# CARTE: Pretraining and Transfer for Tabular Learning

**Myung Jun Kim** [1] **Léo Grinsztajn** [1] **Gaël Varoquaux** [1 2]

## Abstract

Pretrained deep-learning models are the go-to solution for images or text. However, for tabular data the standard is still to train tree-based models. Indeed, transfer learning on tables hits the challenge of *data integration*: finding correspondences, correspondences in the entries (*entity matching*) where different words may denote the same entity, correspondences across columns (*schema matching*), which may come in different orders, names... We propose a neural architecture that does not need such correspondences. As a result, we can pretrain it on background data that has not been matched. The architecture –CARTE for Context Aware Representation of Table Entries– uses a graph representation of tabular (or relational) data to process tables with different columns, string embedding of entries and columns names to model an open vocabulary, and a graph-attentional network to contextualize entries with column names and neighboring entries. An extensive benchmark shows that CARTE facilitates learning, outperforming a solid set of baselines including the best tree-based models. CARTE also enables joint learning across tables with unmatched columns, enhancing a small table with bigger ones. CARTE opens the door to large pretrained models for tabular data.

## 1. Introduction

The wide availability of pre-trained models has greatly facilitated machine learning for various data modalities, for example with images (Simonyan & Zisserman, 2015) or texts (Devlin et al., 2019). These models can be downloaded from model hubs, embarking a lot of implicit information and transformations that unleashes the power of deep learning even on small datasets. This paradigm has led to the revolution of foundation models (Bommasani et al., 2021) such as large language models (Touvron et al., 2023). But this revolution has not happened for tabular data despite its huge importance for enterprise and institutional data. One roadblock is that integrating data from tables in the wild is often difficult, sometimes impossible. Different tables might not have any related data and when they do, data integration is a whole field of database research (Doan et al., 2012). One might need to solve correspondence across columns –*schema matching*– or across data sources with different naming conventions for entries –*entity matching*. For lack of matching schemas and entities, pretraining across tables in the wild has not been possible. Without pretraining, deep learning is less practical, and tree-based methods are often preferable (Grinsztajn et al., 2022).

Here we introduce a learning architecture that learns across tables without schema and string matching. The key is to represent tables with a graph and all symbols with embeddings (for column names and table entries). The architecture, dubbed CARTE (Context-Aware Representation of Table Entries), is pretrained on a large knowledge base, to capture information on a vast amount of entities and relations. It can then be fine-tuned on a given downstream task, helping learning even in few shot settings. It can also be used for joint learning across multiple tables, enriching a target table with weakly related sources. CARTE brings a sizable performance gain, outperforming markedly a set of 42 solid baselines (including the best tree-based methods and various feature engineering). It benefits particularly tables with string entries, frequent in applications but seldom present in machine learning benchmarks.

Section 2 presents related work; section 3 describes the CARTE architecture and training procedures; and section 4 provides an extensive empirical study across many tabular datasets, benchmarking the settings of a single downstream table as well as multiple related ones.

## 2. Related Works

**Tabular deep learning** Tables are central to many applications. As a result, numerous deep learning methods tailored for this modality have been proposed (Abutbul et al., 2020; Arik & Pfister, 2020; Popov et al., 2019; Gorishniy et al., 2023b; Somepalli et al., 2021). However, they typically

[1]SODA Team, Inria Saclay, France [2]Probabl.ai, France. Correspondence to: Myung Jun Kim <myung.kim@inria.fr>.

*Proceedings of the 41st International Conference on Machine Learning*, Vienna, Austria. PMLR 235, 2024. 

under-perform tree-based methods (Grinsztajn et al., 2022; Shwartz-Ziv & Armon, 2021; Gardner et al., 2022). While McElfresh et al. (2023) argue that neural networks perform well on certain types of tables, and promising architectures are continuously published (Gorishniy et al., 2023a; Chen et al., 2023), the difficulty of improving over tree-based methods suggests that deep learning must bring something more to the fight, such as background knowledge.

**Transfer learning for tabular data** Transfer learning mostly focuses on the "conventional" settings of transferring across datasets with the same features, *i.e.* columns. Somepalli et al. (2021) demonstrates pre-training on a larger unlabeled version of the table, while Levin et al. (2023) argues that transfer learning bridges the gap between deep learning and tree-based models when there are few data points but large related datasets, as in their medical setting. They consider new or missing features in the downstream table, but require exact matching between most features.

XTab (Zhu et al., 2023) can work on tables with different columns using data-specific featurizers that map instances to the same dimension followed by federated learning on the common block. However, they did not outperform tree-based models (CatBoost, Dorogush et al., 2018). Transtab (Wang & Sun, 2022) also vectorizes each row of tables into an embedding space to learn across tables, demonstrating data accumulation across multiple clinical trials to outperform baselines including XGBoost (Chen & Guestrin, 2016). These approach benefit from a pool of tables in the subtopic, but it is not clear if they can be adapted to build pretrained models for a wide set of applications.

**Pretrained models for tabular data** TabPFN (Hollmann et al., 2023) made headway in pre-training models for tabular learning: it uses a transformer model pre-trained on large amounts of synthetic data to capture the inductive biases of tabular data, leading to strong performance on small datasets, though it has no dedicated handling for categorical columns, a challenge of tables where trees historically shine. Large language models (LLMs) can also work as pretrained models for tabular data. In TabLLM (Hegselmann et al., 2023), tabular data are represented as a set of tokens which are leveraged to fine-tune an LLM. However, the difficulty of handling numerical values in LLMs makes them a suboptimal choice compared to trees or TabPFN.

**Discrete entries** One challenge of tables –much more tackled by the database literature than the machine learning one– is that many of the entries are discrete, represented as strings. Cerda & Varoquaux (2022) created string-based representations that facilitate learning. The `TableVectorizer` in Skrub (2024) uses these heuristically to turn columns of different types into numerical matrices well suited for learning. KEN (Cvetkov-Iliev et al., 2023) is another approach to embed table entities closer to our goals of pretraining. It provides embeddings of all entities in a knowledge graph, capturing the information in a source such as Wikipedia. These embeddings facilitate learning, but the challenge is that each entry of a column must be linked to a Wikipedia entry, an *entity matching* task.

**Data integration** Traditional statistical models need data assembled in a single consistent table, a task tackled by the data integration literature (Doan et al., 2012). Finding correspondences between columns across data sources is known as *schema matching*. *Entity matching* is a challenge common to data integration and natural language processing (NLP), where a string must be linked to an *entity*: a unique concept. For instance "Davinci" may denote the historical figure "Leonardo da Vinci", but also OpenAI "Text-Davinci-003" GPT3 API. Entity matching must be robust to string variations, but most importantly it must account for the context in which an entity appears to disambiguate potential matches. In NLP, pretrained attention-based models, such as BERT (Devlin et al., 2019), have been crucial to capture the corresponding context.

These pretrained language models also are useful on tables to automate data normalization and integration tasks with few manually-supplied examples (Narayan et al., 2022). Deep learning, and recently attention-based models, is progressing on tasks to structure databases, *e.g.,* column typing, entity linking (Hulsebos et al., 2019; Deng et al., 2020).

We are interested in a different problem: rather than explicit matching, at the entity or schema level, we aim to capture only implicit data structure and integration to enhance downstream machine learning task without any manual operation such as finding related sources. The problem is timely: researchers sharing this vision are assembling large corpora of tables (Hulsebos et al., 2023; Eggert et al., 2023). Yet, the small scale of most tables available and the variability between tabular datasets has so far made this vision elusive.

## 3. The CARTE Model to Learn Across Tables

The ability of CARTE to learn across tables stems from the combination of two elements: a novel representation of table entities with graphs and a deep neural network architecture that captures the context that reside within a table. In particular, the former endows synchronization of multiple tables to the same graph domain, which makes pretraining on formerly unmatched background data possible. Moreover, the context-aware deep neural network trained on broad spectrum of knowledge can readily spread the background information to downstream tasks at our hands. In this section, we introduce CARTE with detailed implementations.

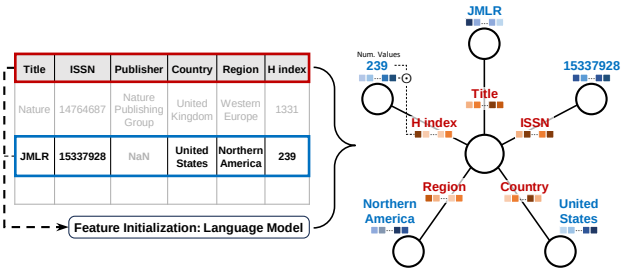


*Figure 1.* **Graphlet representation of tabular entities.** From a table, CARTE represents each row as a star-like graph. Excluding for missing values, the leaf-nodes and the edges are annotated by the cell values and their corresponding column names. Then, CARTE initializes the features of each with a language model. The nodes of numerical values are initialized by the elementwise product with its corresponding column feature. For the center node, it is initially set as the average of the leaflets. It will later act as a readout that captures the overall information of the graphlet.

### 3.1. Graph Representation of Table Entities

The graph representation is crucial for facilitating the generalization of table entities. In general, a graph, $G$, consists of nodes and edges, where the former denote the entities and the latter denote the relations between the nodes. Graphs are useful for capturing relational information between entities, and graph deep learning is promising for relational databases (Fey et al., 2023). CARTE considers each of the row as a small graph, as shown in Figure 1. From a table with $k$ columns, CARTE represents each of the $i$-th instance as a graph, $G_i(X, E)$, where the components $X$ and $E$ denote the node and edge features, respectively, embeddings in $\mathsf{R}^d$. The structure of $G_i(X, E)$ is a star-like graph with $k \quad p_i$ leaf-nodes, with $p_i$ as the number of columns with missing values for row $i$. On the resulting graphlet, each of the leaf-nodes are annotated by the cell values and their corresponding column names. To make these graphlets viable inputs for neural networks, we initialize $X$ and $E$ by using a language model. For categorical values and column names, CARTE simply places a $d$-dimensional embedding that is generated from a language model. For numerical values, the features are initialized by the product of its value with the embedding of the corresponding column name. For instance, the node feature $X_{(239)}$ in Figure 1 is equal to $239 \quad E_{(\text{H index})}$. Lastly, the center node is initialized with the mean of the leaflets, and will later serve as the readout node that captures the overall information of the graph.

This design serves several purposes. First it represents context: for tabular data, an entry is best interpreted accounting for its column's name. In Figure 1, for example, it would be difficult to grasp the row (blue-box) only with the entries of 'JMLR', '15337928', '239'. The column names 'Title', 'ISSN', and 'H index' clarifies that it is an instance of a journal. CARTE represents context in tables through the nodes and edge, exposed to its neural network architecture. This representation also bridges tables with different column order, or more generally different columns.

Second, CARTE uses language models on non-numerical entries, such as strings, categories, and names. Thus, the graph transformation in CARTE does not require any intervention on discrete entries, as opposed to typical data preprocessing or cleaning (deduplication, categorical encodings) used on strings. Moreover, CARTE works with an open set of vocabularies. Problems of typos or wordings of the same meaning, such as 'North America' to 'Northern America', are readily resolved for CARTE.

Together, these features of the proposed graph representation enable generalization across heterogeneous tables. CARTE's graph transformation puts in the same graph domain instances from different tables, without requiring any schema matching for columns or entity matching for entries. Thus, learning process can operate across many tables, which opens the door for pretraining or transfer.

### 3.2. Pretrained Model from a Large Knowledge Base

CARTE is pretrained on YAGO3 (Mahdisoltani et al., 2013), a large knowledge base built from Wikidata and other sources that contain facts about real-world. YAGO stores information as a knowledge graph, which is a collection of triplets *(head, relation, tail)*. For instance, the triplet *(Louvre, is located in, Paris)* from Figure 2 would be a sample that we can find in YAGO. Our current version of YAGO contains over 18.1 million triplets of 6.3 million entities.

In this subsection, we describe the pretraining process of CARTE, summarized in Figure 2. From the knowledge graph, we first extract small graphlets of entities suitable as inputs for CARTE. For self-supervised learning with a contrastive loss, we add to the batch truncated versions of selected graphlets. Through the training process, CARTE learns to aggregate information based on the given context.

**Graphlets for pretraining** From the large knowledge graph of YAGO, we construct small graphlets of the entities that can be used as inputs for CARTE. To construct a suitable graphlet for an entity, we first extract its subgraph within a user-specified $k$-hop relations. To resemble the structure outlined in Figure 1 while benefiting from additional information through multiple hops, we set $k = 2$, but restricting the maximum number of 1-hop and 2-hop relations to 100 and 10 respectively. Graphlets from tables (Figure 1) have as center node a token for the row, while the knowledge-graph procedure could use the entity name (for instance 'Louvre'). To avoid a difference, we use a token as a center node with an additional neighbor which is comprised of the name as its node and 'has name' as its

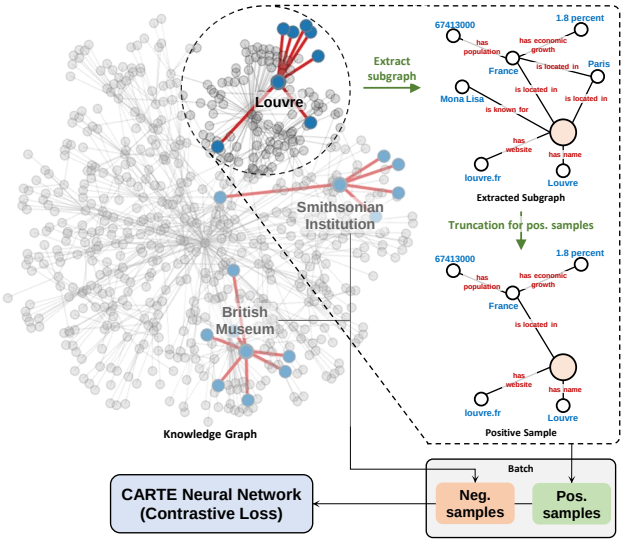


*Figure 2.* **CARTE pretraining process.** From a large knowledge graph, CARTE begins by constructing graphlets and their positives variants. The extracted samples are then fed into the CARTE neural network and trained with a self-supervised scheme. The neural network learns to aggregate information within the graphlets, which reflect the combination of table entries across columns (edges).

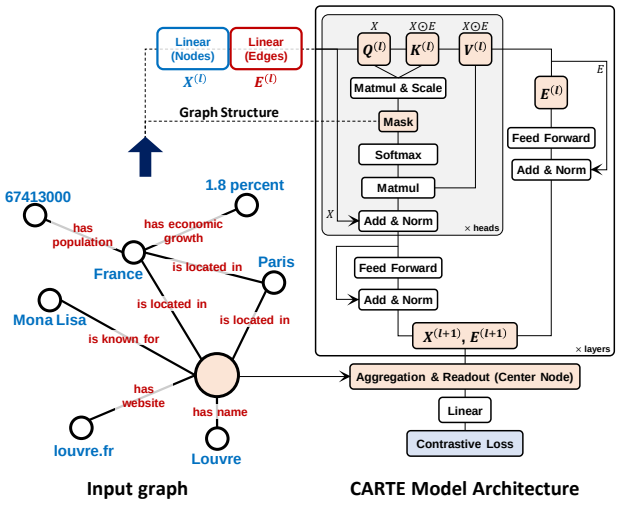


*Figure 3.* **CARTE architecture** The inputs of CARTE are graphs that contain node ($X$) and edge ($E$) features, both used in self-attention layers (shown in grey). The attention layers update node features using the context embodied with the edge information; the graph structure of the input is reflected by attention masks. The Aggregate & Readout layer consists of the attention layer (without the edge update) followed by feature extraction on the center node. The outputs are then processed for the contrastive loss.

relation. Finally, as in subsection 3.1, we initialize node and edge features using FastText embeddings (Mikolov et al., 2017) as the language model.

**Batch samples** To construct a batch sample of size $N_b$, we first select which of the YAGO entities to include, generating the corresponding graphlets. For this, we sample 0.9 of $N_b$ from entities with 6 or more 1-hop relations and the remaining 0.1 from the other subset. The main reason for such sampling scheme is that a large portion of entities in YAGO only have one or two 1-hop relations, while tabular data typical has more (more columns). Moreover, the value 6 was selected so that the rough median of 1-hop relations in the batch samples is 15. To enable the self-supervised contrastive loss, we include positive samples, which are simply truncations of original graphlets: deleting a random fraction (varying from 0.3 to 0.7) of the edges. Figure 2 gives an exemple graphlet of 'Louvre' and its positive.

**Model architecture** Figure 3 depicts the model structure of CARTE. At its basis, CARTE takes the classical Transformer encoder model of Vaswani et al. (2017), and adapts to a graph attentional network. A key component in CARTE's architecture is a self-attention layer computing attention from both node and edge features. In graph models, attention modulates the importance of neighbors for a given node of interest (Velickovic et al., 2017). For table entries, it translates to the importance of the entries for a given instance with the context supplemented by the column information.

We now detail on CARTE's attention mechanism used to capture context and relations. For consistent notations, we write vectors with an arrow on top $\vec{A}$, matrices in bold $\mathbf{A}$, and scalars $A$. To ease reading, we present a single-head attention layer, but it can easily be extended to multi-head schemes of concatenating or averaging the attention outputs.

For a graph with $N$ nodes, let $\vec{X}_i^{(l)}$ 2 $\mathsf{R}^d$ denote the feature of node $i$ and $\vec{E}_{ij}^{(l)}$ 2 $\mathsf{R}^d$ the feature of the edge directed from node $j$ to $i$. By design, graphlets for CARTE always hold the center node, which we denote with an index $i$ = 1. The representation from the attention layer is a function of query, key, and value, crucial elements to account for context. The query is the vectors corresponding to our values of interest: the nodes. Thus, we take the conventional approach and parameterize it with solely the node information. On the other hand, the key-value pairs should convey the elements that the neighboring nodes can offer. Therefore, we add edge information in the corresponding parameterization. With this in mind we set the three components as[1]:

$$\text{Query:} \quad \vec{Q}_i = \vec{X}_i^{(l)} \ \mathbf{W_Q} \tag{1}$$

$$\text{Key:} \quad \vec{K}_{ij} = (\vec{X}_i^{(l)} \quad \vec{E}_{ij}^{(l)}) \ \mathbf{W_K} \tag{2}$$

$$\text{Value:} \quad \vec{V}_{ij} = (\vec{X}_i^{(l)} \quad \vec{E}_{ij}^{(l)}) \ \mathbf{W_V} \tag{3}$$

where denotes the element-wise multiplication and $\mathbf{W_Q}$,

[1]Here, we omit the superscripts indicating layers for $A_{ij}$, $e_{ij}$, and projection weights for $Q$, $K$, $V$ for clarity of presentation.

$\mathbf{W_K}$, and $\mathbf{W_V}$ are trainable weights that reside on $\mathsf{R}^{d\ \ d}$. Here, the choice of element-wise product is motivated from a knowledge graph embedding technique in Balazevic et al. (2019); Cvetkov-Iliev et al. (2023). These works showed that modeling relations (*i.e.* column name) as element-wise multiplication on the node vectors works best, compared to *e.g.,* vector additions. Following the scaled dot-product attention with the above three equations, the attention score of node $j$ from node $i$, $A_{ij}$, is derived as:

$$A_{ij} = \frac{\exp(e_{ij})}{\sum_{k2N_i} \exp(e_{ik})}, \text{ where } e_{ij} = \frac{\vec{Q}_i \vec{K}_{ij}^T}{\sqrt{d}} \quad (4)$$

where the calculation of $A_{ij}$ only takes the sum with respect to the connected neighbors of node $i$. This corresponds to a masking step, which takes the graph structure of the input. Accounting for the relation type (*i.e.* column name) in the attention scores ($\vec{E}$ in eq 2 and 3) is important for proper re-contextualization of the entries, that is to capture their meaning. For instance, an entry "*George Bush*" may denote the 41st or 43rd US presidents, an aircraft carrier... The ambiguity is raised by the relation ("*George Bush*", "*son of*", "*George Bush*"), however capturing fully the information does require knowing the nature of the relation, as "*father of*" would lead to a different entity resolution. Ablations reveal the importance of attention (Appendix C.3).

Outputs of the attention layers are, for nodes and edges:

$$\text{transformed entry} \qquad \vec{X}_i^{(l+1)} = \sigma_X\Big(\sum_j A_{ij} \quad \vec{V}_{ij}\Big)$$

$$\text{transformed relation} \qquad \vec{E}_{ij}^{(l+1)} = \sigma_E(E_{ij}^{(l)} \quad \mathbf{W_E})$$

where $\sigma$ denote the appropriate consecutive operations (see Figure 3). The final layers consist of the attention layer without the edge update, followed by the readout layer that extracts the representation of the center node. For pretraining, the outputs are then processed for the contrastive loss. Appendix A.1 details model specification and training.

**Contrastive loss** For the self-supervised contrastive loss, we adapt the framework of Chen et al. (2020b). The original graphlet and one truncation are set as positives while the other graphlets in the batch are considered as negatives. The learning loss is then based on the cosine similarity of the network outputs, fed in the InfoNCE loss (Oord et al., 2018).

### 3.3. Fine-tuning for Downstream Tasks

For a given downstream task, fine-tuning CARTE proceeds by reusing only part of the pretrained architecture (as shown in Figure 3): the initial layers for nodes and edges (blue and red blocks) and the 'Aggregation & Readout' layer. Though such simplification differs from many fine-tuning approaches, it stems from the behavior of graph-neural networks. Indeed, downstream table entities form simpler graphs than during pre-training. First they are star-like (Figure 1). Second, the downstream tables contain less variability in graph structures and less cardinality of discrete variables compared to YAGO. Too deep an architecture risks washing out discriminant characteristics in the output representations (the over-smoothing problem, Chen et al., 2020a; Rusch et al., 2023, studied in Appendix C.4). Therefore, we use a convention common in graph models: setting the number of attention layers as the maximum $k$-hop relation, here $k = 1$. For the final classifier, we simply attach the linear layers. With the base model for fine-tuning, we consider two different settings of downstream inference.

**Inference on single tables** This is the well-known setting in which we are given a single table with a target variable to predict. Before transforming table entities into graphs, we preprocess numerical variables with a power transform (Yeo & Johnson, 2000). The power transform has shown to be effective in several works (*e.g.,* Hollmann et al., 2023; Cvetkov-Iliev et al., 2023), and likewise, gives stability to the fine-tuning process of CARTE. Moreover we employ a bagging strategy (Breiman, 1996), in which different models, based on different train-validation splits used for early-stopping, are trained. The prediction outputs are calculated as the average of the outputs from each model.

**Transfer from one source table to a target** We also use CARTE in transfer learning settings where we are given a source table $X_S$ that can aid predictions on our target table $X_T$. Importantly, the source table may have larger train samples than the target. We fine-tune CARTE on both tables jointly. The graph representation enables such joint fine tuning without correspondences in the columns; however, we do need to have similar outcomes $y_S$ and $y_T$ on both tables. The source outcomes $y_S$ are transformed to match the first moment of the target outcome $y_T$ using as above a power transform (Yeo & Johnson, 2000, note that here we use the inverse transform). If the target and source table differ on the classification / regression nature of the outcome, we adapt $y_S$ as the following: for a classification target $y_T$, we binarize regression outcomes in the source table, and for a regression target $y_T$, we use binary classification outcomes of the source table, encoded as $\{0, 1\}$ and standard scaled. We then proceed to fine tune CARTE by drawing batches with a fixed proportion of rows from the target and source tables (we use a batch size of 64, 8 of which come from the target). We use early stopping on a validation set of the target table, and still rely on the bagging strategy of building multiple learners on different random validation sets and averaging the predictions. Often, early stopping kicks in before all the data points of the source have been covered. This prevents overfitting the source data, which may be less important than the target data for predicting $y_T$. We use the hyperparameters selected in the single-table setting.

As we choose source tables quite loosely from weakly-related data, the resulting pairwise learner may not actually improve upon the single-table learner if the source table does not bring in enough related information. We thus combine the pairwise learner with the single-table learner ensembling the predictors by combining their output with a softmax. The weights of the softmax are computed using the prediction score computed in the internal validation set of these learners, but divided by the standard deviation across the learners to set the temperature of the softmax.

**Joint learning across multiple tables** The key to transfer learning, as above, is finding the right source table. If we have multiple tables from a given domain or institution, CARTE can be adapted to use them all, finding the most useful information for transfer. In these settings we are given a target table $X_T$ and a set of source tables $X_{S,1}...X_{S,m}$. We proceed to build individual learners: first the single-table learner on $X_T$, then each pair[2] of one source table $X_{S,i}$ and the target table $X_T$ using the pairwise joint learner described above. Here again, not every pairwise learning brings the same amount of useful information. Thus, to find the optimal combination of datasets, we use the same strategy as above of ensembling all the pairwise predictors as well as the single-table predictor. As a consequence, if all source tables lead to predictors that work as well, they are combined with equal weights, but if one source dominates, the prediction is anchored on this one.

## 4. Experimental Study

### 4.1. Experimental Setup

**Datasets** We use 51 tabular learning datasets, all with an associated learning task –40 regressions and 11 classification–, gathered across multiple sources, mainly from previous machine learning studies and kaggle competitions. They cover a variety of topics of society and businesses: accidents, elections, remunerations, food, restaurants, etc. We select datasets representative of modern data science applications: tables with meaningful columns and discrete entries (Table 3), unlike many datasets from UCI. Appendix B gives the specific list of datasets.

**Baselines** We evaluate different baselines with the following abbreviations (specific details on experiment settings and hyper-parameter tuning are presented in Appendix A.2):

**CatBoost** (Dorogush et al., 2018) A gradient-boosted trees package commonly used to learn on tables. We treat text features as categorical, encoded by CatBoost's categorical encoding, an improved version of target encoding (Micci-Barreca, 2001).

**TabVec** The TableVectorizer from the Skrub package (Skrub, 2024) to encode tables that contain string entries into numerical arrays. Columns with low cardinality (number of categories) are one-hot encoded while those with high cardinality are encoded using the Gamma-Poisson encoder from skrub, introduced in Cerda & Varoquaux (2022), which extracts latent categories from substrings. For non tree-based models, missing values are imputed with the mean for numerical features, and treated as another category for categorical features. For neural network models, minmax scale is applied to set values between zero and one.

**XGB, HGB, and RF** Tree-based models: XGBoost (Chen & Guestrin, 2016), HistGradientBoosting and RandomForest (from scikit-learn, Pedregosa et al., 2011).

**MLP and ResNet** The classical Multilayer Perceptron (MLP) and its extension with additional layer-norm/batchnorm and skip-connections (ResNet).

**Ridge and Logistic** Linear models, Ridge and Logistic regression for regression and classification tasks.

**S-LLM** Inspired by TabLLM (Hegselmann et al., 2023), we investigate encoding each row of a table with a large language model (LLM). We represent each row as a sentence, and encode them with `intfloat/e5-small-v2` (Wang et al., 2022) from HuggingFace. Unlike TabLLM, however, the encoded table is passed to the XGB estimator to enable learning for both regression and classification. For numerical entries, we investigate either concatenating them as additional features outside of the LLM (**CN**), or passing them as strings to the LLM (**EN**).

**TabPFN** (Hollmann et al., 2023) is a transformer model pretrained on synthetic data to generate predictions for new (small) datasets in one forward pass. We treat the text features as categorical and encode them with a target encoder (Micci-Barreca, 2001).

### 4.2. Results on Single Tables

**CARTE outperforms alternatives for learning on single tables** Figure 4 compares the prediction performance of the multiple methods, summarizing the different datasets. We see that CARTE consistently outperforms alternatives across the different sample sizes, whether it is with normalized score[3] or critical difference diagrams based on the Conover post hoc test after the Friedman test to detect pairwise significance (Conover, 1999). Another important point is that the bagging strategy used for CARTE also has positive impacts for neural network models: such a bagging

[2]To limit computation cost, we do not explore the full combinatorials of source tables.

[3]The calculation of the normalized score was adapted from Grinsztajn et al. (2022), in which the minimum score is fixed at a value $\rho$: $\rho = 0$ for regression and $\rho = 0.5$ for classification.

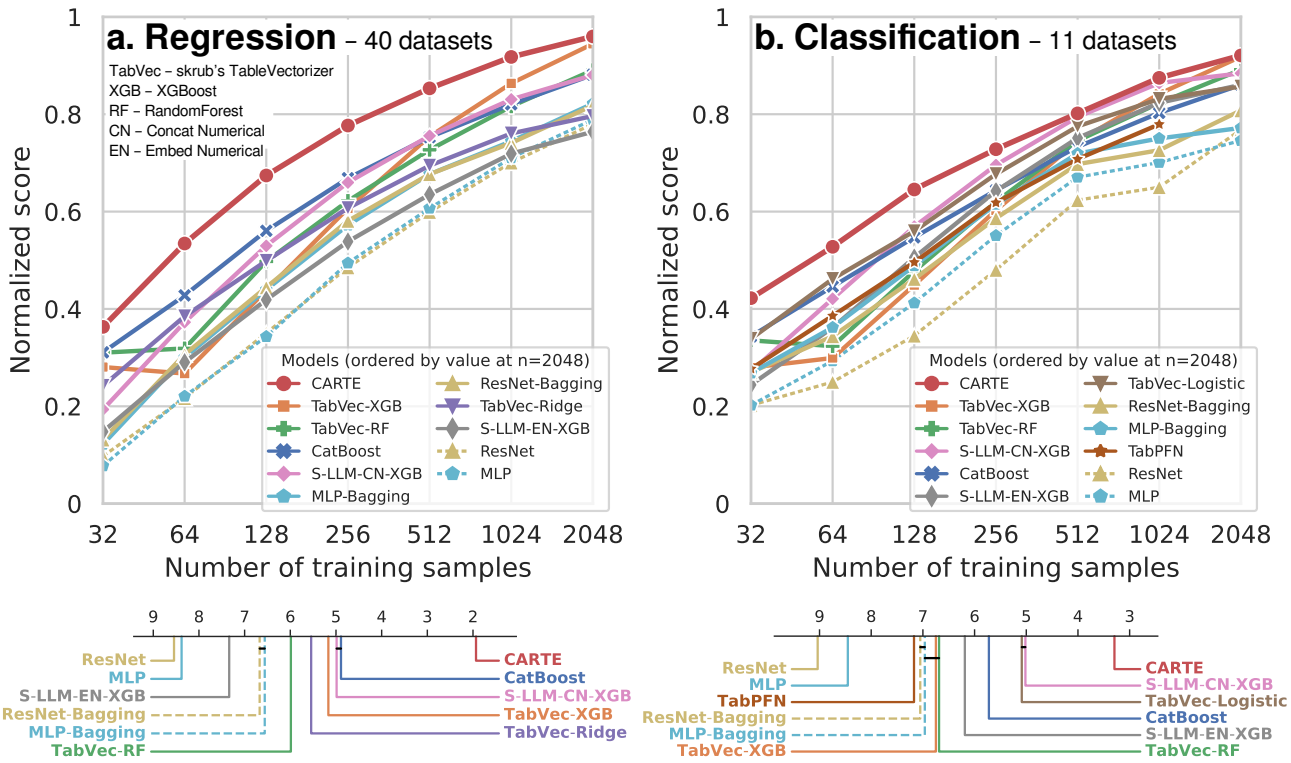


*Figure 4.* **CARTE performs best for learning on single tables**. Learning curve on (a) regression and (b) classification tasks. Top: normalized score (1 is the best performer across all methods and train size for a dataset, and 0 the worst), averaged across datasets. Bottom: critical difference diagrams (Terpilowski, 2019), for all train sizes. Figure 9 gives critical difference diagram for all methods studied.

with different train/validation splits for early stopping may be beneficial for deep learning in general. Appendix C.1 gives comprehensive results of CARTE and 42 baselines.

**CARTE is robust to missing values.** When handling missing values, CARTE discards columns with the missing value in the graph construction step. For example, a data point with one missing value on a table with 10 columns would have nine leaf-nodes after the graph construction step. Table 1 compares the percentage drop in performance and its normalized scores (in comparison to Figure 4) of CARTE and several decision tree baselines that inherently handle the missing values. In the experiment, we randomly drop a proportion of features for each sample (train/test inclusive). The fraction of dropped columns are set as 0.1 (10 % features dropped) and 0.3. The table shows that CARTE continues to outperform the baselines with smaller decrease in performance created by missing values.

**Computation time trade-off for CARTE** Table 2 shows the average computation time (in seconds) of the top-four baselines. The strong prediction performance of CARTE comes at a cost in computation time, and the gap increases with train size ($n$). This cost calls for further optimizations.

*Table 1.* **CARTE is robust to missing values.** Percentage drop in performance and its normalized scores with missing values in which a proportion (0.1 or 0.3) of features are randomly dropped.

**Percentage decrease created by missing values**

| Methods | Train size (Missing fraction) | | | |
|---|---|---|---|---|
| | 64 (0.1) | 64 (0.3) | 512 (0.1) | 512 (0.3) |
| CARTE | 13.28% | **38.35%** | **10.19%** | **24.42%** |
| CatBoost | 21.70% | 53.32% | 12.23% | 29.70% |
| TabVec-XGB | 15.11% | 51.27% | 12.61% | 30.35% |
| TabVec-RF | **7.68%** | 44.43% | 12.77% | 29.79% |

**Normalized absolute score** (as in Figure 4)

| | | | | |
|---|---|---|---|---|
| CARTE | $\mathbf{0.44}_{(0.20)}$ | $\mathbf{0.29}_{(0.18)}$ | $\mathbf{0.75}_{(0.12)}$ | $\mathbf{0.61}_{(0.14)}$ |
| CatBoost | $0.31_{(0.22)}$ | $0.17_{(0.17)}$ | $0.65_{(0.15)}$ | $0.50_{(0.15)}$ |
| TabVec-XGB | $0.19_{(0.20)}$ | $0.11_{(0.15)}$ | $0.65_{(0.17)}$ | $0.50_{(0.17)}$ |
| TabVec-RF | $0.23_{(0.21)}$ | $0.14_{(0.16)}$ | $0.63_{(0.15)}$ | $0.49_{(0.15)}$ |

**Entity matching not required for CARTE** The hypothesis to explain the good performance of CARTE is that has integrated information on many entities because it has been pretrained on the YAGO knowledge base. However, in a given downstream table, entities might appear written in a

*Table 2.* **Computation time (in seconds) for top-four baselines** for preprocessing, training and testing, across the 51 datasets.

| Methods | Preprocessing | | Learning $n = 64$ | | Learning $n = 512$ | |
|---|---|---|---|---|---|---|
| CARTE | 50.20 | 63.68 | 85.43 | 60.30 | 315.49 | 119.84 |
| CatBoost | - | | 0.98 | 1.19 | 1.05 | 1.06 |
| TabVec-XGB | 64.72 | 139.23 | 0.40 | 0.21 | 1.19 | 0.94 |
| S-LLM-XGB | 207.87 | 361.56 | 0.87 | 0.71 | 3.49 | 1.79 |

different way: for instance "Londres" instead of "London". This begs the question of whether CARTE transfers well useful information if the string representation of the entities differs. String matching is necessary for instance when using as features vector embeddings of the entries such as KEN embeddings (Cvetkov-Iliev et al., 2023).

On four of our datasets (company employees, movies, US accidents, and US election), we performed manual entity matching of the entries to their corresponding YAGO entity. Figure 5 shows that while using KEN requires entity matching to have good performance, the string-level modeling in CARTE make its performance robust to entity variants: using manually matched entities or original entries. Ablations confirm the importance of string-level representations that also capture semantic similarity (Appendix Figure 10).

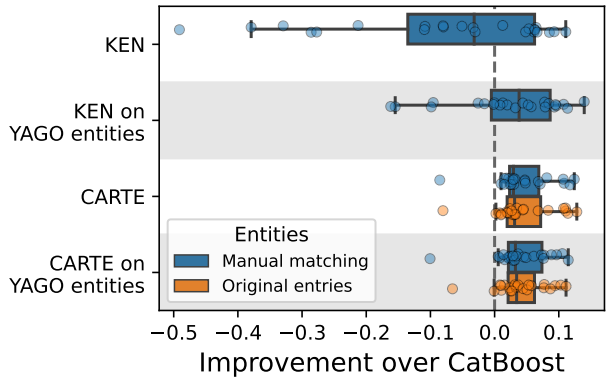


*Figure 5.* **Entity matching not required for CARTE**, and downstream entities do not need to be in YAGO. We evaluate CARTE and KEN either on the full datasets, or on a reduced version of the datasets corresponding to entities present in YAGO. In addition, when entities are present in YAGO, we either match them to their canonical names in YAGO (blue) or keep the original names (orange). When KEN is used to enrich the dataset, CatBoost is used as the estimator, and entities without matching are replaced with missing values. Each point on the figure correspond to an improvement in performance with respect to Catboost without any enrichment. That KEN brings performance gains to CatBoost on YAGO entities confirms the added value of background information. Appendix C.5 gives detailed results.

*Table 3.* **Difference between this study's benchmark and TabLLM datasets.** Our benchmark datasets contain more categorical columns, in particular with higher cardinality ($jCj$).

| Characteristics | This study's benchmark | TabLLM |
|---|---|---|
| Fraction of numerical cols. | 0.194 | 0.613 |
| Fraction of cols. with $jCj > 10$ | 0.625 | 0.043 |
| Cardinality over data size | 0.263 | 0.001 |

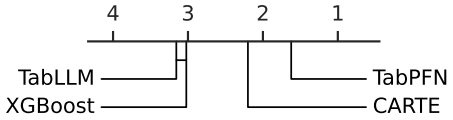


*Figure 6.* **Comparison to three baselines in TabLLM** (Hegselmann et al., 2023). The datasets contain mostly numerical features or low-cardinality categorical columns. In such settings, TabPFN performs best, followed by CARTE, XGBoost, and TabLLM.

**Comparison to TabLLM baselines** We compare CARTE to three baselines over nine datasets presented TabLLM (Hegselmann et al., 2023). Compared to the datasets in Figure 4, the datasets in TabLLM contain higher fraction of numerical features with less cardinality of categorical columns (see Table 3). Figure 6 gives the critical difference diagram of the baselines. TabPFN shows strength in such settings. Yet, CARTE can attain competitive performances although it is geared towards handling both numerical values and strings. Detailed results are analyzed in Appendix C.2.

### 4.3. Learning Across Multiple Tables

We investigate learning across multiple tables without explicit correspondences across columns. We use tables "in the wild": in our 51 datasets, we find groups covering the same general topic (bike prices, restaurant ratings) though they come from different sources (Appendix B.3). In this setting, we can readily use CARTE and S-LLM approaches as they use an open-vocabulary representation of the column to embed entries (but only the EN, Embed Numerical, version of S-LLM, as the CN, Concat Numerical, needs correspondence for the numerical columns). As CatBoost natively deals with missing values, we use it by concatenating the datasets and adding missing values for mismatch columns. We also investigate manual matching of columns.

**Schema matching not required for CARTE** Figure 7 shows results for transfer learning across only two tables. We see that for all approaches transfer learning can help (the dashed line, representing the learning only on the target table, is below), but only CARTE provides consistent improvements without requiring manual column matching.

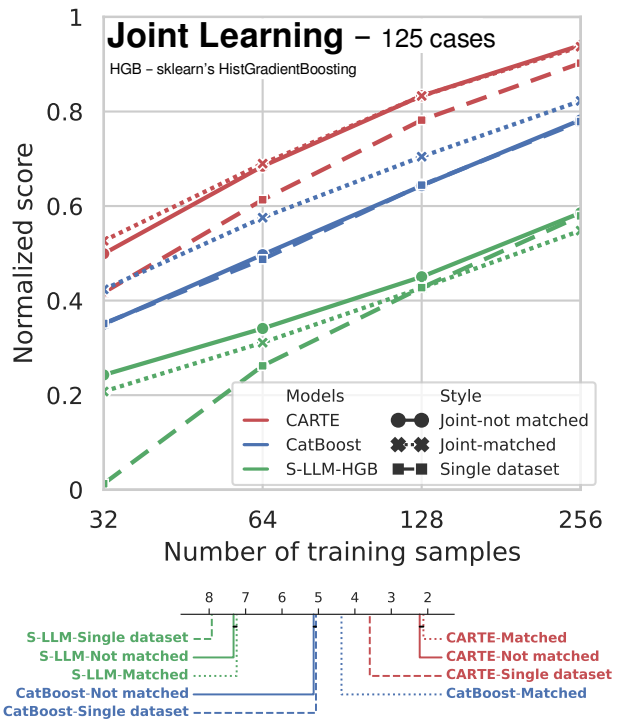


*Figure 7.* **Schema matching not required for CARTE**, with consistent improvements through joint learning. We compare three scenarios (style) – single (dashed lines): only the target tables; joint (full lines): the automated transfer learning without any manual operation; matched (dotted lines): transfer learning after manually matching the columns.

For CatBoost, the matching (dotted line) is crucial, which is not the case for the other approaches. For S-LLM, the benefit of transfer drops rapidly with respect to the number of training samples. The results show that CARTE does not need schema matching, and it provides consistent improvements in the target table with transfer. Extended results of schema-matching can be found in Appendix C.6.

**Joint learning from multiple tables** The difficulty for transfer learning may be finding good source tables. In Figure 8, we investigate bringing in more source tables, up to a total of 4 tables (1 target, and 3 sources, with a total of 245 cases). CARTE benefits from adding source tables: not only does the median performance improve but also, the lower bound in variability improves. In other words, more source tables give higher chances of finding a beneficial one, and thus the worst-case scenario becomes better.

## 5. Discussion and Conclusion

**Strings and numbers in tables** Our study touches on the importance of strings in tabular data. They are often overlooked in tabular machine learning (Table 3 shows how

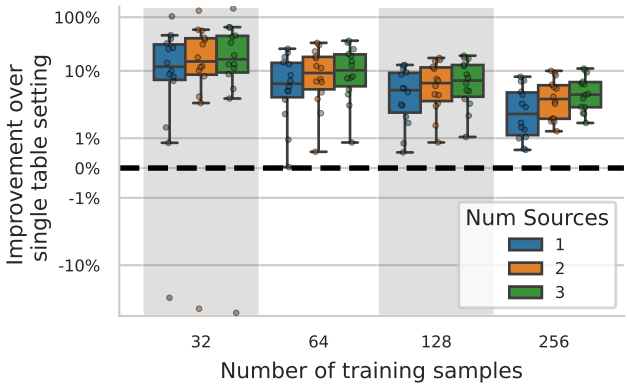


*Figure 8.* **CARTE further benefits from additional source tables.**

datasets are mostly numerical or low-cardinality categories), but central to database research, which focuses on discrete entries. Compared to most tabular-learning models (whether tree-based, or neural networks as TabPFN), the edge of CARTE is on the strings. String preprocessing in skrub's TableVectorizer also boosts baselines (Figure 9). Conversely, language models, as LLMs, focus on strings, enabling pretraining on huge corpora. They give great preprocessing of strings but must be combined with tree-based methods to handle numbers (S-LLM-CN-XGB in our benchmark). CARTE is is tailored to both strings and numbers.

**An architecture that boosts performance** By using an architecture that models table entries not as a $i^{\text{th}}$ feature in a data matrix but as a function of its context (column names and neighboring entries) as well open-vocabulary embedding of strings, CARTE enables consistent representations of very different tables. This opens the door to pre-training across background tables, and fine-tuning to downstream tasks without matched entities or schema. Results show that after pretraining on a large knowledge base, the resulting model brings marked benefits to downstream analytic tasks, consistently outperforming a broad range of baselines both for learning on a single table or transfer learning on tables with imperfect correspondences. It enables transfer from tables in the wild, a setting so far never studied.

**Toward tabular foundation models** Pre-training has been key to the wide application of deep learning on images and text. We hope that the ideas behind CARTE will bring these benefits to tabular learning, leading to tabular foundation models. This will call for further improved architectures: optimizing for larger train-sizes; refining numerical representations with ideas of Gorishniy et al. (2022); leveraging more training information and expressive attention as in large language models; merging with the complementary meta-learning ideas of TabPFN (Hollmann et al., 2023), which shines on heavily-numerical tables.

## Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here. The societal impact of a method depends on how it is used. We do note, however, that tabular data are central to fields such as healthcare which uses a lot codes and more or less normalized entities (*e.g.,* ICD10 codes for diseases, medical informatics as a field has invested hugely on data integration). We thus hope that pretraining a tabular model on health data could provide value to this field, and in turn positive societal impact.

On another topic, we note that our model, CARTE, comes with additional computational cost compared to baselines. We do expect that further research and engineering will bring these costs down. However, our work opens the door to pre-trained models for tabular data, one day maybe foundation models. These models have led to a race for ever-increasing size, which comes with dire consequences in terms of energy and financial cost, carrying over to ecology and concentration of power.

## Acknowledgments

The authors acknowledge the support in part by the French Agence Nationale de la Recherche under Grant ANR-20-CHIA-0026 (LearnI).

We also would like to thank the ICML reviewers, who challenged us in a good way, leading to improved empirical study of CARTE, and hence a better manuscript.

## References

Abutbul, A., Elidan, G., Katzir, L., and El-Yaniv, R. DNF-Net: A Neural Architecture for Tabular Data, June 2020.

Arik, S. O. and Pfister, T. TabNet: Attentive Interpretable Tabular Learning, December 2020.

Balazevic, I., Allen, C., and Hospedales, T. Multi-relational poincaré graph embeddings. *Advances in Neural Information Processing Systems*, 32, 2019.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Breiman, L. Bagging predictors. *Machine learning*, 24: 123–140, 1996.

Cerda, P. and Varoquaux, G. Encoding high-cardinality string categorical variables. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):1164–1176, March 2022. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/TKDE.2020.2992529.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3438–3445, 2020a.

Chen, J., Yan, J., Chen, D. Z., and Wu, J. ExcelFormer: A Neural Network Surpassing GBDTs on Tabular Data, January 2023.

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*, pp. 785–794, 2016.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020b.

Conover, W. J. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

Crossley, S., Heintz, A., Choi, J. S., Batchelor, J., Karimi, M., and Malatinszky, A. A large-scaled corpus for assessing text readability. *Behavior Research Methods*, 55(2): 491–507, 2023.

Cvetkov-Iliev, A., Allauzen, A., and Varoquaux, G. Relational data embeddings for feature enrichment with background information. *Machine Learning*, 112(2):687–720, 2023.

Deng, X., Sun, H., Lees, A., Wu, Y., and Yu, C. TURL: Table Understanding through Representation Learning, December 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019.

Doan, A., Halevy, A., and Ives, Z. *Principles of data integration*. Elsevier, 2012.

Dorogush, A. V., Ershov, V., and Gulin, A. CatBoost: Gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

Eggert, G., Huo, K., Biven, M., and Waugh, J. TabLib: A Dataset of 627M Tables with Context, October 2023.

Fey, M., Hu, W., Huang, K., Lenssen, J. E., Ranjan, R., Robinson, J., Ying, R., You, J., and Leskovec, J. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023.

Gardner, J., Popovic, Z., and Schmidt, L. Subgroup Robustness Grows On Trees: An Empirical Baseline Investigation. *Advances in Neural Information Processing Systems*, 35:9939–9954, December 2022.

Gorishniy, Y., Rubachev, I., and Babenko, A. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35: 24991–25004, 2022.

Gorishniy, Y., Rubachev, I., Kartashev, N., Shlenskii, D., Kotelnikov, A., and Babenko, A. TabR: Tabular Deep Learning Meets Nearest Neighbors in 2023, October 2023a.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting Deep Learning Models for Tabular Data, October 2023b.

Grinsztajn, L., Oyallon, E., and Varoquaux, G. Why do tree-based models still outperform deep learning on tabular data?, July 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, March 2023.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second, September 2023.

Hulsebos, M., Hu, K., Bakker, M., Zgraggen, E., Satyanarayan, A., Kraska, T., Demiralp, Ç., and Hidalgo, C. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1500–1508, 2019.

Hulsebos, M., Demiralp, Ç., and Groth, P. GitTables: A Large-Scale Corpus of Relational Tables. *Proceedings of the ACM on Management of Data*, 1(1):1–17, May 2023. ISSN 2836-6573. doi: 10.1145/3588710.

Levin, R., Cherepanova, V., Schwarzschild, A., Bansal, A., Bruss, C. B., Goldstein, T., Wilson, A. G., and Goldblum, M. Transfer Learning with Deep Tabular Models, August 2023.

Mahdisoltani, F., Biega, J., and Suchanek, F. M. Yago3: A knowledge base from multilingual wikipedias. In *CIDR*, 2013.

McElfresh, D., Khandagale, S., Valverde, J., C, V. P., Feuer, B., Hegde, C., Ramakrishnan, G., Goldblum, M., and White, C. When Do Neural Nets Outperform Boosted Trees on Tabular Data?, October 2023.

Micci-Barreca, D. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter*, 3(1):27–32, 2001.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405*, 2017.

Narayan, A., Chami, I., Orr, L., and Ré, C. Can foundation models wrangle your data? *Proceedings of the VLDB Endowment*, 16(4):738–746, 2022.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

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.

Popov, S., Morozov, S., and Babenko, A. Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data, September 2019.

Rusch, T. K., Bronstein, M. M., and Mishra, S. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

Sanjib, D., AnHai, D., Suganthan, P., Chaitanya, G., Pradap, K., Yash, G., and Derek, P. The Magellan Data Repository, 2023.

Shwartz-Ziv, R. and Armon, A. Tabular Data: Deep Learning is Not All You Need, November 2021.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR 2015)*, 2015.

Skrub. Skrub, prepping tables for machine learning. https://skrub-data.org, 2024.

Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C. B., and Goldstein, T. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training, June 2021.

Terpilowski, M. scikit-posthocs: Pairwise multiple comparison tests in python. *The Journal of Open Source Software*, 4(36):1169, 2019. doi: 10.21105/joss.01169.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

UCI. UC Irvine Machine Learning Repository. https://archive.ics.uci.edu.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., et al. Graph attention networks. *stat*, 1050 (20):10–48550, 2017.

Wang, L., Yang, N., Huang, X., Jiao, B., Yang, L., Jiang, D., Majumder, R., and Wei, F. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

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

Yeo, I.-K. and Johnson, R. A. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

Zhu, B., Shi, X., Erickson, N., Li, M., Karypis, G., and Shoaran, M. Xtab: Cross-table pretraining for tabular transformers. *arXiv preprint arXiv:2305.06090*, 2023.

# A. Detailed Information on Training

## A.1. Pretrained Model of CARTE

**Model specification and training details** The model specification and training details were largely referenced from the work of (Devlin et al., 2019). We set 12 attention layers, each consisting of 12 multi-head attentions, and the hidden dimension was fixed to the same size as the inputs (300). The resulting model contains over 9.3 million parameters. To run the pretraining, we selected 128 entities with one additional positive, resulting in the batch size of 256. The total number of steps for training was $1,000,000$, which approximately covers 40 epochs with respect to YAGO entities. We use the AdamW optimizer accompanied by the cosine scheduler with $lr_{min} = 5 \quad 10^{-6}$, $lr_{max} = 1 \quad 10^{-4}$ and a warmup over the first $10,000$ steps. The dropout rate was fixed to 0.1 and the gelu activation function was used.

## A.2. Details on Experiment Settings for Downstream Tasks

**Single tables** To evaluate the performances of baselines on single tables, we focused on the setting with limited train-size for each table varying from 32, 64, 128, 256, 512, $1,024$, and $2,048$; the rest of the remaining data were set as the test set. To find the optimal hyperparameters of the baselines, 5-fold cross-validation over 100 random search iteration were carried out on all the comparing methods except for CARTE and TabPFN. For CARTE, the same 5-fold cross-validation, but only the grid-search over the learning rate was conducted. For TabPFN, we ran with the default values, as suggested in the paper. For detailed information the hyperparameter spaces of each method, please refer to the Hyperparmeter tuning paragraph below. The performance was recorded on 10 different train/test splits, with the performance measure set as the $R^2$ score for regression and the Area Under Receiver Operating Curve (AUROC) for classification tasks.

**Joint learning across multiple tables** The experiment settings for joint learning is almost the same as in the single table setting, except for minor details. The number of train-set on the target was varied across 32, 64, 128, 256, while the same split was set as in the case of single-tables to make the results comparable. In terms of hyperparameter optimization, CARTE only takes the best values obtained from the the single-table case (section 3). For other baselines, the same scheme for hyperparameter search was conducted.

**Hyperparameter space** The hyperparameter tuning was done using grid search for CARTE, as we only tune the learning rate, and with random search for the baselines as these come with more than two hyperparameters to tune. The hyperparmeter spaces for XGBoost, HistGradientBoosting, RandomForest, Resnet, and MLP baselines are based on that used in Grinsztajn et al. (2022); for CatBoost we follow that used in the CatBoost paper (Dorogush et al., 2018). For the baselines in joint learning across multiple tables, we employ an additional hyperparameter 'fraction source', which denote the fraction of source data used for training. Table 4 below summarizes the hyperparameter spaces for each of the estimators.

## A.3. Hardware Specifications

The pretrained model for CARTE was trained on GPUs. For rest of our experiments, they were run on 32 cores of CPU and the hardware was chosen based on availability.

**GPUs:** NVIDIA V100 (32GB VRAM)

**CPUs:** AMD EPYC 7742 64-Core Processor, AMD EPYC 7702 64-Core Processor (512GB RAM), Intel(R) Xeon(R) CPU E5-2660 v2, Intel(R) Xeon(R) Gold 6226R CPU (256GB RAM)

## A.4. Implementation of CARTE

The implementation and datasets will be avaliable at:

**Implementation:** `https://github.com/soda-inria/carte`.

**Datasets:** `https://huggingface.co/datasets/inria-soda/carte-benchmark`

*Table 4.* Hyperparameter space for CARTE and baseline estimators.

| Methods | Parameters | Grid |
|---|---|---|
| CARTE | Learning rate | [2.5, 5, 7.5] [$1e^{-4}$, $1e^{-3}$] |
| CatBoost | Max depth | UniformInt [2, 10] |
| | Learning rate | LogUniform [$1e^{-5}$, 1] |
| | Bagging temperature | Uniform [0, 1] |
| | $l_2$-leaf regularization | LogUniform [1, 10] |
| | One hot max size | UniformInt [2, 25] |
| | Iterations | UniformInt [400, 1000] |
| XGBoost | Num estimators | UniformInt [50, 1000] |
| | Max depth | UniformInt [2, 10] |
| | Learning rate | LogUniform [$1e^{-5}$, 1] |
| | Min child weight | LogUniform [1, 100] |
| | Subsample | Uniform [0.5, 1] |
| | Colsample by level | Uniform [0.5, 1] |
| | Colsample by tree | Uniform [0.5, 1] |
| | Gamma | LogUniform [$1e^{-8}$, 7] |
| | Lambda | LogUniform [1, 4] |
| | Alpha | LogUniform [$1e^{-8}$, 100] |
| HistGradientBoosting | Learning rate | LogUniform [$1e^{-2}$, 10] |
| | Max depth | [None, 2, 3, 4] |
| | Max leaf nodes | NormalInt [31, 5] |
| | Min samples leaf | NormalInt [20, 2] |
| | $l_2$-regularization | LogUniform [$1e^{-6}$, $1e^{3}$] |
| RandomForest | Num estimators | UniformInt [50, 250] |
| | Max depth | [None, 2, 3, 4] |
| | Max features | [sqrt, sqrt, log2, None, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9] |
| | Min samples leaf | LogUniform [1.5, 50.5] |
| | Bootstrap | [True, False] |
| | Min impurity decrease | [0, 0.01, 0.02, 0.05] |
| ResNet | Num layers | UniformInt [1, 8] |
| | Layer size | UniformInt [32, 512] |
| | Hidden factor | UniformInt [1, 3] |
| | Hidden dropout | Uniform [0, 0.5] |
| | Residual dropout | Uniform [0, 0.5] |
| | Learning rate | LogUniform [$1e^{-5}$, $1e^{-2}$] |
| | Weight decay | LogUniform [$1e^{-8}$, $1e^{-2}$] |
| | Normalization | [batchnorm, layernorm] |
| | Batch size | [16, 32] |
| MLP | Num layers | UniformInt [1, 4] |
| | Layer size | UniformInt [16, 1024] |
| | Dropout | Uniform [0, 0.5] |
| | Learning rate | LogUniform [$1e^{-5}$, $1e^{-2}$] |
| | Weight decay | LogUniform [$1e^{-8}$, $1e^{-2}$] |
| | Batch size | [16, 32] |
| Ridge Regression | Solver | [svd, cholesky, lsqr, sag] |
| | Alpha | LogUniform [$1e^{-5}$, 100] |
| Logistic Regression | Solver | [newton-cg, lbfgs, liblinear] |
| | Penalty | [none, $l_1$, $l_2$, elasticnet] |
| | C | LogUniform [$1e^{-5}$, 100] |
| Baselines in joint learning | Source fraction | Uniform [0, 1] |

# B. Data Description

## B.1. Data Preprocessing

Only minimal data preprocessing were carried out in data preparation. For all datasets, we excluded columns that contained only one unique value or had missing values over half the size of the dataset.

## B.2. Datasets

We provide detailed description of the datasets used in our experiment study.

1. **Anime Planet**[4] This dataset contains information about anime scrapped from the website Anime-Planet. The task is to predict the average rating of the anime on this site.
2. **Babies R Us** (Sanjib et al., 2023)[5] Information of baby products scraped from the Babies R Us website. The task is to predict the price of baby products.
3. **Buy Buy Baby** (Sanjib et al., 2023)[6] Information of baby products scraped from the Buy Buy Baby website. The task is to predict the price of baby products.
4. **Beer Ratings**[7] The dataset contains tasting profiles and consumer reviews for 3197 unique beers from 934 different breweries. The task is to predict overall review ratings of different beers.
5. **Bikedekho** (Sanjib et al., 2023)[8] Information on bikes and scooters from bikedekho website in India. The task is to predict the price of bikes.
6. **Bikewale** (Sanjib et al., 2023)[9] Information on bikes and scooters from bikewale website in India. The task is to predict the price of bikes.
7. **Cardekho**[10] This dataset contains information on used cars, with their listing price in the websit Cardekho. The task is to predict the price.
8. **Chocolate Bar Ratings**[11]Dataset containing information and expert rating on cocoa batches. The task is to predict the rating.
9. **Clear Corpus** (Crossley et al., 2023)[12] Generic information about the reading passage excerpts for elementary school students. The task is to predict the readability of the excerpts. The text feature is the name of the book, not the excerpt.
10. **Coffee Ratings**[13] Dataset scraped from coffeereview.com containing information on various coffees. The task is to predict the review ratings of the coffees.
11. **Company Employees**[14] Information on companies with over $1,000$ employees. The task is to predict the number of employees of each company.
12. **Employee remuneration and expenses earning over 75000**[15] Remuneration and expenses for employees earning over $75,000 per year. The task is to predict the remuneration of employees.

[4] https://www.kaggle.com/datasets/hernan4444/animeplanet-recommendation-database-2020
[5] http://pages.cs.wisc.edu/~anhai/data/784_data/bikes/csv_files/babies_r_us.csv
[6] http://pages.cs.wisc.edu/~anhai/data/784_data/bikes/csv_files/buy_buy_baby.csv
[7] https://www.kaggle.com/datasets/ruthgn/beer-profile-and-ratings-data-set
[8] http://pages.cs.wisc.edu/~anhai/data/784_data/bikes/csv_files/bikedekho.csv
[9] http://pages.cs.wisc.edu/~anhai/data/784_data/bikes/csv_files/bikewale.csv
[10] https://www.kaggle.com/datasets/sukritchatterjee/used-cars-dataset-cardekho
[11] https://www.kaggle.com/datasets/rtatman/chocolate-bar-ratings
[12] https://www.commonlit.org/blog/introducing-the-clear-corpus-an-open-dataset-to-advance-research-28ff8cfea84a
[13] https://www.kaggle.com/datasets/hanifalirsyad/coffee-scrap-coffeereview
[14] https://www.kaggle.com/peopledatalabssf/free-7-million-company-dataset
[15] https://opendata.vancouver.ca/explore/dataset/employee-remuneration-and-expenses-earning-over-75000/information/?disjunctive.department&disjunctive.title

13. **Employee Salaries**[16] Information on salaries for employees of the Montgomery County, MD. The task is to predict the current annual salary range of the employees.

14. **Fifa22 Players**[17] Information on soccer players and their ability scores in Fifa22 game. The task is to predict the player's wage.

15. **Filmtv Movies**[18] Information of movies and ratings scraped from an Italian movie review website Filmtv Movies. The task is to predict the public vote on movies.

16. **Journal Score JCR** Scientific journals and their descriptive features from Journal Citation Reports. The task is to predict the impact factors of the journals.

17. **Journal Score SJR** Scientific journals and their descriptive features from Scimago journal rank. The task is to predict the H-index of journals.

18. **Japanese Anime**[19] List of Japanese animes and their relevant information. The task is to predict score for the animes.

19. **K-Drama**[20] List of korean drama and their basic information from mydramalist website. The task is to predict the score of the Korean dramas.

20. **Michelin**[21] List of restaurants along with additional details curated from the Michelin Restaurants guide. The task is to predict the award of the restaurants.

21. **ML/DS Salaries**[22] salary and basic information of workers in machine learning and data science industry. The task is to predict the salary of workers.

22. **Movie Revenues**[23] Metadata of movies released on or before July 2017. The task is to predict the range of the box-office revenues.

23. **Museums**[24] General information on the US museums. The task is to predict the revenues across the museums.

24. **Mydramalist**[25] General information on Asian drama scraped from mydramalist website. The task is to predict the ratings of Asian dramas.

25. **NBA Draft**[26] Information on all NBA Draft picks from 1989-2021. The task is to predict the 'value over replacement' of players.

26. **Prescription Drugs**[27] The data contains new prescription drugs introduced to market in California with a Wholesale Acquisition Cost (WAC) that exceeding Medicare Part D. The task is to predict WAC at introduction.

27. **Ramen ratings**[28] The dataset contains ratings and characteristics of various ramens produced from multiple countries. The task is to predict the range of ratings of the ramens.

28. **Roger Ebert**[29] The dataset contains movies ratings by famous critic Rogert Ebert. The task is to predict the range of ratings.

[16] https://openml.org/d/42125
[17] https://www.kaggle.com/datasets/joebeachcapital/fifa-players
[18] https://www.kaggle.com/datasets/stefanoleone992/filmtv-movies-dataset/data
[19] https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset
[20] https://www.kaggle.com/datasets/noorrizki/top-korean-drama-list-1500
[21] https://www.kaggle.com/datasets/ngshiheng/michelin-guide-restaurants-2021
[22] https://ai-jobs.net/salaries/download/salaries.csv
[23] https://www.kaggle.com/rounakbanik/the-movies-dataset
[24] https://www.kaggle.com/datasets/markusschmitz/museums
[25] https://www.kaggle.com/datasets/rajchinagundi/mydramalist-complete-dataset
[26] https://www.kaggle.com/datasets/mattop/nba-draft-basketball-player-data-19892021
[27] https://data.ca.gov/uk/dataset/prescription-drugs-introduced-to-market
[28] https://www.kaggle.com/datasets/ankanhore545/top-ramen-ratings-2022
[29] https://github.com/gabrielcs/movie-ratings-prediction

29. **Rotten Tomatoes** (Sanjib et al., 2023)[30] Contain information on movies that can be found in Rotten Tomatoes movie rating website. The task is to predict the rating values of the movies.

30. **Spotify**[31] Generic information on Spotify tracks with some associated audio features. The task is to predict the popularity of the albums.

31. **US Accidents**[32] Information of accidents in US cities between 2016 and 2020. From this dataset, two tasks are conducted: (1) the range of accident counts for the US cities (2) the severity of the reported accidents.

32. **US Presidential** (Cvetkov-Iliev et al., 2023) Voting statistics in the 2020 US presidential election along with information on US counties. The task is to predict the range of voting numbers across US counties.

33. **Used Cars 24**[33] Information on used cars. The task is to predict the price.

34. **Used Cars Benz Italy**[34] Dataset containing information on used cars sold in Italy. The task is to predict the price.

35. **UsedCars.com** Dataset containing information on used cars usedcars.com. The task is to predict the price.

36. **Used Cars Pakistan**[35] Dataset containing information on used cars sold in Pakistan. The task is to predict the price.

37. **Used Cars Saudi Arabia**[36] Dataset containing information on used cars sold in Saudi Arabia from the Syarah Website. The task is to predict the price of used cars.

38. **Videogame Sales**[37] This dataset contains a list of video games with sales greater than 100,000 copies (scrape of vgchartz.com). The task is to predict the global sales of the videogames.

39. **Whisky**[38] Basic and tasting information on whiskies form the whiskyanalysis.com. The task is to predict the range of meta critic of the whiskies.

40. **Wikiliq**[39] Information on alcohol that can be found in the Wikiliq website. We conducted two tasks to predict the prices of (1) beer and (2) spirits.

41. **Wina Poland**[40] Information about wines on the polish market. The task is to predict the price.

42. **Wine.com**[41] Information on wines scraped from the wine.com website. We conducted two tasks on prediction of (1) wine ratings and (2) wine prices.

43. **WineEnthusiasts**[42] Information about a wine and a taster from winemag.com. We conducted two tasks on prediction of (1) wine ratings and (2) wine prices.

44. **WineVivino**[43] Information about wine bottles scrapped from Vivino's website. We conducted two tasks on prediction of (1) wine ratings and (2) wine prices.

45. **Yelp**[44] The Yelp Open dataset for academic research. We extracted information on the restaurants from the original dataset. The task is to predict the range of stars for the restaurants. https://www.yelp.com/dataset

[30] http://pages.cs.wisc.edu/~anhai/data/784_data/movies1/csv_files/rotten_tomatoes.csv
[31] https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset
[32] https://smoosavi.org/datasets/us_accidents
[33] https://www.kaggle.com/datasets/avikasliwal/used-cars-price-prediction
[34] https://www.kaggle.com/datasets/bogdansorin/second-hand-mercedes-benz-registered-2000-2023-ita
[35] https://www.kaggle.com/datasets/mustafaimam/used-car-prices-in-pakistan-2021
[36] https://www.kaggle.com/datasets/turkibintalib/saudi-arabia-used-cars-dataset
[37] https://www.kaggle.com/datasets/gregorut/videogamesales
[38] https://whiskyanalysis.com/index.php/database/
[39] https://www.kaggle.com/datasets/limtis/wikiliq-dataset
[40] https://www.kaggle.com/datasets/skamlo/wine-price-on-polish-market
[41] https://www.kaggle.com/datasets/manyregression/updated-wine-enthusiast-review
[42] https://www.kaggle.com/datasets/manyregression/updated-wine-enthusiast-review
[43] https://www.kaggle.com/datasets/joshuakalobbowles/vivino-wine-data
[44] https://www.yelp.com/dataset

46. **Zomato**[45] Information and reviews of restaurants found in the zomato websites. The task is to predict the range of ratings for each restaurants.

### B.3. Datasets from multi-table experiments

For the multi-table experiments we extract from the list above groups of tables that are related to the same topics:

**Wine prices**: Wina Poland, WineEnthusiasts, WineVivino, Wine.com

**Wine ratings**: WineEnthusiasts, WineVivino, Wine.com

**Beers**: Beer Ratings, Wikiliq-Beer

**Used Car**: Used Cars 24, Used Cars Benz Italy, UsedCars.com, Used Cars Pakistan, Used Cars Saudi Arabia

**Films**: Filmtv Movies, Rotten Tomatoes

**Dramas**: K-Drama, Mydramalist

**Animes**: Anime Planet, Japanese Anime

**Baby products**: Buy Buy Baby, Babies R Us

**Bike sales**: Bikedekho, Bikewale

**Employee remunerations**: Company Employees, Employee remuneration and expenses earning over 75000, ML/DS Salaries

**Restaurant ratings**: Zomato, Michelin, Yelp

**Journal scores**: Journal Score JCR, Journal Score SJR

## C. Extended Results

### C.1. Performance Comparison of CARTE with 42 Baselines for Learning on Single Tables

As an extension to the results shown in subsection 4.2, Figure 9 shows the overall comparison of CARTE with 42 baselines that additionally accounts for Scikit-Learn's HistGradientBoosting (**HGB**), target encoding for categorical variables (Micci-Barreca, 2001) (**TarEnc**), employing external information from language models of Fasttext (**FT**) and `intfloat/e5-small-v2` (Wang et al., 2022) (**LLM**), and the '**Bagging**' strategy. We see that CARTE attains the pronounced lead to all the baselines for both regression and classification tasks. Moreover, it is interesting to observe that the bagging strategy has positive impacts for neural network models, while the effect is limited on linear or ensembling baselines (tree-based models or TabPFN). This may hint that bagging with different train/validation splits is an important setups for other deep learning architectures, especially for limited train-sizes.

### C.2. Detailed Results for TabLLM Datasets

Table 5 shows the dataset specifications and detailed results on performance comparison between CARTE and baselines presented in Hegselmann et al. (2023). The datasets generally contain high fraction of numerical features (four datasets) or categorical columns with low cardinality (eight datasets). In such settings, TabPFN tends to outperform other methods. For the dataset 'bank', however, CARTE outperforms other baselines. In particular, the dataset is in line with the 51 datasets that contain both numerical and categorical features with relatively high cardinality of the latter. In a sense, CARTE is in the middle of both TabPFN (suitable for numerical features) and TabLLM (representing information as tokens), with an attentional architecture that has been designed to handle both numerical values and strings.

[45] `https://www.kaggle.com/datasets/anas123siddiqui/zomato-database?select=restaurant.csv`

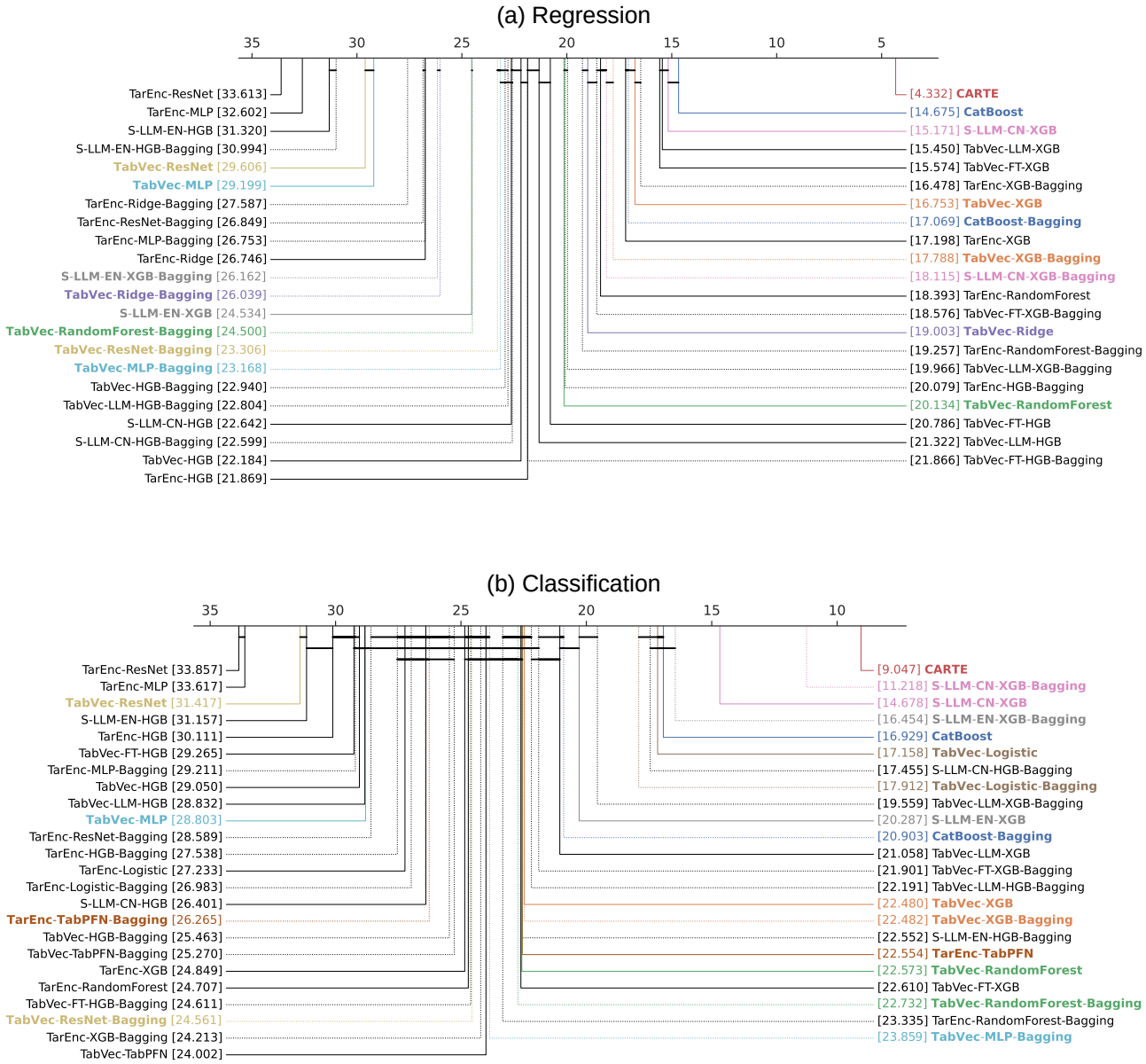


*Figure 9.* **Comparison of CARTE with 42 baselines on single tables.** The critical difference diagram for CARTE and 42 baselines for (a) regression (b) classification tasks. In addition to the methods in Figure 4 (with the same coloring), we include additional baselines with HistGradientBoosting, target encoding, and external information from language models. The figure shows that CARTE attains the pronounced lead to all the baselines for both regression and classification tasks. Moreover, the bagging strategy brings larger benefits to neural networks compared to linear or ensembling baselines, which suggests an important setups for other deep learning architectures.

*Table 5.* **Detailed results of TabLLM datasets.** Dataset specification and performance comparison among CARTE and three baselines presented in TabLLM (Hegselmann et al., 2023). The datasets contain high fraction of numerical features or categorical columns with low cardinality. As observed from the comparison results, TabPFN performs the best followed by CARTE, XGBoost, and TabLLM.

**TabLLM dataset specificiations**

| Datasets | Fraction of numerical columns | Average cardinality |
|---|---|---|
| bank | 0.40 | 38.44 |
| blood | 1.00 | 0.00 |
| calhousing | 1.00 | 0.00 |
| car | 0.00 | 3.50 |
| creditg | 0.33 | 3.79 |
| diabetes | 1.00 | 0.00 |
| heart | 0.45 | 2.67 |
| income | 0.33 | 12.38 |
| jungle | 1.00 | 0.00 |

**Performance comparison**

| Datasets | Methods | Number of Shots | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 32 | | 64 | | 128 | | 256 | | 512 | |
| bank | CARTE | **0.81** | **0.03** | 0.83 | 0.03 | **0.87** | **0.04** | 0.89 | 0.03 | 0.90 | 0.01 |
| | XGBoost | 0.76 | 0.03 | 0.83 | 0.02 | 0.85 | 0.03 | 0.88 | 0.01 | 0.90 | 0.01 |
| | TabPFN | 0.76 | 0.03 | 0.82 | 0.03 | 0.86 | 0.02 | 0.89 | 0.00 | 0.90 | 0.00 |
| | TabLLM | 0.64 | 0.06 | 0.69 | 0.03 | 0.82 | 0.05 | 0.87 | 0.01 | 0.88 | 0.01 |
| blood | CARTE | 0.68 | 0.01 | 0.68 | 0.01 | 0.72 | 0.02 | 0.72 | 0.0 | 0.71 | 0.01 |
| | XGBoost | 0.67 | 0.06 | 0.68 | 0.05 | 0.71 | 0.06 | 0.70 | 0.07 | 0.67 | 0.06 |
| | TabPFN | **0.70** | **0.04** | **0.73** | **0.04** | **0.75** | **0.04** | **0.76** | **0.04** | **0.76** | **0.03** |
| | TabLLM | 0.68 | 0.04 | 0.68 | 0.04 | 0.68 | 0.06 | 0.70 | 0.08 | 0.68 | 0.04 |
| calhousing | CARTE | 0.79 | 0.02 | 0.83 | 0.03 | 0.85 | 0.04 | 0.87 | 0.05 | 0.89 | 0.05 |
| | XGBoost | 0.79 | 0.04 | 0.82 | 0.04 | 0.87 | 0.01 | 0.90 | 0.01 | 0.92 | 0.01 |
| | TabPFN | **0.85** | **0.03** | **0.89** | **0.01** | **0.91** | **0.01** | **0.92** | **0.00** | **0.93** | **0.00** |
| | TabLLM | 0.77 | 0.08 | 0.77 | 0.04 | 0.81 | 0.02 | 0.83 | 0.01 | 0.86 | 0.02 |
| car | CARTE | 0.87 | 0.06 | 0.94 | 0.07 | 0.98 | 0.03 | 0.99 | 0.03 | 1.00 | 0.02 |
| | XGBoost | 0.82 | 0.03 | 0.91 | 0.02 | 0.95 | 0.01 | 0.98 | 0.01 | 0.99 | 0.01 |
| | TabPFN | **0.92** | **0.02** | **0.97** | **0.00** | **0.99** | **0.01** | **1.00** | **0.00** | 1.00 | 0.00 |
| | TabLLM | 0.91 | 0.02 | 0.96 | 0.02 | 0.98 | 0.01 | 0.99 | 0.00 | 1.00 | 0.00 |
| creditg | CARTE | 0.67 | 0.03 | 0.68 | 0.01 | 0.70 | 0.02 | 0.75 | 0.01 | **0.77** | **0.02** |
| | XGBoost | 0.66 | 0.03 | 0.67 | 0.06 | 0.68 | 0.02 | 0.73 | 0.02 | 0.75 | 0.03 |
| | TabPFN | 0.69 | 0.07 | 0.70 | 0.07 | **0.72** | **0.06** | 0.75 | 0.04 | 0.75 | 0.02 |
| | TabLLM | **0.72** | **0.06** | 0.70 | 0.07 | 0.71 | 0.07 | 0.72 | 0.03 | 0.72 | 0.02 |
| diabetes | CARTE | 0.76 | 0.06 | 0.79 | 0.02 | 0.81 | 0.01 | 0.82 | 0.01 | 0.81 | 0.00 |
| | XGBoost | 0.69 | 0.08 | 0.73 | 0.05 | 0.78 | 0.05 | 0.80 | 0.03 | 0.80 | 0.01 |
| | TabPFN | **0.77** | **0.03** | **0.82** | **0.03** | **0.83** | **0.03** | **0.83** | **0.03** | 0.81 | 0.02 |
| | TabLLM | 0.68 | 0.04 | 0.73 | 0.03 | 0.79 | 0.04 | 0.78 | 0.02 | 0.78 | 0.04 |
| heart | CARTE | 0.90 | 0.02 | 0.91 | 0.02 | 0.92 | 0.02 | **0.93** | **0.01** | **0.93** | **0.01** |
| | XGBoost | 0.88 | 0.04 | 0.91 | 0.01 | 0.91 | 0.01 | 0.90 | 0.01 | 0.92 | 0.01 |
| | TabPFN | **0.91** | **0.02** | **0.92** | **0.02** | 0.92 | 0.02 | 0.92 | 0.01 | 0.92 | 0.02 |
| | TabLLM | 0.87 | 0.06 | 0.91 | 0.01 | 0.90 | 0.01 | 0.92 | 0.01 | 0.92 | 0.01 |
| income | CARTE | 0.84 | 0.09 | 0.84 | 0.02 | 0.85 | 0.03 | 0.87 | 0.01 | 0.88 | 0.01 |
| | XGBoost | 0.79 | 0.03 | 0.82 | 0.02 | 0.84 | 0.01 | 0.87 | 0.01 | 0.88 | 0.00 |
| | TabPFN | 0.80 | 0.04 | 0.82 | 0.04 | 0.84 | 0.01 | 0.86 | 0.01 | 0.87 | 0.01 |
| | TabLLM | 0.84 | 0.01 | 0.84 | 0.02 | **0.86** | **0.01** | 0.87 | 0.00 | **0.89** | **0.01** |
| jungle | CARTE | 0.71 | 0.03 | 0.80 | 0.02 | 0.81 | 0.02 | 0.86 | 0.02 | 0.90 | 0.02 |
| | XGBoost | 0.78 | 0.03 | 0.81 | 0.02 | 0.84 | 0.02 | 0.87 | 0.01 | 0.91 | 0.01 |
| | TabPFN | 0.78 | 0.02 | 0.81 | 0.01 | 0.84 | 0.01 | **0.88** | **0.01** | 0.91 | 0.00 |
| | TabLLM | 0.71 | 0.02 | 0.78 | 0.02 | 0.81 | 0.02 | 0.84 | 0.01 | 0.89 | 0.01 |

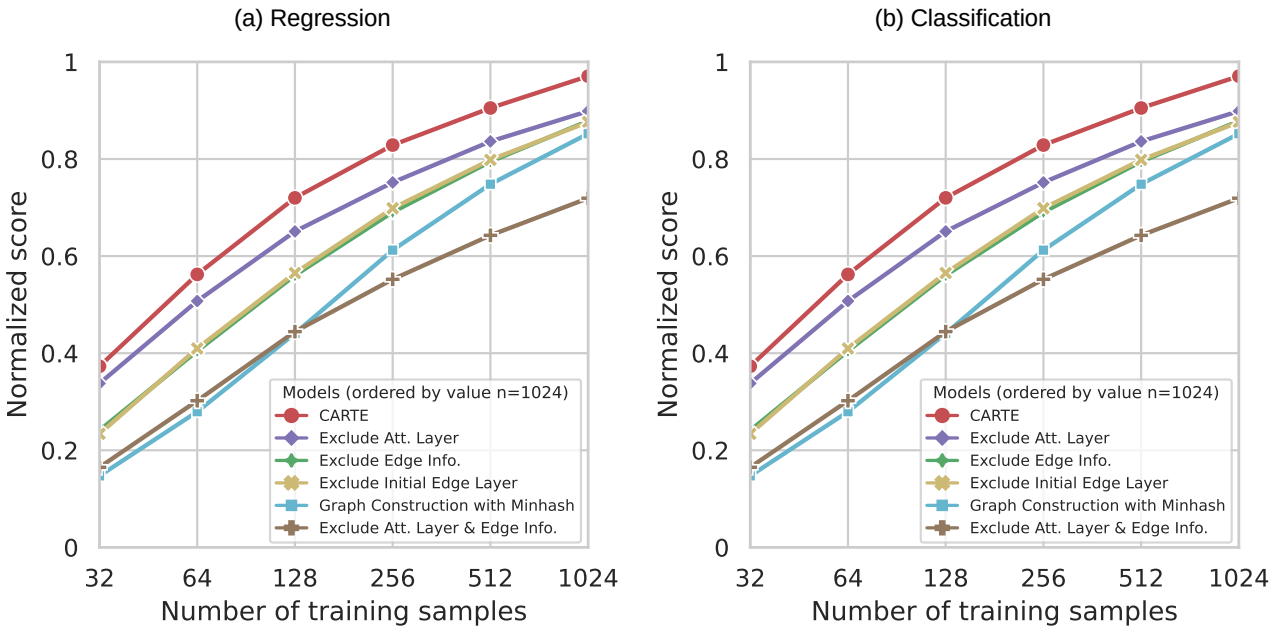


*Figure 10.* **Ablation on various components of CARTE.** The learning curves from the train size of 32 up to 1,024 on various cases excluding or switching the components. Each are crucial for gaining the performance of CARTE. In particular, there is significant decrease with the exclusion of edge information and the attention layer, which are essential for leveraging context within a given table. Moreover, the result with Minhash (A string-level representation without any semantic content, Cerda & Varoquaux, 2022) shows that language models are crucial for effectively using external information.

### C.3. Ablation Study on the Components of CARTE

To study the effect of various components of CARTE, we conducted additional experiments in which we exclude or change the associated components. Figure 10 shows the learning curves from the train size of 32 up to 1,024 for each case of excluding or switching the components. For the graph construction with Minhash, we change the feature initialization step with Skrub's Minhash encoder (Skrub, 2024), which encode string categorical features by applying the MinHash method to n-gram decompositions of strings. The figure shows that each are crucial for gaining the performance of CARTE. In particular, it is interesting to observe the significant decrease with the exclusion of edge information and the attention layer. Since both are essential for leveraging context within a given table, it implies that capturing context is pivotal for attaining the strong performances in predictions. Moreover, the performance gap between CARTE and Minhash confirm that string-level models, that also capture semantic similarity, is important and the use of language models are pivotal for effectively using external information, especially when information given in the table is limited.

### C.4. The Effect of Oversmoothing

Figure 11 show that representations extracted from deeper layers of the Graph Neural Networks (GNNs) are less useful for prediction on downstream tasks[46]. We interpret this as an effect of oversmoothing, a well known problem in GNNs (Chen et al., 2020a; Rusch et al., 2023).

[46]Here, they are used inside a HistGradientBoosting predictor from scikit-learn.

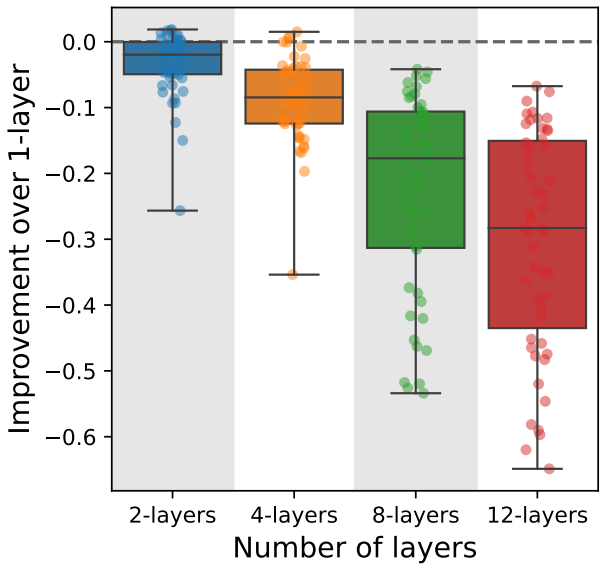


*Figure 11.* **The effect of oversmoothing**: comparing prediction from representations extracted from the GNN from the $2^{nd}$, $4^{th}$, $8^{th}$, and $12^{th}$ layers, with that build from the first layer. We see that the deeper we go in the GNN, the less useful the representation is for downstream task. We interpret this as an effect of oversmoothing.

### C.5. Details of the Entity Matching Experiment

The experiments were conducted with the same experiment settings as that of the singletable experiments. Table 6 gives the specific results behind each dataset used in the entity matching experiment Figure 5. The specific datasets are

- CE = Company employees : 32% of the companies matched to YAGO
- MV = movie revenues : 84% of the movies matched to YAGO
- US-Acc = US accidents : 67% of the cities matched to YAGO
- US-Elec = US elections : 98% of the counties matched to YAGO

### C.6. Schema-matching Results on Joint Learning Across Multiple Tables

Figure 12 gives a direct comparison of performance between CARTE with and without schema-matching over 275 different cases in number of source data (ranging from one source to five sources). Each point represents a comparison of the average score over a dataset with different train/test split for a given size of train data. If a point is located below the diagonal line, it indicates a higher performance of x-axis. The figures shows that dots align along the diagonal line, indicating similar performance between CARTE with and without schema-matching (also with $p$-value of 0.728 for two-sided t-test on difference in means). The results suggest schema-matching is not required for CARTE on transfer across multiple tables.

*Table 6.* **Detailed results of the entity matching experiment**: individual scores on each dataset. The abbreviations are as follows: O-Original entries, M-Matched entries, R-Reduced dataset, and F-Full dataset.

| | **CatBoost-MR** | **CatBoost-MF** | **CatBoost-OR** | **CatBoost-OF** |
|---|---|---|---|---|
| CE-32 | 0.673 0.036 | 0.672 0.063 | 0.683 0.052 | 0.668 0.062 |
| CE-64 | 0.707 0.021 | 0.72 0.013 | 0.702 0.025 | 0.718 0.017 |
| CE-128 | 0.734 0.007 | 0.739 0.024 | 0.731 0.011 | 0.745 0.01 |
| CE-256 | 0.739 0.008 | 0.747 0.014 | 0.74 0.009 | 0.749 0.008 |
| CE-512 | 0.744 0.005 | 0.758 0.005 | 0.744 0.005 | 0.758 0.004 |
| CE-1024 | 0.752 0.006 | 0.763 0.004 | 0.752 0.006 | 0.764 0.003 |
| MV-32 | 0.4 0.058 | 0.398 0.049 | 0.394 0.058 | 0.403 0.042 |
| MV-64 | 0.436 0.043 | 0.449 0.045 | 0.426 0.059 | 0.453 0.035 |
| MV-128 | 0.484 0.027 | 0.492 0.028 | 0.482 0.018 | 0.495 0.019 |
| MV-256 | 0.511 0.011 | 0.515 0.017 | 0.51 0.017 | 0.523 0.012 |
| MV-512 | 0.545 0.007 | 0.55 0.007 | 0.545 0.009 | 0.552 0.008 |
| MV-1024 | 0.559 0.007 | 0.574 0.007 | 0.563 0.005 | 0.573 0.005 |
| US-Acc-32 | -0.016 0.063 | -0.018 0.064 | -0.023 0.076 | -0.02 0.058 |
| US-Acc-64 | 0.007 0.055 | -0.01 0.069 | 0.028 0.036 | -0.023 0.087 |
| US-Acc-128 | 0.082 0.026 | 0.055 0.029 | 0.084 0.028 | 0.057 0.026 |
| US-Acc-256 | 0.129 0.018 | 0.089 0.031 | 0.129 0.015 | 0.089 0.025 |
| US-Acc-512 | 0.163 0.02 | 0.12 0.022 | 0.163 0.021 | 0.121 0.02 |
| US-Acc-1024 | 0.214 0.007 | 0.157 0.01 | 0.217 0.005 | 0.155 0.009 |
| US-Elec-32 | 0.31 0.133 | 0.318 0.142 | 0.34 0.118 | 0.285 0.161 |
| US-Elec-64 | 0.433 0.062 | 0.441 0.068 | 0.449 0.038 | 0.445 0.056 |
| US-Elec-128 | 0.512 0.019 | 0.505 0.02 | 0.511 0.02 | 0.51 0.02 |
| US-Elec-256 | 0.547 0.009 | 0.543 0.011 | 0.546 0.009 | 0.544 0.011 |
| US-Elec-512 | 0.572 0.007 | 0.571 0.01 | 0.57 0.009 | 0.571 0.008 |
| US-Elec-1024 | 0.586 0.004 | 0.586 0.006 | 0.586 0.005 | 0.587 0.005 |

| | **CARTE-MR** | **CARTE-MF** | **CARTE-OR** | **CARTE-OF** | **KEN-R** | **KEN-F** |
|---|---|---|---|---|---|---|
| CE-32 | 0.699 0.023 | 0.69 0.034 | 0.693 0.023 | 0.692 0.029 | 0.518 0.11 | 0.459 0.119 |
| CE-64 | 0.729 0.019 | 0.744 0.025 | 0.733 0.01 | 0.747 0.022 | 0.612 0.077 | 0.434 0.284 |
| CE-128 | 0.755 0.007 | 0.763 0.012 | 0.755 0.01 | 0.763 0.014 | 0.708 0.019 | 0.409 0.305 |
| CE-256 | 0.762 0.009 | 0.776 0.01 | 0.763 0.008 | 0.781 0.007 | 0.738 0.02 | 0.368 0.812 |
| CE-512 | 0.773 0.011 | 0.785 0.007 | 0.778 0.012 | 0.789 0.008 | 0.757 0.006 | 0.267 0.494 |
| CE-1024 | 0.783 0.008 | 0.793 0.006 | 0.787 0.01 | 0.798 0.006 | 0.772 0.008 | 0.486 0.301 |
| MV-32 | 0.3 0.057 | 0.313 0.083 | 0.329 0.066 | 0.322 0.095 | 0.301 0.057 | 0.318 0.033 |
| MV-64 | 0.452 0.044 | 0.471 0.027 | 0.458 0.025 | 0.461 0.038 | 0.42 0.035 | 0.369 0.055 |
| MV-128 | 0.521 0.022 | 0.523 0.022 | 0.519 0.023 | 0.515 0.018 | 0.493 0.024 | 0.384 0.08 |
| MV-256 | 0.556 0.022 | 0.562 0.02 | 0.554 0.021 | 0.555 0.014 | 0.543 0.014 | 0.464 0.089 |
| MV-512 | 0.594 0.013 | 0.597 0.013 | 0.595 0.014 | 0.595 0.011 | 0.589 0.012 | 0.517 0.04 |
| MV-1024 | 0.62 0.008 | 0.622 0.008 | 0.62 0.008 | 0.618 0.009 | 0.616 0.007 | 0.544 0.032 |
| US-Acc-32 | 0.061 0.054 | 0.053 0.055 | 0.054 0.067 | 0.048 0.045 | 0.062 0.094 | 0.047 0.045 |
| US-Acc-64 | 0.112 0.057 | 0.114 0.046 | 0.122 0.056 | 0.105 0.051 | 0.146 0.038 | 0.051 0.09 |
| US-Acc-128 | 0.155 0.06 | 0.136 0.053 | 0.16 0.058 | 0.14 0.051 | 0.175 0.025 | 0.117 0.026 |
| US-Acc-256 | 0.225 0.024 | 0.197 0.023 | 0.232 0.02 | 0.2 0.024 | 0.225 0.029 | 0.152 0.014 |
| US-Acc-512 | 0.278 0.008 | 0.237 0.01 | 0.275 0.015 | 0.235 0.014 | 0.27 0.012 | 0.173 0.029 |
| US-Acc-1024 | 0.303 0.01 | 0.263 0.008 | 0.304 0.008 | 0.265 0.012 | 0.298 0.004 | 0.205 0.006 |
| US-Elec-32 | 0.387 0.082 | 0.387 0.083 | 0.393 0.08 | 0.393 0.082 | 0.149 0.193 | 0.209 0.159 |
| US-Elec-64 | 0.467 0.031 | 0.467 0.032 | 0.465 0.021 | 0.465 0.022 | 0.432 0.089 | 0.454 0.073 |
| US-Elec-128 | 0.52 0.021 | 0.52 0.021 | 0.52 0.022 | 0.52 0.022 | 0.564 0.038 | 0.571 0.035 |
| US-Elec-256 | 0.552 0.011 | 0.553 0.011 | 0.546 0.013 | 0.546 0.013 | 0.625 0.019 | 0.628 0.011 |
| US-Elec-512 | 0.586 0.008 | 0.586 0.008 | 0.58 0.007 | 0.581 0.007 | 0.667 0.006 | 0.664 0.01 |
| US-Elec-1024 | 0.615 0.007 | 0.615 0.007 | 0.612 0.007 | 0.613 0.007 | 0.7 0.009 | 0.697 0.005 |

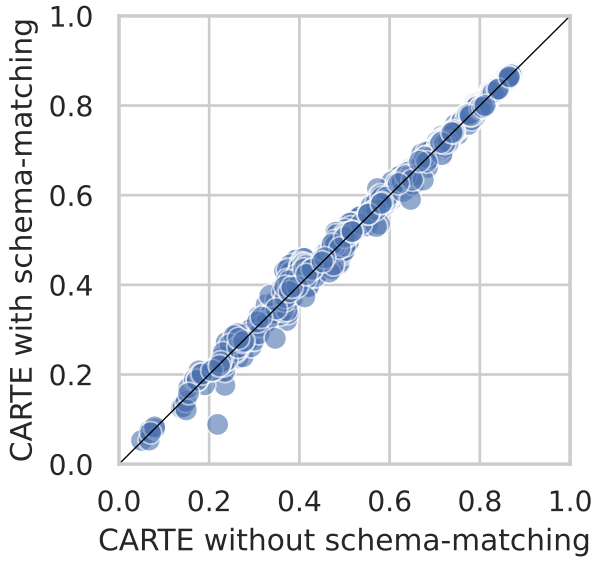


*Figure 12.* **Performance comparison of CARTE with and without schema-matching**: The figures portrays a direct comparison of performance between CARTE with and without schema-matching over 275 different cases in number of source data. A point below the diagonal line indicate better performance of the method in x-axis. We see that the dots align along the diagonal line, showing similar performance both approaches of CARTE on joing learning ($p$-value of 0.728). The results bolster no schema-matching for CARTE.