# ConTextTab: A Semantics-Aware Tabular In-Context Learner

**Marco Spinaci** [*1] **Marek Polewczyk** [*2] **Maximilian Schambach** [2] **Sam Thelin** [2]

## Abstract

Tabular in-context learning (ICL) models such as TabPFN and TabICL have recently achieved state-of-the-art (SOTA) performance on several tabular prediction tasks. Trained exclusively on synthetic data, these models however do not fully leverage the rich semantics and world knowledge contained in real-world data. Tabular ICL models based on pretrained large language models such as TabuLa-8B integrate semantics and world knowledge but are only able to make use of a small amount of context due to inherent architectural limitations. Aiming to bridge this gap, we introduce **ConTextTab**, integrating semantic understanding and alignment into a table-native ICL framework. Using specialized embeddings for different data modalities and training on large-scale real-world tabular data, our model is competitive with SOTA across a broad set of benchmarks while setting a new standard on the semantically rich CARTE benchmark. Code and checkpoints: https://github.com/SAP-samples/contexttab.

## 1. Introduction

Tables remains a predominant data format in many real-world applications (Chui et al., 2018), making its understanding through machine learning algorithms critical. Recently, applying the in-context learning (ICL) paradigm to tabular prediction tasks has shown promising results. The approach was pioneered by TabPFN (Hollmann et al., 2023). Pretrained on large amounts of synthetic data, its latest incarnation TabPFNv2 (Hollmann et al., 2025) produces SOTA results on datasets with up to 10 000 rows for both classification and regression tasks. In recent work, TabICL (Qu et al., 2025) extends the success story of this approach to larger datasets, using special tabular embedding modules. Common to both TabPFN and TabICL is that they are trained entirely on synthetically generated numerical data, with categorical features produced by an indexing procedure. While using synthetic data has many advantages, in particular its diversity at scale, a consequence is that the data does not contain any semantically meaningful values as found in real-world applications. In particular, column names are not used in either TabPFN or TabICL, and categorical features are encoded via ordinal encoding disregarding any underlying semantics. We argue that semantic understanding can be captured by training on real-world data. A primary example of this is TabuLa-8B (Gardner et al., 2024), turning a pretrained LLM into a tabular ICL model. However, utilizing pretrained LLMs for tabular tasks has several limitations. Most importantly, textual serialization and tokenization of the input table is not token efficient, effectively limiting the maximum context length that can be processed. For example, TabuLa-8B operates on a maximum of 32 context rows. Furthermore, the tokenization schema and autoregressive nature of LLMs are not adapted to the tabular structure, resulting in a linear non-uniform token sequence.

Aiming to bridge these approaches, we propose a table-native ICL model trained on real-world data, using embeddings tailored to different data modalities, in particular incorporating semantic embeddings of column names and categorical values. The resulting model is competitive to existing table-native ICL approaches across a range of tabular prediction benchmarks and achieves a new SOTA for the semantically rich CARTE benchmark (Kim et al., 2024), in particular in the low-data regime.

## 2. Related Work

**Tabular deep learning:** Prediction on tabular data has traditionally been dominated by decision trees, particularly boosted variants. They deliver strong performance but require separate training for each task and cannot leverage pretraining. Recent deep learning approaches have successfully shown consistently good performance overtaking boosted trees, for example RealMLP (Holzmüller et al., 2024).

**In-context learning on tabular data:** TabPFN broke the long-standing dominance of boosted trees on small classification tasks, outperforming them by using row-level ICL. A recent variant, TabDPT (Ma et al., 2024), showed that equally excellent results can be achieved by training on

[*]Equal contribution [1]SAP France [2]SAP SE. Correspondence to: Marco Spinaci <marco.spinaci@sap.com>, Marek Polewczyk <marek.polewczyk@sap.com>.

*Proceedings of the $42^{nd}$ International Conference on Machine Learning*, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

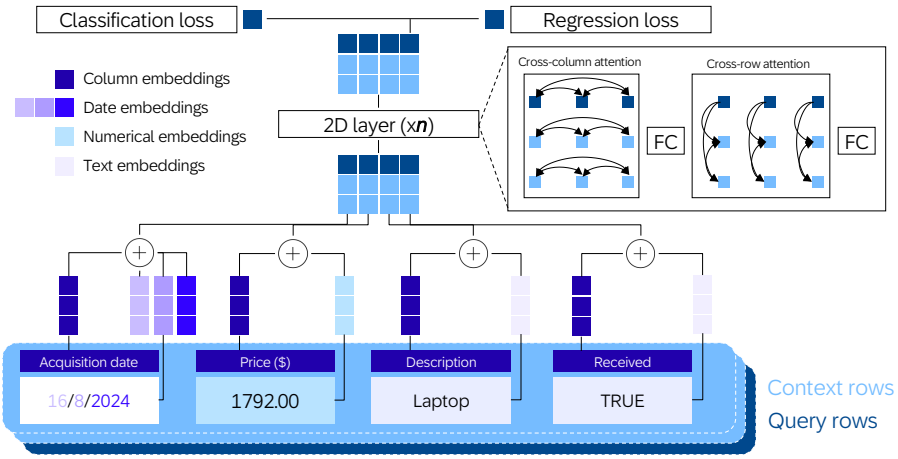



*Figure 1.* Our proposed model architecture illustrating the integration of data type-specific embeddings, an interleaved attention backbone, and customized output heads.

real-world data using similarity-based retrieval of context rows. Generalizing the row-based encoding, cell-based ICL introduced with TabPFNv2[1] and utilized also by TabICL extended this success to larger datasets with up to 10 000 and more samples.

**Semantics and real data:** Capturing the rich semantics of real-world tabular data enables the transfer of world knowledge across prediction tasks in addition to statistical patterns. The CARTE (Kim et al., 2024) architecture enables pretraining across a range of real-world sources while capturing semantics. It achieves SOTA results on the semantically rich CARTE benchmark, although it requires task-specific fine-tuning. Several works approach tabular ICL by tuning LLMs on tabular tasks, e.g. TabLLM (Hegselmann et al., 2023), LIFT (Dinh et al., 2022), or TabuLa-8B. In particular, the works by Gardner et al. (2024) are note-worthy for curating the T4 dataset, containing roughly 3 M tables, and for its excellent results in the very low data-regime.

## 3. Method

To overcome the aforementioned limitations of existing table-native ICL methods and bridge the gap to LLM-based ones, we propose ConTextTab, a semantics-aware table-native ICL model. To this end, we perform several key modifications to the TabPFN architecture and utilize large-scale pretraining on real-world data. An overview of our proposed architecture is given in Figure 1.

### 3.1. Encoding

We encode data differently depending on its modality – i.e. text, date, or numeric type. Column headers are also encoded, effectively replacing positional encodings.

[1]Hereinafter, we focus on TabPFNv2 and refer to it as TabPFN.

**Text:** We transform each text cell to an embedding vector using a pretrained text embedding model. We apply this to both free text as well as categorical columns, which can thus retain the meaning in their labels. Any off-the-shelf embedding model can be used for this purpose. We settle for the comparably small but fast model `all-MiniLM-L6-v2` (Wang et al., 2020). We apply a learnable linear layer to the text embeddings to map to dimension $d$. Semantic embeddings of cell values have been previously investigated (Yak et al., 2023; Ye et al., 2024a; Kim et al., 2024; Spinaci et al., 2024), however, details on how they are used differ in each implementation.

**Date:** We embed day, month, and year values separately and sum the three resulting latent vectors.

**Numerical:** As numbers do not contain semantic meaning beyond their value, we apply a one-dimensional encoding. During training, we clip columns between the 2% and 98% quantiles of their distribution and scale them to have zero mean and unit variance. This avoids exploding gradients during training. Finally, the resulting number is multiplied by a learnable vector and a bias is added. If the original value was NaN, 0 is used instead.

**Column headers:** We embed column headers with the same model used for text cells. The result is passed through a separate learnable linear layer to map to the correct target dimension and summed with the cell embedding. Note that all the above embeddings are fully equivariant under permutations of either rows or columns – a property often desired (Van Breugel & Van Der Schaar, 2024).

### 3.2. Backbone

We leave the TabPFN architecture mostly unchanged, stacking $n$ layers with alternating cross-column and cross-row

self-attention. Following TabPFN, cross-column attention has no masking, while cross-row attention is masked so that each row can only attend to the provided context. To increase the modularity, the feedforward MLP block of the transformer encoder is repeated after each self-attention, so that horizontal and vertical blocks have the same weights.

Model weights can also be optionally shared between each instance of the transformer block, which consists of two interleaved attention layers. Such an architecture can be interpreted as a recurrent neural network but unrolled in depth rather than in time. This results in high parameter efficiency. Empirically, we observed that sharing weights did not affect model performance and therefore use weight sharing as the default option for our model in the following.

### 3.3. Decoding

**Classification:** We apply a standard cross-entropy loss after an MLP. However, this imposes two limitations. First, the number of classes seen in pretraining cannot be exceeded at inference without resorting to extensions such as hierarchical classification as used in TabPFN's many-class extension. Secondly, it prevents us from using the semantic value of the class label. For that purpose, we create an additional special input encoding, only for the target column, in addition to the ones previously described. Despite breaking equivariance under permutation, we found this approach to be effective.

**Regression:** The model predicts the float value of the target, clipped and normalized as previously described. Empirically, we have found this simple schema to work well. During training, an $L_2$ loss is applied. At inference, the inverse normalization is applied to the prediction.

## 4. Experimental Setup

### 4.1. Training and inference

For pretraining, we use the T4 dataset (Gardner et al., 2024). We discard tables with fewer than 150 rows, which leaves 2.18 M tables with a median of 750 rows and 9 columns. We randomly select 1000 rows, then between 50 and 900 rows as query, and use the rest as context. Then we randomly select one target column aiming to equal probability of classification and regression. We train each model for between 2 to 5 epochs, until convergence. We use a micro batch size of 1 and accumulate gradients to simulate a batch size of 256. To improve stability, we employ gradient clipping and the AdamW optimizer with a maximum learning rate of $10^{-4}$, reached after a linear warm-up phase of 1000 gradient updates. We set $n = 12$ and $d = 768$, and train on a single H100 GPU, reaching a throughput of roughly 10 tables/sec.[2] Further experiments with longer context and incorporating data from Ma et al. (2024) did not lead to measurable improvements.

Like TabPFN, we employ bagging at inference time. From the original train split of a given evaluation dataset, we sample 8 times $c$ context rows with replacement, make a separate prediction with each collection of $c$ rows using the same model, and average the predictions (regression values or classification probabilities). We typically set $c = 8192$, which is much larger than the training context size.

### 4.2. Evaluation

**Datasets:** We use a variety of tabular prediction datasets to evaluate and compare our approach to established baselines and other SOTA methods. Namely, we evaluate all models on the following benchmarks: OpenML-CC18 (Bischl et al., 2021), a pure classification benchmark; OpenML-CTR23 (Fischer et al., 2023), a pure regression benchmark; TALENT (Ye et al., 2024b), a recent diverse benchmark containing over 300 classification and regression tasks. Here, we focus on a subset containing 45 datasets that are representative of the overall performance of the baselines investigated in the original works, which we refer to as the TALENT-Tiny benchmark; TabReD (Rubachev et al., 2025), a small but challenging benchmark of large datasets; and finally CARTE (Kim et al., 2024), a mixed classification and regression benchmark containing highly semantic features and few numerical ones.

Across all benchmarks, we evaluate a total of 203 tasks, 91 regression and 112 classification. Due to the large number of evaluated datasets, we do *not* perform cross-validation but evaluate a fixed test split for each task. We refer to Appendix B.2 for dataset statistics and further details.

**Baselines:** We compare our approach to several established classical methods as well as recent ICL and other deep learning models. Namely, we compare with TabPFN, TabICL, TabDPT, CARTE, RealMLP, XGBoost (Chen & Guestrin, 2016), LightBGM (Ke et al., 2017), CatBoost (Prokhorenkova et al., 2018), random forest, histogram-based gradient boosting tree, KNN, linear, and a naive estimator from `scikit-learn` (Pedregosa et al., 2011). For all pretrained models, we use the latest available release and checkpoints as of May 2025. For the boosted tree baselines and RealMLP, we use the `pytabkit` wrapper and evaluate tuned-default (TD) as well as hyperparameter optimized (HPO) variants (Holzmüller et al., 2024). As the gold standard in tabular prediction, we also evaluate AutoGluon (Erickson et al., 2020). We refer to Appendix B.1 for full details on the used baseline.

[2] In total, this would make the model have 172M parameters, but due to weight sharing there are only 16M trainable parameters.

*Table 1.* Performance comparison across all evaluated benchmarks, depicting mean accuracy (Acc) and $R^2$ score in percent. For a full comparison with all evaluated models we refer to Appendix A.

| Model Name | All | CARTE | | | OML-CC18 | | OML-CTR23 | | TabReD | | | TALENT-Tiny | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Rank | Rank | Acc | R2 | Rank | Acc | Rank | R2 | Rank | Acc | R2 | Rank | Acc | R2 |
| AutoGluon | – | – | 78.7 | 73.8 | – | 88.5 | – | 67.0 | – | 86.0 | 64.6 | – | 87.9 | 73.7 |
| **ConTextTab** | **1.61** | **1.16** | **76.9** | **72.2** | 1.71 | 86.8 | 2.06 | 72.9 | 2.38 | 85.4 | 63.4 | 1.43 | **87.7** | **76.1** |
| TabPFN | 1.86 | 2.59 | 72.4 | 65.0 | **1.67** | 87.0 | **1.63** | **74.9** | 2.00 | 85.6 | 63.8 | **1.41** | 87.3 | 75.1 |
| CatBoost [HPO] | 2.37 | 2.22 | 75.6 | 66.1 | 2.44 | 86.7 | 2.71 | -44.4 | 1.5 | 85.8 | 63.7 | 2.30 | 86.0 | 72.1 |
| RealMLP [TD] | 2.42 | 3.55 | 70.2 | 59.6 | 1.97 | **87.2** | 2.06 | 45.7 | **1.12** | **86.0** | **64.6** | 2.35 | 86.2 | 71.5 |
| Naive | 4.84 | 5.00 | 53.0 | -1.8 | 4.81 | 47.0 | 4.89 | -8.5 | 4.50 | 80.8 | -0.6 | 4.70 | 53.4 | -22.3 |

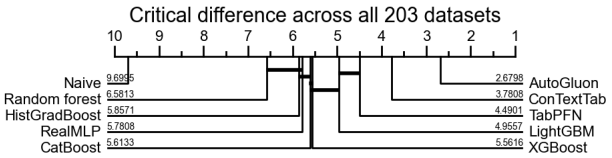



*Figure 2.* Critical difference diagram between ConTextTab and several baselines, across all 203 evaluated datasets.

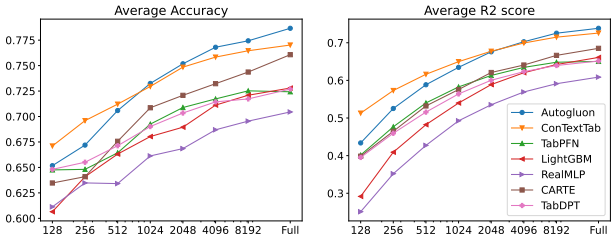



*Figure 3.* Average accuracy, and regression results on CARTE benchmark across various data subsets.

**Metrics:** We evaluate accuracy and $R^2$ for classification and regression tasks, respectively. As averaging across a large number of datasets with varying performance can blur relative performance across models, we also evaluate mean rank. For each benchmark, we calculate the mean across all constituent datasets and models. Full rank performance is averaged across all datasets across all benchmarks. To reduce the impact of noise, we count model performances as ties when their evaluation lie within 0.005 of each other.

## 5. Results

The main results are shown in Table 1 with extended results presented in Appendix A.1. Here, we are excluding AutoGluon from the overall rank and best-model comparison as it is ensembling and stacking a multitude of model architectures, making it difficult to highlight architectural strengths and weaknesses. Overall, our model achieves the highest rank across all evaluated datasets. This performance gain is statistically significant, as depicted in the critical difference diagram shown in Figure 2. Our model performs particularly well for the semantically rich CARTE benchmark, where TabPFN is outperformed by boosted trees and even non-tuned CatBoost, highlighting the importance of incorporating semantic understanding into tabular ICL. On the other end, while the absolute performance of ConTextTab for large datasets, e.g. within the TabReD benchmark, is decent, it falls behind tuned boosted trees and RealMLP here, which achieve SOTA on par with AutoGluon.

To investigate the dependence on dataset size, we depict the performance on the CARTE benchmark across varying subsampled sizes in Figure 3, ranging from 128 rows to the full dataset. ConTextTab consistently outperforms other models across all sample sizes and even surpasses AutoGluon for up to 2048 training samples. This highlights the strong capabilities of tabular ICL but also the need for further research into scaling these architectures to effectively deal with much larger context sizes as well as training datasets. We investigate this more closely in Appendix A.2.

## 6. Limitations and future work

While achieving SOTA results across the investigated datasets, we observe several limitations of our proposed approach. One drawback of using real-world data for training is the possibility of contamination, e.g. the presence of evaluation tasks in the training corpus. For the CARTE benchmark, being our focus, we have conducted a contamination study, using the column name and cell value embeddings created by a text embedder and matching similarities. We did not find any contamination of CARTE in T4. For contamination of OpenML datasets, we refer to the original study by Gardner et al. (2024). Either way, as a single table is only seen a few times during training, we believe that memorization is likely not a practical problem when training on real-world data. Generally, all investigated table-native ICL models fail to scale their performance to very large datasets. Increasing the context length did not resolve these issues. In the large-data case, conventional methods, in particular when stacked via AutoGluon, still

perform best. Overcoming this remains one of the major challenges for tabular foundation models.

Overall, we show that incorporating semantics in table-native models is beneficial but also that more semantically rich data is required – both at scale for training models with longer context, as well as curated for evaluation.

## Acknowledgements

We would like to thank Johannes Hoehne, Johannes Hoffart, and Markus Kohler for their insightful comments and suggestions throughout the development of this work, which have greatly contributed to shaping its direction and quality.

## References

Bischl, B., Casalicchio, G., Feurer, M., Gijsbers, P., Hutter, F., Lang, M., Mantovani, R. G., van Rijn, J. N., and Vanschoren, J. OpenML benchmarking suites. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, 2016. ISBN 978-1-4503-4232-2.

Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., and Malhotra, S. Notes from the AI frontier: Insights from hundreds of use cases. *McKinsey Global Institute*, pp. 28, 2018.

Dinh, T., Zeng, Y., Zhang, R., Lin, Z., Gira, M., Rajput, S., Sohn, J.-y., Papailiopoulos, D., and Lee, K. LIFT: Language-interfaced fine-tuning for non-language machine learning tasks. *Advances in Neural Information Processing Systems*, 35:11763–11784, 2022.

Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M., and Smola, A. AutoGluon-Tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Fischer, S. F., Feurer, M., and Bischl, B. OpenML-CTR23 – A curated tabular regression benchmarking suite. In *AutoML Conference 2023 (Workshop)*, 2023.

Gardner, J. P., Perdomo, J. C., and Schmidt, L. Large scale transfer learning for tabular data via language modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on typical tabular data? In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 507–520, 2022.

Hegselmann, S., Buendia, A., Lang, H., Agrawal, M., Jiang, X., and Sontag, D. TabLLM: Few-shot classification of tabular data with large language models. In *International Conference on Artificial Intelligence and Statistics*, pp. 5549–5581. PMLR, 2023.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Holzmüller, D., Grinsztajn, L., and Steinwart, I. Better by default: Strong pre-tuned MLPs and boosted trees on tabular data. *Advances in Neural Information Processing Systems*, 37:26577–26658, 2024.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, 2017.

Kim, M. J., Grinsztajn, L., and Varoquaux, G. CARTE: Pretraining and transfer for tabular learning. In *Forty-first International Conference on Machine Learning*, 2024.

Ma, J., Thomas, V., Hosseinzadeh, R., Kamkari, H., Labach, A., Cresswell, J. C., Golestan, K., Yu, G., Volkovs, M., and Caterini, A. L. TabDPT: Scaling tabular foundation models. *arXiv preprint arXiv:2410.18164*, 2024.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in python. *the Journal of Machine Learning Research*, 12:2825–2830, 2011.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: Unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31, 2018.

Qu, J., Holzmüller, D., Varoquaux, G., and Morvan, M. L. TabICL: A tabular foundation model for in-context learning on large data. *arXiv preprint arXiv:2502.05564*, 2025.

Rubachev, I., Kartashev, N., Gorishniy, Y., and Babenko, A. TabReD: A benchmark of tabular machine learning in-the-wild. In *International Conference on Learning Representations*, 2025.

Spinaci, M., Polewczyk, M., Hoffart, J., Kohler, M. C., Thelin, S., and Klein, T. PORTAL: Scalable tabular foundation models via content-specific tokenization. In *NeurIPS 2024 Third Table Representation Learning Workshop*, 2024.

Van Breugel, B. and Van Der Schaar, M. Why tabular foundation models should be a research priority. In *Proceedings of the 40th International Conference on Machine Learning*, 2024.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

Yak, S., Dong, Y., Gonzalvo, J., and Arik, S. IngesTables: Scalable and efficient training of LLM-enabled tabular foundation models. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

Ye, C., Lu, G., Wang, H., Li, L., Wu, S., Chen, G., and Zhao, J. Towards cross-table masked pretraining for web data mining. In *Proceedings of the ACM on Web Conference 2024*, pp. 4449–4459, 2024a.

Ye, H.-J., Liu, S.-Y., Cai, H.-R., Zhou, Q.-L., and Zhan, D.-C. A closer look at deep learning methods on tabular datasets. *arXiv preprint arXiv:2407.00956*, 2024b.

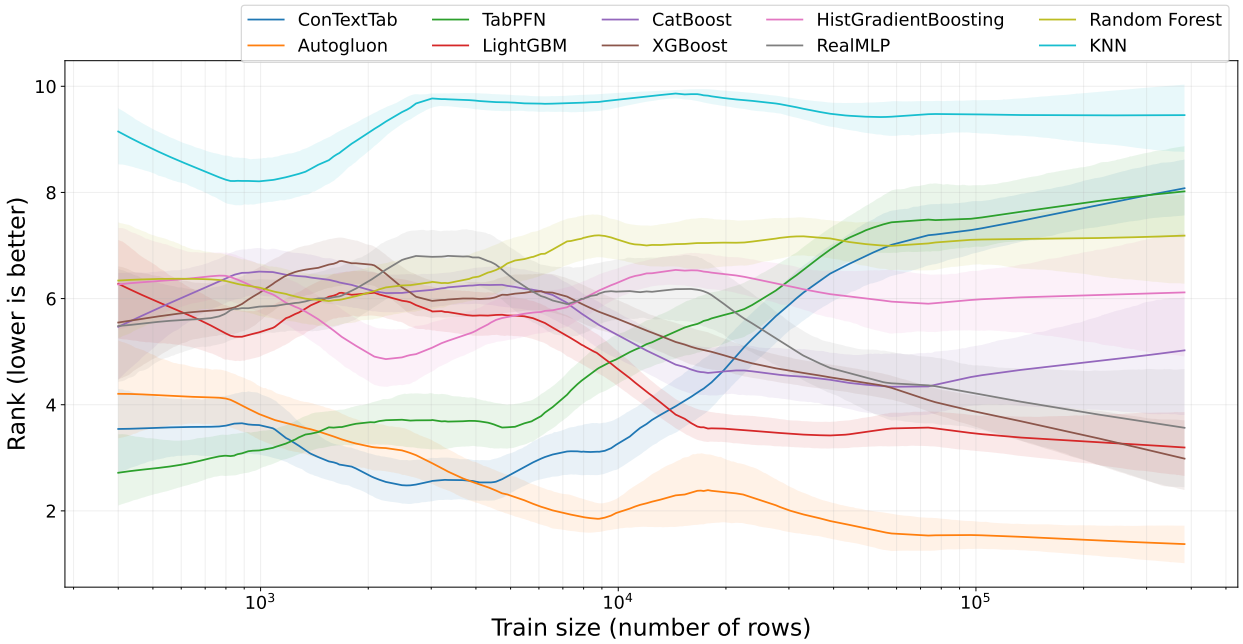



*Figure 4.* Relation between number of training dataset rows and performance, obtained as a LOWESS regression in the plane $\log(n_{\text{rows}}, \text{rank})$. The confidence bands are the 80% confidence intervals obtained via bootstrapping.

# A. Further Results

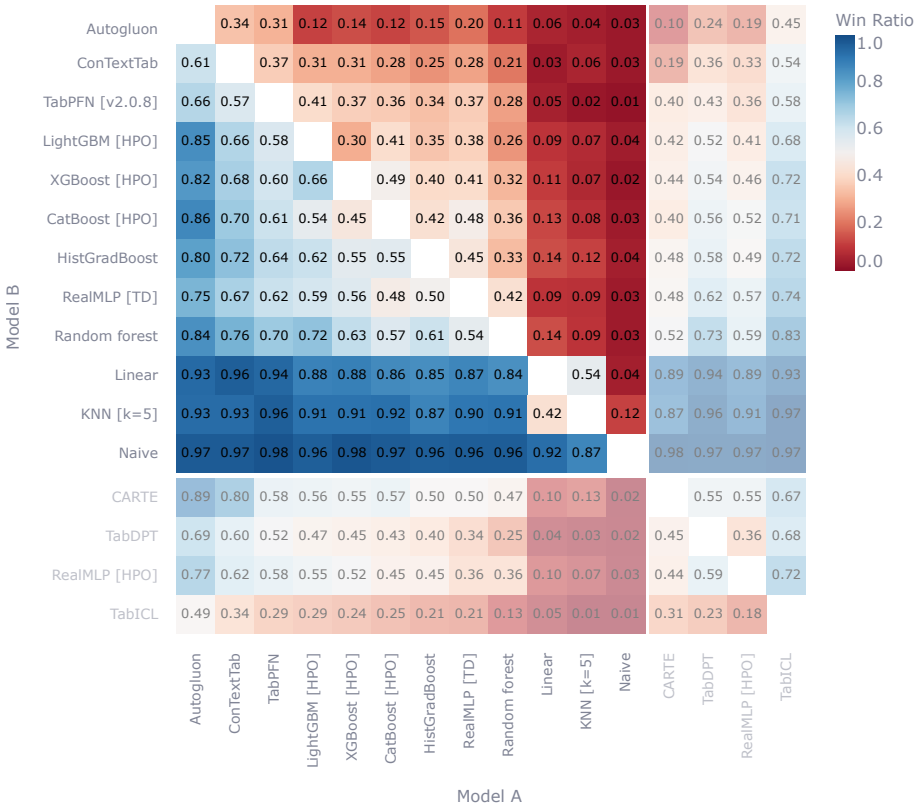
## A.1. Extended results

Including further baselines not depicted in the main paper, Table 2 shows all collected evaluations across the investigated benchmarks. Furthermore, we show the win ratio confusion matrix and averages of the investigated models in Figure 5.

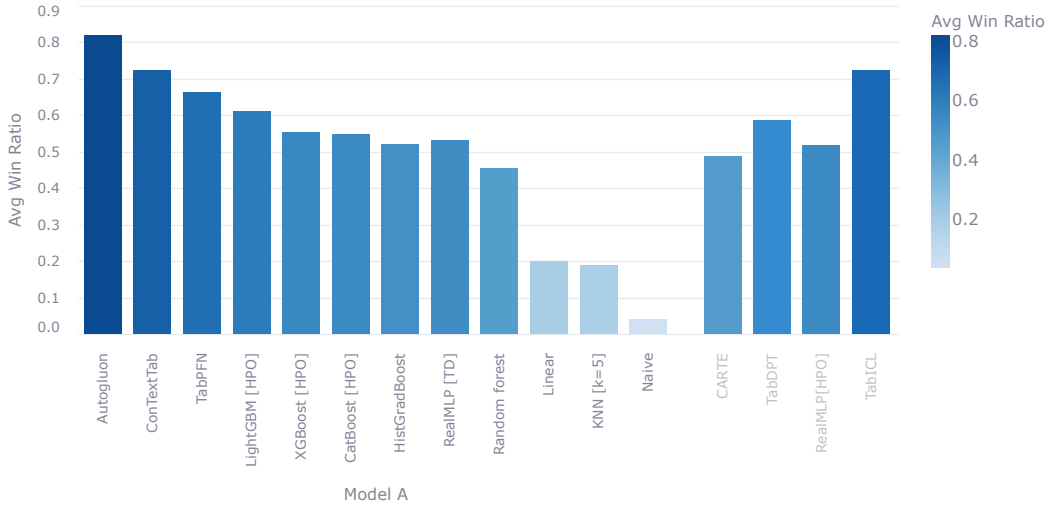
## A.2. Relation between dataset size and model performance

In addition to Figure 3, we depict average rank of the experiment previously shown in Figure 6.

Furthermore, we plot the average rank of each model as a function of the dataset size (expressed in number of rows) across all evaluated datasets in Figure 4. Note that TabICL is missing from this comparison as it only handles classification tasks. We can observe that, as expected, ConTextTab and TabPFN excel in the low data regime, while AutoGluon takes the lead when enough data is available. In the very low data regime below 1000 training rows, TabPFN performs best, however our model surpasses its performance for larger datasets with more than 1000 rows. Overall, ConTextTab remains competitive with AutoGluon until roughly 10 000 rows, possibly as a result of the limitation in both the inference context size and the context size seen during training. After 10 000 rows, gradient boosting methods, as well as RealMLP, start surpassing the performance of ConTextTab as well as TabPFN. Overall, this indicates the need of further research for tabular ICL to handle larger amounts of training data, but also the availability of more diverse benchmarks, covering larger datasets, as the current evaluation is dominated by datasets with less than 10 k rows and less than 100 columns, see Figure 7.




(a) Win ratio matrix with Model A wins over Model B.




(b) Average Win ratio of Model A against all others.

*Figure 5.* Win ratio confusion matrix and average of the investigated models across all 203 datasets. Wins are calculated based on accuracy on classification and $R^2$ on regression datasets. Ties are not counted as wins. Models are sorted by descending overall rank. Note that CARTE, TabDPT, RealMLP [HPO], and TabICL were not successfully evaluated on all datasets and only compared on evaluated ones, resulting in a smaller support and skewing their win rates. Therefore, they are separately grouped last, not used in averaging, and only shown for completeness.

*Table 2.* Performance comparison across all evaluated benchmarks, depicting mean accuracy (Acc) for classification and $R^2$ score for regression tasks, in percent. Missing values, due to architectural limitations or failed evaluations, are denotes as N/A and excluded from the mean rank calculation.

| **Model Name** | **All** | **CARTE** | | | **OML-CC18** | | **OML-CTR23** | | **TabReD** | | | **TALENT-Tiny** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Rank** | **Rank** | **Acc** | **R2** | **Rank** | **Acc** | **Rank** | **R2** | **Rank** | **Acc** | **R2** | **Rank** | **Acc** | **R2** |
| Autogluon | **2.48** | **1.55** | **78.7** | **73.8** | **2.61** | 88.5 | 4.06 | 67.0 | 1.75 | **86.0** | **64.6** | **2.16** | **87.9** | 73.7 |
| **ConTextTab** | 3.53 | 2.24 | 76.9 | 72.2 | 4.08 | 86.8 | 4.37 | 72.9 | 4.62 | 85.4 | 63.4 | 3.22 | 87.7 | **76.1** |
| TabPFN | 4.27 | 7.25 | 72.4 | 65.0 | 3.61 | 87.0 | 3.43 | **74.9** | 4.5 | 85.6 | 63.8 | **2.16** | 87.3 | 75.1 |
| LightGBM [HPO] | 4.59 | 5.41 | 72.8 | 66.1 | 4.56 | 86.9 | 4.63 | 61.9 | 1.75 | 85.9 | 64.4 | 4.11 | 86.4 | 72.4 |
| CatBoost [HPO] | 4.99 | 5.33 | 75.6 | 66.1 | 4.6 | 86.7 | 6.63 | -44.4 | 2.12 | 85.8 | 63.7 | 4.32 | 86.0 | 72.1 |
| CatBoost [TD] | 4.99 | 6.35 | 75.1 | 65.6 | 4.96 | 86.4 | 4.51 | 67.2 | 1.25 | 85.9 | 64.3 | 4.43 | 85.9 | 74.8 |
| LightGBM [TD] | 5.05 | 7.02 | 73.1 | 64.9 | 4.01 | 86.8 | 6.29 | 39.7 | 2.25 | 85.7 | 63.2 | 3.81 | 86.3 | 72.6 |
| XGBoost [HPO] | 5.05 | 5.84 | 72.6 | 65.7 | 5.74 | 86.3 | 4.34 | 71.4 | 1.75 | **86.0** | 64.2 | 4.03 | 86.2 | 72.6 |
| XGBoost [TD] | 5.52 | 8.24 | 72.3 | 64.4 | 4.03 | 87.0 | 6.54 | 63.7 | 2.5 | 85.6 | 62.8 | 4.35 | 86.0 | 72.7 |
| HistGradBoost | 5.81 | 7.04 | 72.5 | 64.8 | 5.11 | 86.1 | 7.14 | 51.2 | 2.25 | 85.9 | 63.9 | 5.0 | 86.3 | 67.6 |
| RealMLP [TD] | 6.52 | 12.2 | 70.2 | 59.6 | 4.62 | 87.2 | 5.14 | 45.7 | **1.12** | **86.0** | **64.6** | 4.86 | 86.2 | 71.5 |
| Random forest | 7.59 | 9.82 | 71.5 | 63.3 | 6.57 | 85.7 | 7.77 | 53.6 | 7.25 | 85.4 | 60.7 | 6.38 | 85.8 | 70.6 |
| Linear | 13.46 | 16.55 | 62.7 | 19.1 | 11.17 | 80.9 | 14.2 | 43.8 | 12.12 | 80.8 | -4269.6 | 13.27 | 80.5 | 41.3 |
| KNN [k=5] | 13.78 | 16.25 | 65.5 | 34.3 | 12.06 | 81.7 | 13.57 | 14.1 | 13.5 | 78.7 | -15.4 | 14.0 | 80.3 | 60.0 |
| Naive | 16.35 | 18.04 | 53.0 | -1.8 | 15.28 | 47.0 | 16.4 | -8.5 | 12.62 | 80.8 | -0.6 | 16.89 | 53.4 | -22.3 |
| CARTE | N/A | 4.65 | 76.1 | 68.5 | N/A | N/A | N/A | N/A | N/A | N/A | N/A | 7.38 | 84.4 | 71.1 |
| TabDPT | N/A | 6.63 | 72.7 | 65.1 | 3.74 | 87.8 | **3.26** | 72.4 | N/A | N/A | 60.9 | 3.73 | 86.7 | 74.8 |
| RealMLP [HPO] | N/A | 11.78 | 70.4 | 60.9 | 4.08 | 87.3 | 4.51 | 53.2 | N/A | N/A | N/A | 4.0 | 86.3 | 72.1 |
| TabICL | N/A | N/A | 72.5 | N/A | N/A | **88.8** | N/A | N/A | N/A | 85.1 | N/A | N/A | 87.4 | N/A |

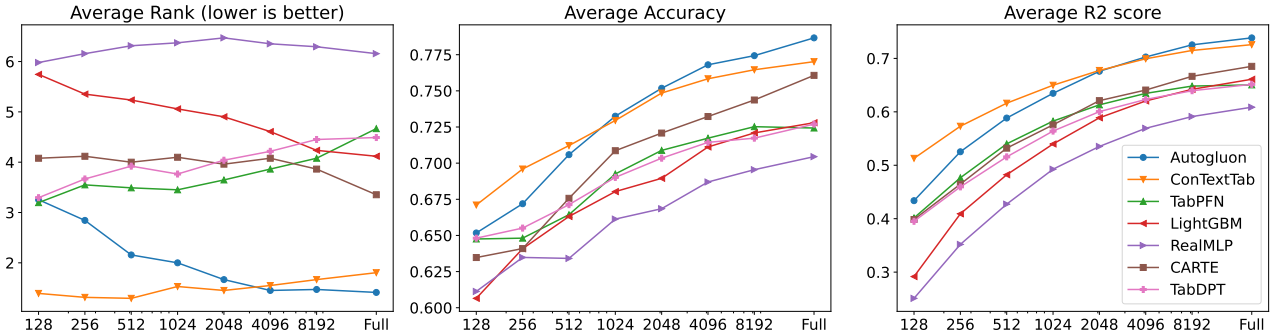



*Figure 6.* Average rank, accuracy, and regression results on CARTE benchmark across various data subsets.

# B. Experimental Setup – Details

## B.1. Baselines

**Data preprocessor:** Evaluating models across a multitude of datasets can be tricky. Datasets may have inconsistent data type annotations, such as categories represented as strings or categorical data types, covering low- and high-cardinality categorical features, date, time, or datetime instances, free text, boolean values and more. Most models, however, require numerical input to process and handle non-numerical data types or missing values differently. To unify our evaluation, we implemented a configurable default feature encoder built on the `AutoMLPipelineFeatureGenerator` from AutoGluon (Erickson et al., 2020) which we found to be very versatile and robust. In particular, the encoder natively handles low- and high-cardinality categorical data, free text (to some extent), as well as datetime encoding. For flexibility and compatibility with a multitude of models, we extended the default implementation to cover the following options that can be adapted to the capabilities of the baseline model at hand:

- Whether to convert booleans to string/categorical values
- Whether to *not* encode string/categorical values for models that natively handle them, such as CatBoost
- Whether to scale numerical data via quantile scaling with a normal distribution as target
- Whether to drop constant features
- Whether to impute missing values, extending the standard imputation (using most frequent categories and mean for numerical data) to bools and datetime data types

As deault, we choose to convert booleans, encode categoricals via ordinal encoding, scale numerical data, drop constant columns and impute missing values.

Below, we describe for which baselines the default values are changed or when other types of feature encodings are used.

**TabPFN:** We use the model from the official Python `tabpfn` package with version 2.0.8 together with the `tabpfn-extensions` package version 0.1.0 at commit `d44606e35f89e18b6bc4c4a2eef2f46918c4302e` of the Git repository[3] as the PyPi release is not up-to-date.

Naturally, we use `TabPFNClassifier` and `TabPFNRegressor` for classification and regression tasks, respectively, using default parameters for both. In particular, `TabPFNClassifier` uses an ensemble of 4 and `TabPFNRegressor` an ensemble of 8 estimators. We combine the classification estimator with the `ManyClassClassifier` extension with a redundancy factor of 4 to enable classification beyond the native 10-class limit of TabPFN which is required for the evaluation of some of the 203 evaluated datasets.

For datasets larger than the native 10 k limit of TabPFN, we sample a random 10 k subset of the training split. This affects 66 out of the 203 evaluated datasets. For datasets with more than the 500 feature limit that TabPFN was trained with, we select a random subsample of 500 features. This affects 12 out of 203 evaluated datasets. While this is not optimal, and post-hoc ensembling as well as a random forest preprocessing is recommended by the authors (Hollmann et al., 2025), these extensions cannot be combined with the many-class extension required to predict beyond the 10-class limit of the native TabPFN model. Hence, we cannot evaluate TabPFN with the post-hoc ensembling or random forest extension.

As we found the native feature encoder of TabPFN to not work across all evaluated datasets, we use our standard feature encoder (see above), encoding categorical columns, scaling numerical values, dropping constant columns, and imputing missing values. As this procedure should be very similar to the TabPFN-native encoder, we anticipate this deviation to affect the results only insignificantly if at all.

**TabICL:** We use the latest model weights `tabicl-classifier-v1.1-0506.ckpt` from the recent 0.1.2 version of the official `tabicl` package. This updated variant is an improved checkpoint over the one reported in the original works (Qu et al., 2025). As TabICL solely supports classification tasks, we exclude it from the overall mean rank evaluation. For encoding, we use our default encoder, but do not scale numericals, do not drop constant values, and do not impute missing ones as it is natively handled by the model.

[3] https://github.com/PriorLabs/tabpfn-extensions.git

**TabDPT:** We use the model from the official GitHub repository[4] at the most recent 1.1.0 release and `tabdpt1_1.pth` model checkpoint. Naturally, we use `TabDPTClassifier` and `TabDPTRegressor` for classification and regression tasks, respectively, using default parameters for both. Throughout, we evaluate the model with a (local) context size of 2048 which is the best-performing one in the original works (Ma et al., 2024). However, evaluation failed for some datasets due to an error in the original code leading to empty predictions for very large datasets in the TabReD benchmark. Here, we use our default encoder, scaling numericals, dropping constant values, and imputing missing ones.

**CARTE:** We use the model provided in the official Python `carte-ai` package with version 0.0.26. We use `CARTEClassifier` and `CARTERegressor` with default parameters for classification and regression tasks, respectively We treat binary classification tasks as 2-class multi-class classification and hence set `loss=``categorical_crossentropy''` for the classifacation estimator. With CARTE, we use our default preprocessor to convert bool values and datetime instances and to impute missing values, but otherwise rely on the `Table2GraphTransformer` provided in the reference implementation.

**Pytabkit models:** We use the `pytabkit` implementation wrapper for evaluating RealMLP, XGBoost, LightGBM, and CatBoost. We evaluate all models both in the tuned-defaults (TD) variant proposed by Holzmüller et al. (2024) as well as hyperparameter-optimized (HPO). However, the HPO version did run into OOM issues for RealMLP, even when running on a H100 with 96 GB of VRAM which is why we display the TD variant by default.

In particular, for RealMLP (TD), we use `RealMLP_TD_Classifier` and `RealMLP_TD_Regressor` for classification and regression tasks, respectively. For RealMLP (HPO), we use `RealMLP_HPO_Classifier` and `RealMLP_HPO_Regressor` for classification and regression tasks, respectively, conducting the default 50 rounds of random search HPO.

For XGBoost (TD), LightGBM (TD), and CatBoost (TD), we use `XGB_TD_Classifier`, `XGB_TD_Regressor`, `LGBM_TD_Classifier`, `LGBM_TD_Regressor`, `CatBoost_TD_Classifier`, and `CatBoost_TD_Regressor` for classification and regression tasks, respectively. For the HPO-variants, we use the `HPO_TPE` versions of the estimators, performing Parzen-tree based HPO with 50 rounds using the search space as defined by (Grinsztajn et al., 2022). LightGBM and XGBoost are evaluated on 8-core CPU machines with 64 GB of RAM, whereas CatBoost and RealMLP are evaluated on H100 GPUs with 96 GB of VRAM. Note that CatBoost evaluation on CPU was too slow to evaluate at scale, in particular in the HPO variant. However, there are known issues with the GPU implementation of CatBoost which might degrade performance[5]. We were not able to observe systematically worse results on those datasets on which we were also able to evaluate the CPU variant. Hence, for consistency, we present results for the GPU variant throughout.

Throughout, we use our default encoder, scaling numericals, dropping constant values, and imputing missing ones. For all models but CatBoost, we perform ordinal encoding of categoricals.

**Sklearn models:** We use several standard baseline models from `scikit-learn` (Pedregosa et al., 2011), combining them with the default preprocessor as outlined above. Across all `scikit-learn` baselines, preprocessing only differs in missing value imputation, depending on the model's capability to handle missing values natively. Throughout, evaluation is performed using `scikit-learn` v1.5.2.

For the naive predictor, we use the `DummyClassifier` and `DummyRegressor` to predict the most frequent, respectively mean value of the train splits as the naive majority baseline.

For the linear predictor, we use the `LogisticRegression` and `LinearRegression` for classification and regression tasks, respectively, using default hyperparameters.

For the KNN predictor, we use the `KNeighborsClassifier` and `KNeighborsRegressor` for classification and regression tasks, respectively, using default hyperparameters and $k = 5$ nearest neighbors.

For the random forest predictor, we use the `RandomForestClassifier` and `RandomForestRegressor` for classification and regression tasks, respectively, using default hyperparameters. The model handles missing values natively.

Finally, for the histogram-based gradient boosted tree predictor, we use the `HistGradientBoostingClassifier` and `HistGradientBoostingRegressor` for classification and regression tasks, respectively, using default hyperparameters. The model handles missing values natively.

[4] https://github.com/layer6ai-labs/TabDPT.git

[5] See https://github.com/catboost/catboost/issues/1408.

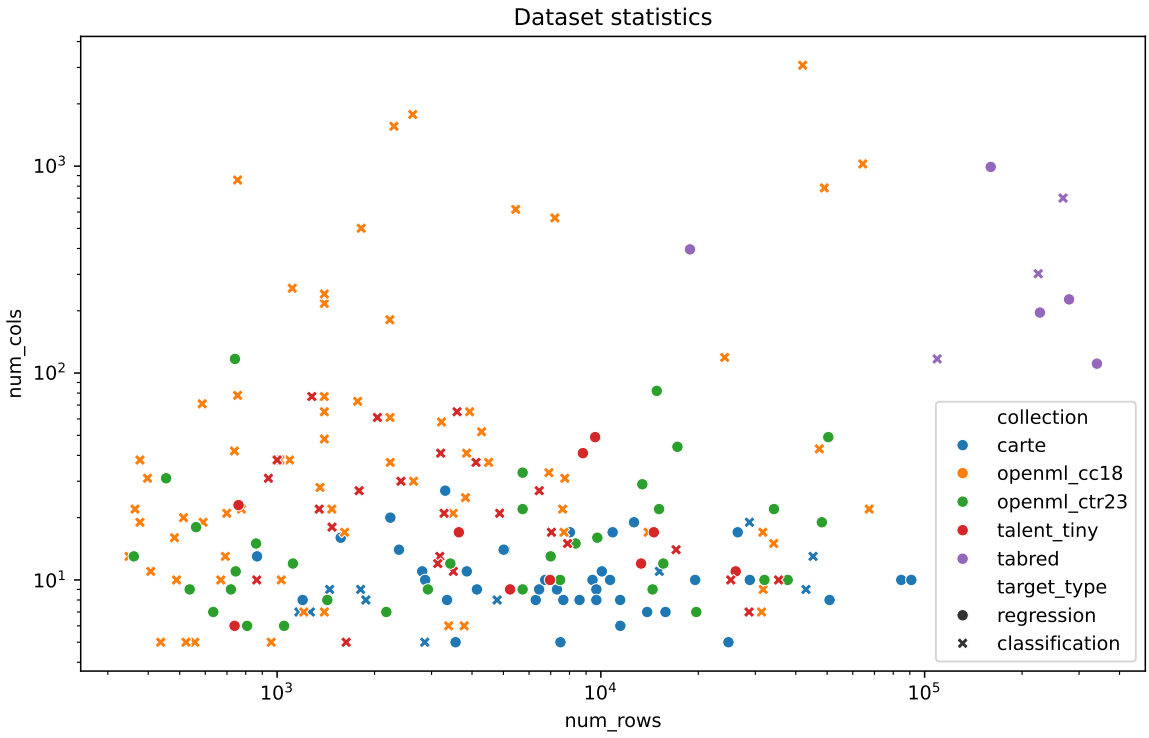



*Figure 7.* Column and row distribution of the evaluated benchmark datasets.

**AutoGluon:** Throughout, we use AutoGluon v1.2 and its `TabularPredictor` without custom preprocessing. We use the `best_quality` preset and set a per-dataset time limit of 4 h. Otherwise, parameters are left at their default values. For all datasets, evaluation is executed on a single 16-core machine with 128 GB of RAM and no GPU.

### B.2. Datasets

The row and column count statistics of the evaluation datasets are visualized in Figure 7.

We extracted all datasets from their original source and performed a custom stratified train-validation-test split with a 70-10-20 ratio. For classification tasks, the target column is used for stratification. For regression tasks, we perform stratification on the binned target column, binning it into 5 quantiles using the `qcut` method from the `pandas` library. Otherwise, we do not perform any alterations on the data. Models not using a specific validation procedure are provided with the concatenated train and validation split for training.