffusion model for time series tabular data synthesizing. arXiv:2406.16028
Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision.
Trirat, P., Shin, Y., Kang, J., Nam, Y., Na, J., Bae, M., Kim, J., Kim, B., & Lee, J.-G. (2024). Universal time-series representation learning: A survey. arXiv:2401.03717
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*.
Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2022). Transformers in time series: A survey. arXiv:2202.07125
Wu, Y., Rabe, M. N., Hutchins, D., & Szegedy, C. (2022). Memorizing transformers. In *ICLR*.
Xu, Z. (2020). Loan Default Prediction with Berka Dataset. https://towardsdatascience.com/loan-default-prediction-an-end-to-end-ml-project-with-real-bank-data-part-1-1405f7aecb9e
Ye, H.-J., Liu, S.-Y., Cai, H.-R., Zhou, Q.-L., & Zhan, D.-C. (2024). A closer look at deep learning on tabular data. arXiv:2407.00956

Zhang, H., Zhang, J., Shen, Z., Srinivasan, B., Qin, X., Faloutsos, C., Rangwala, H., & Karypis, G. (2024). Mixed-type tabular data synthesis with score-based diffusion in latent space. In *ICLR*.

Authors and Affiliations

Simone Luetto[1] · Fabrizio Garuti[2,3] · Enver Sangineto[3] · Lorenzo Forni[2,4] · Rita Cucchiara[3,5]

✉ Enver Sangineto
enver.sangineto@unimore.it

Simone Luetto
simone.luetto@prometeia.com

Fabrizio Garuti
fabrizio.garuti@prometeia.com

Lorenzo Forni
lorenzo.forni@prometeia.com

Rita Cucchiara
rita.cucchiara@unimore.it

[1] Prometeia Spa, Bologna, Italy

[2] Prometeia Associazione, Bologna, Italy

[3] AImageLab, University of Modena and Reggio Emilia, Modena, Italy

[4] Department of Economics, University of Padova, Padua, Italy

[5] Istituto di Informatica e Telematica CNR, Pisa, Italy