MambaTab: A Simple Yet Effective Approach for Handling Tabular Data

Md Atik Ahamed[1] , **Qiang Cheng**[1,2*]
[1]Department of Computer Science, University of Kentucky, Lexington, KY, USA
[2]Institute for Biomedical Informatics, University of Kentucky, Lexington, KY, USA
{atikahamed, qiang.cheng}@uky.edu

Abstract

Tabular data remains ubiquitous across domains despite growing use of images and texts for machine learning. While deep learning models like convolutional neural networks and transformers achieve strong performance on tabular data, they require extensive data preprocessing, tuning, and resources, limiting accessibility and scalability. This work develops an innovative approach based on a structured state-space model (SSM), MambaTab, for tabular data. SSMs have strong capabilities for efficiently extracting effective representations from data with long-range dependencies. MambaTab leverages Mamba, an emerging SSM variant, for end-to-end supervised learning on tables. Compared to state-of-the-art baselines, MambaTab delivers superior performance while requiring significantly fewer parameters and minimal preprocessing, as empirically validated on diverse benchmark datasets. MambaTab's efficiency, scalability, generalizability, and predictive gains signify it as a lightweight, "out-of-the-box" solution for diverse tabular data with promise for enabling wider practical applications.

1 Introduction

Tabular data remains the predominant data type across industrial, healthcare, academic, and various other domains due to its structured format, despite the recent trend of using images and natural language processing in machine learning. To handle tabular data, numerous strategies have been developed, including machine learning (ML) techniques that use traditional shallow models as well as newer deep learning (DL) architectures. Foundational models such as convolutional neural networks (CNNs) and Transformers [Vaswani *et al.*, 2017] have been actively explored and tailored to tabular data modeling in DL, enabling impactful insights and analytics.

While powerful, state-of-the-art deep tabular models typically require a large number of learning parameters, extensive data preprocessing, and hyperparameter tuning. This demands significant computational and human resources, which can impose barriers to developing and deploying these complex models for tabular data, thereby impeding their wider application. Moreover, almost all existing tabular learning methods, except TransTab [Wang and Sun, 2022], operate under vanilla supervised learning, requiring identical train and test table structures. They are not well-suited for feature incremental learning where features are sequentially added; under such a setting, they have to either drop new features or old data, leading to insufficient use of training data. It is desirable to have the ability to continuously learn from new features.

To address these challenges, we introduce a new approach for tabular data based on structured state-space models (SSMs) [Gu *et al.*, 2021b] [Gu *et al.*, 2021a] [Fu *et al.*, 2022]. These models can be interpreted as a combination of CNNs and recursive neural networks, having advantages of both types of models. They offer parameter efficiency, scalability, and strong capabilities for learning representations from varied data, particularly for sequential data with long-range dependencies. To tap into these potential advantages, we leverage SSMs as an alternative to CNNs or Transformers for modeling tabular data.

Specifically, we leverage Mamba [Gu and Dao, 2023], an emerging SSM variant, as a critical building block to build a new supervised model for tabular data called *MambaTab*. This proposed model has several key advantages over existing models. Thanks to Mamba's innovative approach as an SSM, MambaTab not only requires significantly fewer model weights and exhibits linear parameter growth, but also inherently aligns well with feature incremental learning. Additionally, MambaTab has a simple architecture needing minimal data preprocessing or manual tuning. Finally, MambaTab outperforms state-of-the-art baselines, including Transformers and CNN-based models as well as classic learning models.

We extensively benchmark MambaTab against state-of-the-art tabular data approaches. Experiments under two different settings - vanilla supervised learning and feature incremental learning - on 8 public datasets demonstrate MambaTab's superior performance. It consistently and significantly outperforms the state-of-the-art baselines, including Transformer-based models, while using a small fraction, typically $< 1\%$, of their parameters.

In summary, the key innovations and contributions of MambaTab are:

- Extremely small model size and number of learning parameters

[*] Correspondence should be addressed to: qiang.cheng@uky.edu

- Linear scalability of model parameters in Mamba blocks, number of features, or sequence length
- Effective end-to-end training and inference with minimal data wrangling needed, in particular, naturally suitable for feature incremental learning
- Superior performance over state-of-the-art tabular learning approaches

As the first Mamba-based architecture for tabular data, MambaTab's advantages suggest that it can serve as an *out-of-the-box*, plug-and-play model for tabular data on systems with varying computational resources. This holds promise to enable wide applicability across diverse practical settings.

2 Related Work

In this section, we briefly review existing approaches for learning from tabular data. We roughly categorize them into three groups based on whether they utilize classical shallow models, deep learning like CNN- or contemporary Transformer-based architectures, or self-supervised learning strategies.

Classic Learning-based Approaches A variety of models exist based on classic ML techniques such as logistic regression (LR), XGBoost [Chen and Guestrin, 2016] [Zhang *et al.*, 2020], and multilayer perceptron (MLP). For example, an MLP variant called self-normalizing neural network (SNN) [Klambauer *et al.*, 2017] uses the scaled exponential linear unit (SELU) specifically for tabular data. SNN neuron activations automatically converge towards zero mean and unit variance, enabling high-level abstract representations.

Deep Learning-based Supervised Models TabNet [Arik and Pfister, 2021] is a DL model for tabular data modeling based on attention mechanism. It uses sequential attention to choose which features to attend to at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features. TabNet is shown to have high performance on a wide range of non-performance-saturated tabular datasets and yield interpretable feature attributions or insights into the global model behavior. Deep cross networks (DCN) [Wang *et al.*, 2017] constructs a new network structure consisting of two parts: a deep network and a cross network. The deep network is a standard feed-forward network that can learn high-order feature interactions. The cross network is a new component that can explicitly apply automatic feature crossing with a special operation called vector-wise cross. DCN is shown to be efficient in learning certain bounded-degree feature interactions.

A variety of models have been developed with Transformers as building blocks. AutoInt [Song *et al.*, 2019] uses Transformers to learn the importance of different input features. By relying on self-attention networks, this model can automatically learn high-order feature interactions in a data-driven way. TabTransformer [Huang *et al.*, 2020] is also built upon self-attention based Transformers, which transform the embeddings of categorical features into robust contextual embeddings to achieve higher prediction accuracy. The contextual embeddings are shown to be highly robust against both missing and noisy data features and provide better interpretability. Moreover, FT-Transformer [Gorishniy *et al.*, 2021] converts categorical features into continuous embeddings using a tokenizer, and models the interactions between the continuous embeddings with Transformers.

Self-Supervised Learning-based Models Recently several approaches have been developed to pre-train deep learning models using self-supervised strategies. VIME [Yoon *et al.*, 2020] introduces a tabular data augmentation method for self- and semi-supervised learning frameworks. It builds a pretext task of estimating mask vectors from corrupted tabular data in addition to the reconstruction pretext task. Transformer is used as a component of the pre-trained model. SCARF [Bahri *et al.*, 2021] is a self-supervised contrastive learning technique for pre-training on real-world tabular datasets. It forms views for contrastive learning by corrupting a random subset of features. TransTab [Wang and Sun, 2022] proposes a novel framework for learning from tabular data across tables with different learning strategies such as self-supervised and feature incremental learning. Transformer is used as an integral component of the framework.

3 Method

In this section, we present our approach for robust learning of tabular data classification, aiming to improve performance through a simple, efficient, yet effective method. Below we describe each component of our method and the working procedures.

Data preprocessing We consider a tabular dataset, $\{F_i, y_i\}_{i=1}^{m}$, where the features of the i-th sample are represented by $F_i = \{v_{i,j}\}_{j=1}^{n}$, its corresponding label is $y_i \in \{0, 1\}$, and $v_{i,j}$ can be categorical, binary or numerical. We treat both binary and categorical features as categorical and utilize an ordinal encoder for encoding them, as shown in Figure 1. Unlike TransTab [Wang and Sun, 2022], our method does not require manual identification of feature types such as categorical, numerical, or binary. We keep numerical features unchanged in the dataset and handle missing values by imputing the mode. This preprocessing preserves the feature set cardinality, i.e. $n(F_i) = n({F_i}'))$, where $n(F_i)$ and $n({F_i}')$ are the numbers of features before and after processing. Before feeding data into our model, we normalize values $v_{i,j} \in [0, 1]$ using min-max scaling:

$$ {v'}_{i,j} = \frac{v_{i,j} - min_{i,j=1}^{i=n,j=m}(v_{i,j})}{max_{i,j=1}^{i=n,j=m}(v_{i,j}) - min_{i,j=1}^{i=n,j=m}(v_{i,j})}. \quad (1) $$

Embedding representation learning After getting the preprocessed data, we utilize a fully connected layer to learn an embedded representation from the processed features. This is necessary to provide more meaningful representations as input to the proposed architecture. Moreover, while the ordinal-encoder enforces ordered representations for categorical features, some may not necessarily have inherent ordering among them. The embedding representation learner enables our method to learn multi-dimensional representations directly from the features without relying on the imposed ordering. In addition, this embedding representation learning can ensure that the downstream Mamba blocks have the same input

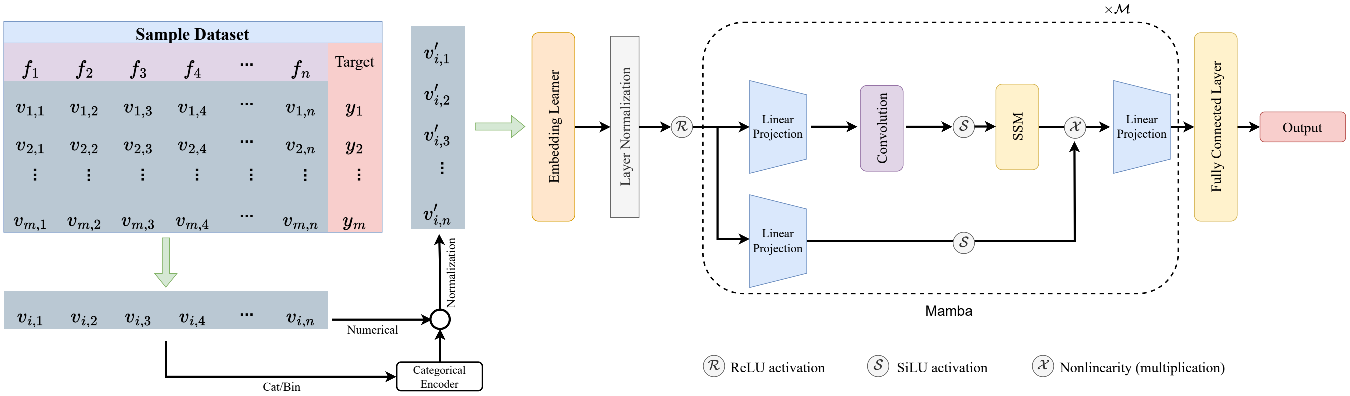

Sample Dataset
f_1 f_2 f_3 f_4 ··· f_n
Target
$v_{1,1}$ $v_{1,2}$ $v_{1,3}$ $v_{1,4}$ ··· $v_{1,n}$ y_1
$v_{2,1}$ $v_{2,2}$ $v_{2,3}$ $v_{2,4}$ ··· $v_{2,n}$ y_2
$v_{m,1}$ $v_{m,2}$ $v_{m,3}$ $v_{m,4}$ ··· $v_{m,n}$ y_m
$v_{i,1}$ $v_{i,2}$ $v_{i,3}$ $v_{i,4}$ ··· $v_{i,n}$
Numerical
Cat/Bin
Categorical Encoder
Normalization
$v'_{i,1}$ $v'_{i,2}$ $v'_{i,3}$ $v'_{i,n}$
Embedding Learner
Layer Normalization
$\times\mathcal{M}$
Linear Projection
Convolution
SSM
Linear Projection
Linear Projection
Mamba
Fully Connected Layer
Output
$\mathcal{R}$ ReLU activation
$\mathcal{S}$ SiLU activation
$\mathcal{X}$ Nonlinearity (multiplication)

Figure 1: Schematic diagram of our proposed method (MambaTab). **Left:** Data preprocessing and representation learning. The embedding learner module is critical to ensure the embedded feature dimension is the same before and after new features are added under incremental learning. **Right:** Conversion of input data to prediction values via Mamba and a fully connected layer.

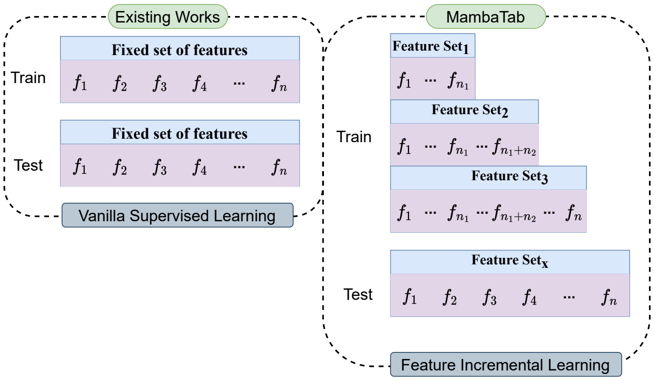

Existing Works
Train
Fixed set of features
f_1 f_2 f_3 f_4 ··· f_n
Test
Fixed set of features
f_1 f_2 f_3 f_4 ··· f_n
Vanilla Supervised Learning
MambaTab
Train
Feature Set$_1$
f_1 ··· f_{n_1}
Feature Set$_2$
f_1 ··· f_{n_1} ··· $f_{n_1+n_2}$
Feature Set$_3$
f_1 ··· f_{n_1} ··· $f_{n_1+n_2}$ ··· f_n
Test
Feature Set$_X$
f_1 f_2 f_3 f_4 ··· f_n
Feature Incremental Learning

Figure 2: Illustration of feature incremental learning setting. While most existing methods are capable of only learning from a fixed set of features, our method MambaTab and existing TransTab can learn under an incremental feature setting. Here, Feature Set$_i$, $i = 1, 2, 3$, have incrementally added features. Feature Set$_X$ represents the set of features for test data.

feature dimensions during training and testing under incremental feature learning. We demonstrate the feature incremental learning setting in Figure 2. While most existing methods, except TransTab, are only capable of learning from a fixed set of features, our method MambaTab can learn and transfer weights from Feature Set$_1$ to Feature Set$_2$, and so on. We also utilize layer normalization [Ba *et al.*, 2016] instead of batch normalization [Ioffe and Szegedy, 2015] on the learned embedded representations due to its independence of batch size.

Cascading Mamba Blocks After getting the normalized embedded representations from layer normalization, we apply ReLU activation [Agarap, 2018] and pass the resulting values $\{u_k^i\}$, with u_k^i being the k-th token for example i, to a Mamba block [Gu and Dao, 2023]. This maps features $Batch \times Length \times Dimension \rightarrow Batch \times Length \times Dimension$. Here, $Batch$ is the minibatch size; $Length$ refers to the token sequence length, and $Dimension$ is the number of channels for each input token. For simplicity, we use $Dimension = 1$ by default and $Length$ matches the output dimension from the embedding learning layer (Figure 1). Although Mamba blocks can repeat $\mathcal{M}$ times, we set $\mathcal{M} = 1$ as our default value. However, we perform sensitivity study for $\mathcal{M} = 2, \cdots, 100$ with stacked Mamba blocks, which are connected with residual connections [He *et al.*, 2016], to evaluate their information retention or propagation capacity.

Inside a Mamba block, two fully-connected layers in two branches calculate linear projections (LP_1, LP_2). The first branch LP_1's output passes through a 1D causal convolution and SiLU activation $\mathcal{S}(\cdot)$ [Elfwing *et al.*, 2018], then a structured state space model (SSM). The continuous-time SSM is a system of first-order ordinary differential equation, which maps an input function or sequence $u(t)$ to output $y(t)$ through a latent state $h(t)$:

$$dh(t)/dt = A\,h(t) + B\,u(t), \quad x(t) = C\,h(t), \qquad (2)$$

where $h(t)$ is N-dimensional, with N also known as a state expansion factor, $u(t)$ is D-dimensional, with D being the $Dimension$ factor or the number of channels, $x(t)$ is usually taken as 1D, and A, B, and C are coefficient matrices of appropriate sizes. This dynamic system induces a discrete version governing state evolution and SSM outputs given the input token sequence via time sampling at $\{k\Delta\}$ with a Δ time interval. This discrete SSM version is a difference equation:

$$h_k = \bar{A}\,h_{k-1} + \bar{B}\,u_k, \quad x_k = C\,h_k, \qquad (3)$$

where h_k, u_k, and x_k are respectively samples of $h(t)$, $u(t)$, and $x(t)$ at time $k\Delta$, $\bar{A} = \exp(\Delta A)$, and $\bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I)\Delta B$. For SSMs, diagonal A is often used, and Mamba also makes B, C, and Δ linear time-varying functions dependent on the input. In particular, for an input token u, B and C are both $Linear_N(u)$, and Δ is $softplus(parameter + Linear_D(Linear_1(u)))$, with $Linear_p(u)$ being a linear projection to a p-dimensional space and $softplus$ activation function. With such time-varying coefficient matrices, the resulting SSM possesses context and input selectivity properties [Gu and Dao, 2023], facilitating Mamba blocks to selectively propagate or forget information along the potentially long input token sequence based on the current token. Subsequently, the SSM output is multiplicatively modulated with $\mathcal{S}(LP_2)$ before another fully connected projection. As a result, these integrated blocks empower MambaTab for content-dependent feature extraction and reasoning with long-range dependencies and feature interactions.

Output Prediction In this portion, our method learns representations from the concatenated Mamba blocks' output $\{x_k^i\}$ of shape $Batch \times Length \times Dimension$, where x_k^i is the k-th token output for example i in a minibatch. These are projected via a fully connected layer from $Batch \times Length \times Dimension \rightarrow Batch \times 1$, resulting in prediction logit y_i' for example i. With sigmoid activation,

$$sigmoid(y_i') = \frac{1}{1 + exp(-y_i')}, \tag{4}$$

we obtain the predicted probability score for calculating AUROC and binary-cross-entropy loss.

4 Experiments

4.1 Datasets, Implementation Details, and Baselines

Datasets To systematically evaluate the effectiveness of our method, we utilize 8 diverse public datasets. We provide the dataset's details and abbreviations in Table 1. Links to the datasets can be found in Supplementary Table 1. Our default experimental settings follow those of [Wang and Sun, 2022]. We split all datasets into train (70%), validation (10%), and test (20%) sets.

Implementation Details To keep the preprocessing simple, we follow the approach described in Section 3, generalizing for all datasets without manual intervention or tuning. Post training-validation, we take the best validation model and use it on the test set for prediction. We set up MambaTab with default hyperparameters and tuned potential hyperparameters for each dataset under vanilla supervised learning. For our default hyperparameters, we set up training for 1000 epochs with early stopping patience = 5. We adopt Adam optimizer [Kingma and Ba, 2014] and cosine-annealing learning rate scheduler with initial learning rate = $1e^{-4}$. In addition to training hyperparameters, MambarTab also involves other model-related hyperparameters and their default values are: embedded representation size ($Length$) = 32, SSM state expansion factor (N) = 32, local convolution width (d_conv) = 4, SSM block expansion factor ($\mathcal{M}$) = 1.

Baselines We extensively benchmark our model by comparing against standard and current state-of-the-art methods. These include: LR, XGBoost, MLP, SNN with SELU MLP, TabNet, DCN, AutoInt, TabTransformer, FT-Transformer, VIME, SCARF, and TransTab. More information about them can be found in Section 2. For fair comparison, we follow their architectures and implementation detailed in TransTab [Wang and Sun, 2022].

Performance Benchmark With default hyperparameters under vanilla supervised learning, our method denoted by MamabaTab-D achieves better performance than state-of-the-art baselines on many datasets and comparable performance on others with far fewer parameters (Table 4). After tuning hyperparameters, we denote our tuned model by MambaTab-T, whose performance further improves. Moreover, under feature incremental learning, our method substantially outperforms the existing method simply with default hyperparameters. We implement MambaTab in PyTorch and will release code upon acceptance. For evaluation, we use Area Under the Receiver Operating Characteristic (AUROC) following [Gorishniy *et al.*, 2021; Wang and Sun, 2022]. We obtain probability scores via Equation 4 on model output logits and use them and ground truth labels to calculate AUROC.

Table 1: Publicly available datasets with statistics (positive sample ratio, train, validation (val), test data points) and abbreviations used in this paper.

Dataset Name	Abbreviation	Datapoints	Train	Val	Test	Positive
Credit-g	CG	1000	700	100	200	0.70
Credit-approval	CA	690	483	69	138	0.56
Dresses-sales	DS	500	350	50	100	0.42
Adult	AD	48842	34189	4884	9769	0.24
Cylinder-bands	CB	540	378	54	108	0.58
Blastchar	BL	7043	4930	704	1409	0.27
Insurance-co	IO	5822	4075	582	1165	0.06
Income-1995	IC	32561	22792	3256	6513	0.24

4.2 Vanilla Supervised Learning Performance

For this setting, we follow the protocols from [Wang and Sun, 2022] directly using the training-validation sets for model learning and the test set for evaluation. To overcome potential sampling bias, we report average results over 10 runs with different random seeds on each of the 8 datasets. With defaults, MambaTab-D outperforms baselines on 3 public datasets (CG, CA, BL) and has comparable performance to transformer-based baselines on others. For example, MambaTab-D outperforms TransTab [Wang and Sun, 2022] on 5 out of 8 datasets (CG, CA, DS, CB, BL). After tuning hyperparameters, MambaTab-T achieves even better performance, outperforming all baselines on 6 datasets and achieving the second best on the other 2.

4.3 Feature Incremental Learning Performance

For this setting, we divide the feature set F of each dataset into three non-overlapping subsets s_1, s_2, s_3. set_1 contains s_1 features, set_2 contains s_1, s_2 features, and set_3 contains all features in s_1, s_2, s_3. While other baselines can only learn from either set_1 by dropping all incrementally added features (with respect to s_1) or set_3 by dropping old data, TransTab [Wang and Sun, 2022] and MambaTab can incrementally learn from set_1 to set_2 to set_3. In our method, we simply change the input feature cardinality $n(set_i)$ between settings, with the architecture fixed. Our method works because Mamba has strong content and context selectivity for extrapolation and we keep the representation space dimension fixed, that is, independent of feature set cardinality $n(F)$. Thus, this demonstrates the adaptability and simplicity of our method for incremental environments. Even with default hyperparameters, MambaTab-D outperforms all baselines as shown in Table 3. Here, we report the results averaged over 10 runs with different random seeds. Since it already achieves strong performance, we do

Table 2: Test AUROC results on 8 public datasets for vanilla supervised learning. Results reported here are averaged over 10 runs with random splits for our method. The best achieved results are shown in **bold** and the second best are shown in underlined.

Methods	Datasets							
	CG	CA	DS	AD	CB	BL	IO	IC
LR	0.720	0.836	0.557	0.851	0.748	0.801	0.769	0.860
XGBoost	0.726	0.895	0.587	0.912	0.892	0.821	0.758	**0.925**
MLP	0.643	0.832	0.568	0.904	0.613	0.832	0.779	0.893
SNN	0.641	0.880	0.540	0.902	0.621	0.834	0.794	0.892
TabNet	0.585	0.800	0.478	0.904	0.680	0.819	0.742	0.896
DCN	0.739	0.870	0.674	0.913	0.848	0.840	0.768	0.915
AutoInt	0.744	0.866	0.672	0.913	0.808	0.844	0.762	0.916
TabTrans	0.718	0.860	0.648	**0.914**	0.855	0.820	0.794	0.882
FT-Trans	0.739	0.859	0.657	0.913	0.862	0.841	0.793	0.915
VIME	0.735	0.852	0.485	0.912	0.769	0.837	0.786	0.908
SCARF	0.733	0.861	0.663	0.911	0.719	0.833	0.758	0.905
TransTab	0.768	0.881	0.643	0.907	0.851	0.845	**0.822**	0.919
MambaTab-D	0.771	0.954	0.643	0.906	0.862	0.852	0.785	0.906
MambaTab-T	**0.801**	**0.963**	**0.681**	**0.914**	**0.896**	**0.854**	0.812	0.920

not tune the hyperparameters further, although doing so could potentially improve performance.

4.4 Learnable Parameter Comparison

Our method not only achieves superior performance compared to existing state-of-the-art methods, it is also memory and space efficient. We demonstrate our method's superiority in terms of learnable parameter size while comparing against transformer-based approaches in Table 4. It is seen that our method (both Mambda-D and -T) achieves comparable or better performance than TransTab typically with $< 1\%$ of its learnable parameters. To evaluate learnable parameter size, we use the default settings specified in FT-Trans, TransTab, and TabTrans [1] [2]. We also notice that, despite varying features, TransTab's model size remains unchanged. The most important tunable hyperparameters for MambaTab include the block expansion factor (the local kernel size), the state expansion factor (N), and the embedded representation space dimension. We perform sensitivity analysis on them in Section 5 and also fine-tune them for each dataset. In addition, we conduct ablation study for the normalization layer of our model.

4.5 Hyperparameter Tuning

As mentioned above, we tune important hyperparameters and use the validation loss for tuning. Therefore, the test set is never used for tuning. The hyperparameters achieving the best validation loss are used for testing. We have reported averaged test results over 10 runs with different random seeds with the tuned MambaTab (Table 2). Learnable parameter sizes of MambaTab-T are reported in Table 4. Interestingly, MambaTab-T sometimes consumes fewer parameters than even MambaTab-D, e.g., on DS, BL, IO, and IC. We demonstrate key components of our tuned model MambaTab-T in Table 5, where the tuned values for these components are shown, with other training-related hyperparameters, such as batch size and learning rate, at default values; see Implementation Details in Section 4.

Table 3: Test AUROC results on 8 public datasets for feature incremental learning. Results reported here are averaged for 10 runs with random splits for our method MambaTab-D. The best results are shown in **bold**.

Methods	Datasets							
	CG	CA	DS	AD	CB	BL	IO	IC
LR	0.670	0.773	0.475	0.832	0.727	0.806	0.655	0.825
XGBoost	0.608	0.817	0.527	0.891	0.778	0.816	0.692	0.898
MLP	0.586	0.676	0.516	0.890	0.631	0.825	0.626	0.885
SNN	0.583	0.738	0.442	0.888	0.644	0.818	0.643	0.881
TabNet	0.573	0.689	0.419	0.886	0.571	0.837	0.680	0.882
DCN	0.674	0.835	0.578	0.893	0.778	0.840	0.660	0.891
AutoInt	0.671	0.825	0.563	0.893	0.769	0.836	0.676	0.887
TabTrans	0.653	0.732	0.584	0.856	0.784	0.792	0.674	0.828
FT-Trans	0.662	0.824	0.626	0.892	0.768	0.840	0.645	0.889
VIME	0.621	0.697	0.571	0.892	0.769	0.803	0.683	0.881
SCARF	0.651	0.753	0.556	0.891	0.703	0.829	0.680	0.887
TransTab	0.741	0.879	0.665	0.894	0.791	0.841	0.739	0.897
MambaTab-D	**0.787**	**0.961**	**0.669**	**0.904**	**0.860**	**0.853**	**0.783**	**0.908**

Table 4: Comparison of total learnable parameters between our method MambaTab and transformer-based methods. (M = million, K = thousand)

Methods	Datasets							
	CG	CA	DS	AD	CB	BL	IO	IC
TabTrans	2.7M	1.2M	2.0M	1.2M	6.5M	3.4M	87.0M	1.0M
FT-Trans	176K	176K	179K	178K	203K	176K	193K	177K
TransTab	4.2M	4.2M	4.2M	4.2M	4.2M	4.2M	4.2M	4.2M
MambaTab-D	**13K**	**13K**	13K	**13K**	**14K**	13K	15K	13K
MambaTab-T	50K	38K	**5K**	255K	30K	**11K**	**13K**	**10K**

5 Hyperparameter Sensitivity Analysis and Ablation Study

In this section, we demonstrate extensive sensitivity analyses and ablation experiments on MambaTab's most important hyperparameters using two randomly selected datasets: CylinderBands (CB) and Credit - g (CG). We measure performance by changing each factor, including block expansion factor, state expansion factor, and embedding representation space dimension, while keeping $\mathcal{M} = 1$ and other hyperparameters at default values as in MambaTab-D. We report results averaged over 10 runs with different random splits to overcome the potential bias due to randomness.

5.1 Block Expansion Factor

We experiment with block expansion factor (kernel size) $\{1, 2, ..., 10\}$, keeping the other hyperparameters at default values as in MambaTab-D. As seen in Figure 3, MambaTab's performance changes only slightly with different block ex-

[1] https://github.com/lucidrains/tab-transformer-pytorch

[2] https://github.com/RyanWangZf/transtab

Table 5: Hyperparameters of our tuned model, MambaTab-T. The performance of MambaTab-T for vanilla supervised learning has been shown in Table 2.

Hyperparameters	Datasets							
	CG	CA	DS	AD	CB	BL	IO	IC
Embedding Representation Space	64	32	16	64	32	16	16	32
State Expansion Factor	16	64	32	64	8	4	8	64
Block Expansion Factor	3	4	2	10	7	10	9	1

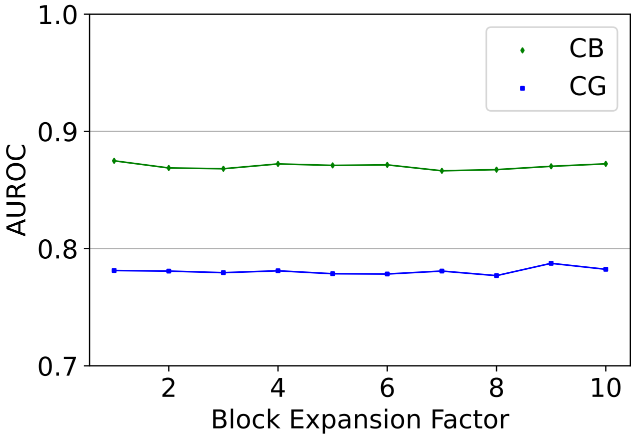


Figure 3: Sensitivity of MambaTab AUROC to block expansion factor on two randomly chosen datasets CB and CG.

pansion factors, with no clear or monotonic trends. Thus we set the default to 2, inspired by [Gu and Dao, 2023], though tuning this parameter further could improve performance on some datasets.

5.2 State Expansion Factor

We demonstrate the effect of the state expansion factor (N) using values in $\{4, 8, 16, 32, 64, 128\}$. As seen in Figure 4, MambaTab performance in AUROC improves with increasing state expansion factor for both datasets CG and CB. Thus tuning this hyperparameter could further improve performance. However, a larger state expansion factor consumes more memory. To balance performance versus memory consumption, we select 32 as the default value.

5.3 Size of Embedded Representations

As mentioned in the Method section, we allow flexibility for the model to learn the embedding via a fully connected layer. We also perform sensitivity analysis for the length of the embedded representations, with values in $\{4, 8, 16, 32, 64, 128\}$. As seen in Figure 5, MambaTab's performance essentially increases for both CG and CB datasets with larger embedding sizes, though at the cost of more parameters and thus larger CPU/GPU space. To balance performance versus model size, we choose to keep the default embedding length to 32.

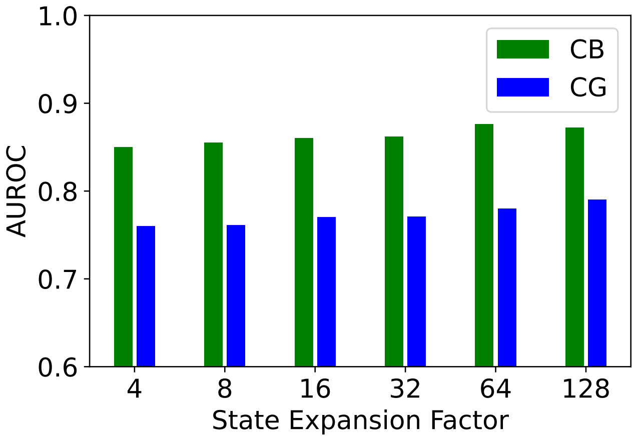


Figure 4: Sensitivity of MambaTab AUROC to state expansion factor on two randomly chosen datasets CB and CG.

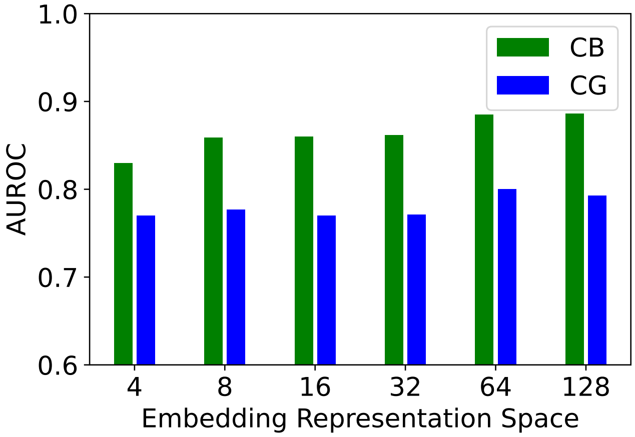


Figure 5: Sensitivity of MambaTab performance in AUROC to size of embedded representations on two datasets CB and CG.

5.4 Ablation of Layer Normalization

We demonstrate the effect of layer normalization, which is applied to the embedded representations, in our model architecture shown in Figure 1. We contrast the performance by keeping or dropping this layer in vanilla supervised learning experiments on CG and CB datasets. The results in AUROC metric are shown in Table 6. We can see the effectiveness of the normalization layer. Without layer normalization, the embeddings would directly pass through the ReLU activation, as shown in the overall scheme (Figure 1). On both CG and CB datasets, MambaTab's performance improves with layer normalization versus without. This ablation thus justifies the incorporation of layer normalization in our model.

5.5 Effect of Batch Size

In addition to the above model-related hyperparameters, we also perform sensitivity analysis with experiments on batch size. Due to its small model size, MambaTab can easily handle a large number of samples per batch. We demonstrate AUROC results in Figure 6 using sizes of $\{60, 80, 100, 120, 140\}$.

Table 6: Ablation analysis of layer normalization. Experiments are conducted with and without layer normalization under supervised learning using the model architecture shown in Figure 1. Results demonstrated are on test set AUROC.

Ablation	Datasets	
	CG	CB
Without Layer Normalization	0.759	0.847
With Layer Normalization	**0.771**	**0.862**

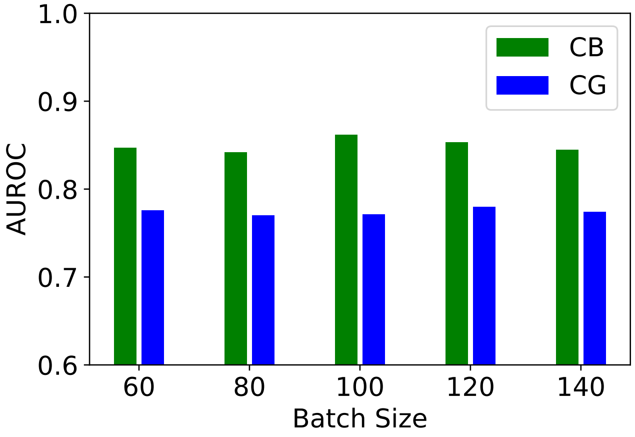


Figure 6: Sensitivity of MambaTab performance in AUROC to batch size. Other parameters are kept at default values while the batch size is varied from 60 to 140 in steps of 20.

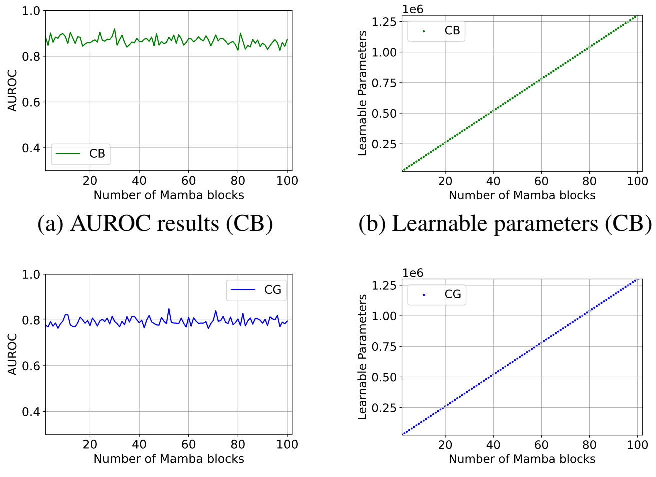
We see small variations in performance for both CG and CB datasets across minibatch sizes. This demonstrates our method's generalization capability regarding batch size. Considering this insensitivity, we set 100 as the default batch size.

5.6 Scaling Mamba

Although we have achieved comparable or superior performance to current state-of-the-art methods with a default $\mathcal{M} = 1$ under regular supervised learning (see Table 2), we also study the effect of scaling Mamba blocks via residual connections following [He *et al.*, 2016]. We stack Mamba blocks as in Figure 1, concatenating $\mathcal{M} = 2$ up to 100 blocks, as shown in Equation 5:

$$h^{(i)} = Mamba_i(h^{(i-1)}) + h^{(i-1)}. \tag{5}$$

Here, $h^{(i)}$ is the hidden state from the i-th Mamba block, that is, $Mamba_i$, taking the prior block's hidden state $h^{(i-1)}$ as input. As seen in Figure 7, with increasing Mamba blocks, MambaTab retains comparable performance while the learnable parameters increase linearly on both CG and CB datasets. This demonstrates Mamba block's information retention capacity. We observe that few Mamba blocks suffice for strong performance. Hence, we choose to use $\mathcal{M} = 1$ by default in MambaTab.



(a) AUROC results (CB)

(b) Learnable parameters (CB)

(c) AUROC results (CG)

(d) Learnable parameters (CG)

Figure 7: AUROC results and learnable parameter sizes versus number of stacked residual Mamba blocks ($\mathcal{M}$) on CB and CG datasets. Other parameters are kept at default values of MambaTab-D.

6 Future Scope

Although we have evaluated our method on tabular datasets for classification, in the future we would like to incorporate our method for regression tasks as well on tabular data. Our method is flexible enough to incorporate regression tasks since we have kept the output layer open to predict real values. Therefore, our future research scope includes but is not limited to evaluating performance on different learning tasks.

7 Conclusion

This paper presents Mambatab, a simple yet effective method for handling tabular data. It uses Mamba, a state-space-model variant, as a building block to classify tabular data. MambaTab can effectively learn and predict in both vanilla supervised learning and feature incremental learning settings. Moreover, it requires no manual data preprocessing. MambaTab demonstrates superior performance over current state-of-the-art deep learning and traditional machine learning-based baselines under both supervised and incremental learning on 8 public datasets. Remarkably, MambaTab occupies only a small fraction of memory in learnable parameter size compared to Transformer-based baselines for tabular data. Extensive results demonstrate MambaTab's efficacy, efficiency, and generalizability for diverse tabular learning applications.

Acknowledgments

We thank the creators of the public datasets used, for making these valuable resources openly available for research purposes. We also thank the authors of the baseline models and Mamba [Gu and Dao, 2023] for releasing their valuable code publicly. This research is supported in part by the NSF under Grant IIS 2327113 and the NIH under Grants R21AG070909, P30AG072946, and R01HD101508-01.

References

[Agarap, 2018] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[Arik and Pfister, 2021] Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021.

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Bahri *et al.*, 2021] Dara Bahri, Heinrich Jiang, Yi Tay, and Donald Metzler. Scarf: Self-supervised contrastive learning using random feature corruption. *arXiv preprint arXiv:2106.15147*, 2021.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 22:785–794, 2016.

[Elfwing *et al.*, 2018] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. Special issue on deep reinforcement learning.

[Fu *et al.*, 2022] Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. Hungry hungry hippos: Towards language modeling with state space models. In *International Conference on Learning Representations*, 2022.

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

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

[Gu *et al.*, 2021a] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

[Gu *et al.*, 2021b] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in Neural Information Processing Systems*, 34:572–585, 2021.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

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

[Ioffe and Szegedy, 2015] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *International Conference on Machine Learning*, pages 448–456, 2015.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Klambauer *et al.*, 2017] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, 30:972– 981, 2017.

[Song *et al.*, 2019] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. *ACM International Conference on Information and Knowledge Management*, pages 1161–1170, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

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

[Wang *et al.*, 2017] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. *Proceedings of the ADKDD'17*, 2017.

[Yoon *et al.*, 2020] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020.

[Zhang *et al.*, 2020] Yixuan Zhang, Jialiang Tong, Ziyi Wang, and Fengqiang Gao. Customer transaction fraud detection using xgboost model. *International Conference on Computer Engineering and Application (ICCEA)*, pages 554–558, 2020.

Supplementary Materials

Dataset details with accessible links

Here in this section, we provide the dataset accessible links with their abbreviations.

Supplementary Table 1: Publicly available datasets abbreviations and external accessible link

Dataset Name	Abbreviation	External link
Credit-g	CG	openml.org/search?type=data&status=active&id=31
Credit-approval	CA	archive.ics.uci.edu/ml/datasets/credit+approval
Dresses-sales	DS	openml.org/search?type=data&status=active&id=23381
Adult	AD	openml.org/search?type=data&status=active&id=1590
Cylinder-bands	CB	openml.org/search?type=data&status=active&id=6332
Blastchar	BL	kaggle.com/datasets/blastchar/telco-customer-churn
Insurance-co	IO	archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+%28COIL+2000%29
Income-1995	IC	kaggle.com/datasets/lodetomasi1995/income-classification

Instructions on how to run the code

The following instructions are helpful to run our codes. The instructions are written in a sequential manner.

Installation

Please install the following required libraries:

- pip install torch==2.1.1 torchvision==0.16.1
- pip install causal-conv1d==1.1.1
- pip install mamba-ssm

File instruction and brief definition

- **config.py**: contains training related configurations setting.
- **MambaTab.py**: contains our method-related code of MambaTab.
- **supervised_mambatab.py**: contains code related to vanilla supervised learning setting.
- **feature_incremental.py**: contains code related to incremental feature setting.
- **train_val.py**: contains training and validation related code for training and validating the model over the epochs.
- **utility.py**: contains code for data reading and data preprocessing.

Data download and processing

Please download data using the accessible links mentioned in the supplementary materials table 1.
Cautions:

- Dataset must be in *.csv* format.
- Header row must be in the first row in the *.csv*
- Target column must be in the last column in the *.csv*
- Rename the *X.csv* to *data_processed.csv* and place it into *datasets/X* folder. Here 'X' can be 'dress', 'cylinder', etc.

Configurations

Please utilize our **config.py** file for the necessary configurations to run the code. Moreover, more configurations can also be modified in our **MambaTab.py** file for model-related configurations.

Running specific files

- To run vanilla supervised learning, please execute **supervised_mambatab.py**. Please make sure to change the necessary configurations in **config.py** before running this file. We have mentioned the necessary comments for an easier understanding of the flow of the code.
- To run the feature incremental learning setting, please execute **feature_incremental.py**. Here, as mentioned in our paper, we divide the features into 3 non-overlapping sets and perform the training incrementally.
- As mentioned above, our code is flexible in changing parameters and getting results with tuned settings.