IngesTables: Scalable and Efficient Training of LLM-Enabled Tabular Foundation Models

Scott Yak, Yihe Dong, Javier Gonzalvo, Sercan Ö. Arık
Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
{scottyak,yihed,xavigonzalvo,soarik}@google.com

Abstract

There is a massive amount of tabular data that can be taken advantage of via 'foundation models' to improve prediction performance for downstream tabular prediction tasks. However, numerous challenges constitute bottlenecks in building tabular foundation models, including learning semantic relevance between tables and features, mismatched schemes, arbitrarily high cardinality for categorical values, and scalability to many tables, rows and features. We propose IngesTables, a novel canonical tabular foundation model building framework, designed to address the aforementioned challenges. IngesTables employs LLMs to encode representations of table/feature semantics and the relationships, that are then modeled via an attention-based tabular architecture. Unlike other LLM-based approaches, IngesTables is much cheaper to train and faster to run inference, because of how LLM-generated embeddings are defined and cached. We show that IngesTables demonstrates significant improvements over commonly-used models like XGBoost on Clinical Trial datasets in standard supervised learning settings, and is competitive with tabular prediction models that are specialized for Clinical Trial datasets without incurring LLM-level cost and latency. Moreover, the transfer learning enabled by IngesTables is prominent – on Airbnb dataset, we demonstrate that IngesTables yields significant improvements when pretrained on large-scale relevant data, even with partially-overlapping features.

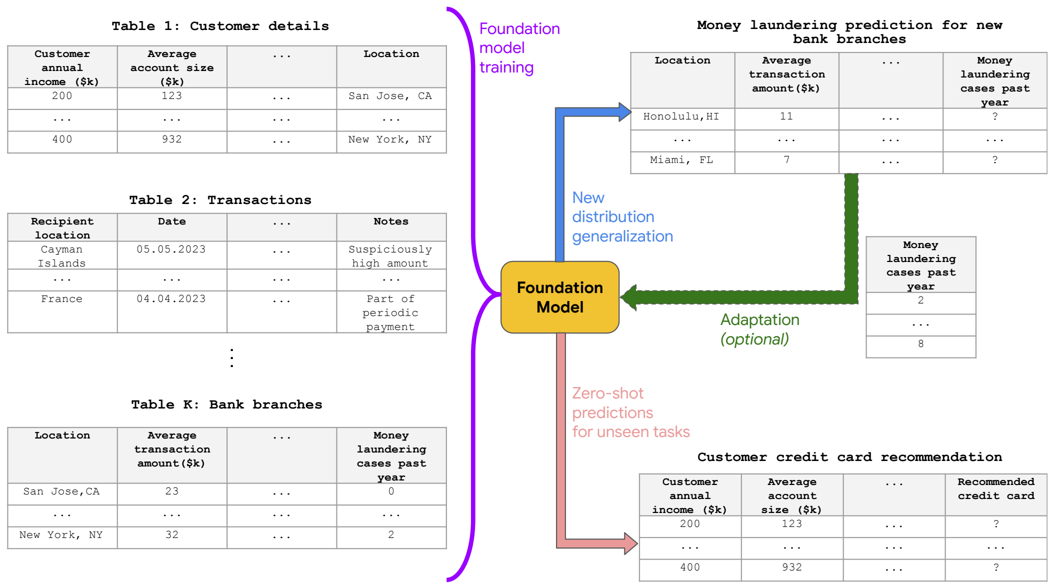
1 Introduction

A foundation model broadly refers to a large machine learning model trained on a large amount of data at scale such that it can be adapted to a wide range of downstream tasks with unseen datasets or tasks. Foundation models have been highly successful for image and text data [1], playing a major role in many state-of-the-art achievements on downstream tasks such as object detection [2] and question answering [3]. A key element to their success is pretraining of a high-capacity model (e.g., ViT, BERT) on vast amounts of data (e.g., JFT for images, English CommonCrawl for text).

Despite being the most common data type in real-world machine learning [4], tabular data has not been "conquered" by deep learning, let alone foundation models. Tree-based methods remain at the top of several benchmarks for tabular learning tasks [5, 6]. Efficient representation learning and correspondingly information transfer across relevant datasets and tasks are not among the capabilities enabled by such tree based models, however. We postulate that harnessing relevant information from other datasets has significant potential for tabular data, and achieving such breakthrough requires substantial rethinking in how deep learning should be designed for foundation model building for tabular data.

Table Representation Learning Workshop at NeurIPS 2023.

Transfer learning across tabular datasets is fundamentally challenging because the schemas, statistics, and tasks for the datasets can vary wildly – numeric features may be of different scales, categorical values may have different numbers of classes and class balance, and the total number of features can also vary. Moreover, even between two closely related tasks, there would be often be substantial differences in the schemas and distributions that it is unclear how exactly the model could "map" a feature from one dataset to the corresponding related feature in another dataset in the hopes of finding some common pattern that can be jointly learned.



Foundation model training
New distribution generalization
Foundation Model
Adaptation (optional)
Zero-shot predictions for unseen tasks

Table 1: Customer details

Customer annual income ($k)	Average account size ($k)	...	Location
200	123	...	San Jose, CA
...	...	...	...
400	932	...	New York, NY

Table 2: Transactions

Recipient location	Date	...	Notes
Cayman Islands	05.05.2023	...	Suspiciously high amount
...	...	...	...
France	04.04.2023	...	Part of periodic payment

⋮

Table K: Bank branches

Location	Average transaction amount($k)	...	Money laundering cases past year
San Jose,CA	23	...	0
...	...	...	...
New York, NY	32	...	2

Money laundering prediction for new bank branches

Location	Average transaction amount($k)	...	Money laundering cases past year
Honolulu,HI	11	...	?
...	...	...	...
Miami, FL	7	...	?

Money laundering cases past year
2
...
8

Customer credit card recommendation

Customer annual income ($k)	Average account size ($k)	...	Recommended credit card
200	123	...	?
...	...	...	...
400	932	...	?

Figure 1: `IngesTables` framework. Tables of different schemas, sizes and features are used to build the tabular foundation model, which is capable of generating predictions for unseen samples from different distributions or tasks. Adaptation of the foundation model can be task specific, using some amount of data from the target domain.

Prior work has established that positive information transfer *is* possible even when there is only partial overlap between features [7], though their experiments were limited to partitions of the same dataset and task, and have not truly demonstrated cross-dataset cross-task transfer. Pushing this literature, particularly with the real-world modeling challenges and scalability motivations, `IngesTables`, shown in Fig. 1 demonstrate that by harnessing the power of large language models (LLMs) efficiently, with minimal overhead, transfer learning is also possible even in the challenging setting where the datasets and tasks are different. In addition to enabling the information transfer across datasets, LLMs also unlock the capability of bringing prior knowledge to tabular datasets – rather than learning the embeddings for categorical features from scratch, LLMs can readily map them to meaningful representations utilizing their pretraining datasets. For example, for a feature like "product category" given a demand prediction task, an LLM would already know the semantic relationships between different product categories that can be very useful for the task of interest. On the other hand, naive employment of LLMs, such as to encode entire tabular data as text, would not be an optimal direction as we demonstrate, since LLMs are not pretrained with tabular data (their training mixtures are heavily dominated by free-form text), and they would be much more expensive to train, and much slower and costlier to serve, particularly compared to the commonly-employed tabular models for real-world tasks. The design of `IngesTables` is motivated with these real-world cost and latency motivations on top of mind as well while utilizing LLMs.

Overall, the contributions can be summarized as:

- We propose a novel schema-independent architecture, `MapTransformer`, designed to enable knowledge transfer across tabular datasets.
- To train the `MapTransformer`, we propose the `IngesTables` learning framework, which treats each row of a tabular dataset as a set of (key embedding, value embedding) tuples. This enables

the same model weights to be trained (or finetuned) on multiple tabular datasets with different schemas, then used for inference on unseen tabular datasets with unencountered schemas.

- We harness the expressiveness of LLMs to include task-context into the feature key embeddings, and the inclusion of feature-context into the feature value embeddings. This allows the model to disambiguate feature keys in different contexts, while enabling LLM embeddings to be aggressively cached and reused, so that the required number of LLM forward passes only scales with the number of features and unique categories, not with the dataset size.

2 Related Work

Various deep learning models have been explored recently for tabular data, with the goal of outperforming commonly-used ensemble tree based models [8]. TabNet [9] is one popular attention-based architecture for tabular data that has a schema-agnostic input preprocessing step, treats each row as a sequence, and demonstrates the use of self-supervised learning to improve the model with unlabeled data. SAINT [10] proposes attention over both rows and columns, along with contrastive self-supervised pretraining. TabTransformer [11] harnesses the permutation equivariance of Transformers for tabular datasets. In general, such deep learning models have shown the premise for improved representation learning from row data, particularly with attention based architectures well tailored for the tabular data type. However, they do not contain the necessary components to be used as foundation models – they cannot handle mismatched schemas across datasets effectively; they are not designed to be trained from many datasets in a scalable manner, and they use randomly initialized feature and category embedding weights, hindering transfer learning. There are also data agnostic improved attention based modeling approaches like SetTransformer [12] and PerceiverIO [13], that can also be applied to tabular data, but in its pursuit of generality, they miss some symmetries and invariances specific to tabular datasets.

In an attempt to bring extra prior knowledge, LLMs have been explored for tabular data. TabLLM [14] fine-tunes pretrained LLMs to make predictions on tabular datasets, while OptFormer [15] uses a similar approach for optimization. However, utilizing LLMs for raw tabular data has the fundamental disadvantage that LLMs are pretrained with text data, that are quite different from tabular data. In addition, naive employment of LLMs would incur orders of magnitude cost and latency in most cases, for both training and serving, compared to commonly-used tabular models.

Along the direction of enabling information transfer across tabular datasets, TransTab [7] demonstrates positive transfer from source to training task, converting each sample to a generalizable embedding vector and harnessing the feature names' semantic information. However, this method has not been extended beyond the standard transfer learning setting to build foundation models that can perform predictions for multiple tabular datasets. TabPFN [16] proposes a pretrained transformer that is designed to perform well on a large variety of tabular tasks. However, by treating the entire dataset as the input sequence, it prevents the model from scaling to larger datasets. MediTab [17] leverages LLMs to consolidate tabular samples to handle tables with varying schema and align out-domain data with the target task using a 'learn, annotate, and audit' pipeline. However, it requires the number of LLM output tokens to scale linearly with the size of the dataset, which can be cost-prohibitive for large datasets, as we show. Moreover, as the numeric feature values are also treated as text, MediTab cannot benefit much from caching of LLM outputs whenever there are floating point values in a dataset, requires at least one forward pass of the LLM for every row at inference time, making the latency of this approach prohibitively high for most real-world use cases.

3 The `IngesTables` Framework

Fig. 1 depicts the overall `IngesTables` framework, that is designed to learn from many different tables to obtain superior transfer learning as well as unseen distribution and task generalization. To address aforementioned challenges towards these goals, we propose the `MapTransformer` architecture in `IngesTables`, that inputs the information coming from the judiciously-designed Input Preprocessor. Next, we describe the major constituents of `IngesTables`, summarized by Fig. 2.

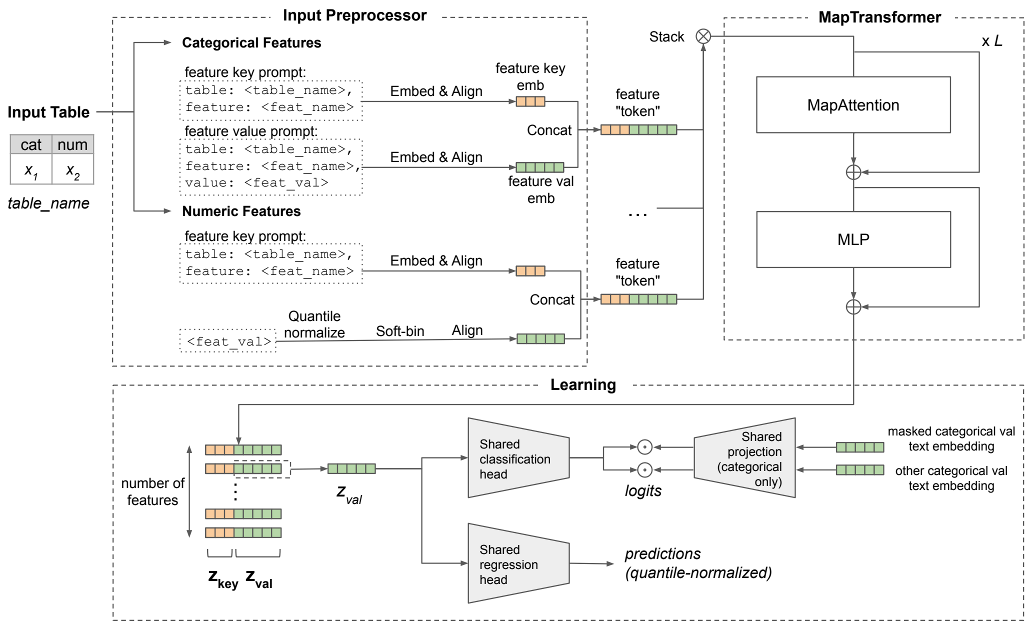


Figure 2: `IngesTables` learning setup. The Input Preprocessor converts each row of an input table into a sequence of feature embeddings, which is then fed into `MapTransformer`. Note that no positional encoding is added. The `MapTransformer` then transforms each feature embedding into another embedding space, which is then used to perform classification or regression.

3.1 Input preprocessor

Each example is treated as a set of key-value tuples (or a map), with the specific order of the key-value tuples not mattering. Within an example, all feature key-value tuples are converted into feature embedding vectors, then stacked along a new axis into a 2D array. Note that positional encodings are not added, as permutation equivariance is desirable.

To ensure low training cost and mitigate overfitting in small data regimes, we propose using a frozen embedding LLM (we use the GTE-large [18] model) to encode entire feature keys and values into a embedding space while including the task context, instead of converting each row into a natural language or text representation to be fed into an LLM. We propose quantile normalization for numeric features values instead of directly passing into the LLM as the numerical data coverage would often be low in pretraining mixtures of LLMs. Unlike recent approaches that consider LLMs for tabular data, the embedding LLM itself is fixed, which allows the embedding outputs to be cached, thereby allowing the LLM inference costs associated with the pretraining process to scale with the metadata size, instead of the dataset size. Moreover, unlike other approaches, there are no dataset-specific, task-specific, or feature-specific weights to keep track of, which makes transfer learning across datasets with differently-named-but-semantically-related features possible.

To encode feature keys (whether categorical or numerical), we insert the task context (which could be a semantically meaningful table name or a task description) and feature name into a prompt template, and pass this prompt through an embedding LLM to obtain the feature key embedding. Note that we only need to do this once for each feature for each dataset, since we can cache the LLM outputs.

For simplicity, the preprocessing logic assumes that all string features are categorical features, and all floating point values are numeric features. This means that before running this input preprocessing step, the user needs to make sure that numeric features are cast to floats, and categorical features are cast to strings. Specifically, id-type features, which are often stored as integers, must be cast to strings to be correctly handled as categorical features. We defer the handling of other types of features such as date/time and geolocation to future work.

To encode categorical values, we insert the table name, feature name, and feature value into a prompt template, and pass this prompt through an embedding LLM to obtain the feature value embedding.

Again, due to caching, we only need to do this once for each unique category for each dataset. This approach also gracefully handles out-of-vocabulary categories as it would use the semantic meaning of the category name.

To encode numerical values, we propose quantile normalization, followed by soft-binning it by taking the softmax over the negative squared distance to each bin center, uniformly spaced apart on the unit interval. For efficiency, the quantiles can be estimated using a random sample of examples. In our experiments, we use 100 quantiles and 100 bins. By quantile normalization, we first map from the real line to the unit interval, then by soft-binning, we map from the unit interval to a point on a 100-dimensional L_1 unit sphere. The reason why we don't simply use an LLM to embed numeric values is that the number of unique numeric values tends to scale with the dataset size, which means that we would need an LLM forward pass for every training example, which doesn't scale.

Finally, since the numeric encoding does not have the same dimensions as the text embeddings, we use a dense layer to ensure they end up with the same dimensions. All feature key embeddings and feature value embeddings are then concatenated and stacked.

It is worth emphasizing that despite the extensive use of LLM embedding models, our design allows for aggressive caching, thus the number of LLM forward passes scales with the number of features and unique categorical values, not the dataset size, which is typically orders of magnitude larger.

Even though different tables may have different numbers of features, Transformer architectures are perfectly capable of handling variable-length input dimensions. Indeed, we propose using a Transformer-like architecture.

3.2 `MapTransformer`: An attentive schema-independent architecture

We propose `MapTransformer`, an attention-based encoder model that interleaves MLP blocks with Self-Attention layers, but with the Self-Attention layers replaced with the `MapAttention` layers (see Fig. 3). The main difference between Self-Attention and `MapAttention` is that the feature value embeddings are not used to compute attention weights. This way, a feature value embedding can be masked out without masking out the corresponding feature key embedding. Thus, the feature key embedding can be used to compute attention weights used to reconstruct the masked feature value using other feature values during masked input prediction. This attention layer is stacked to define a new Transformer architecture. This allows the use of masked input prediction as a way to perform self-supervised learning on unlabelled tabular datasets in a scalable way.

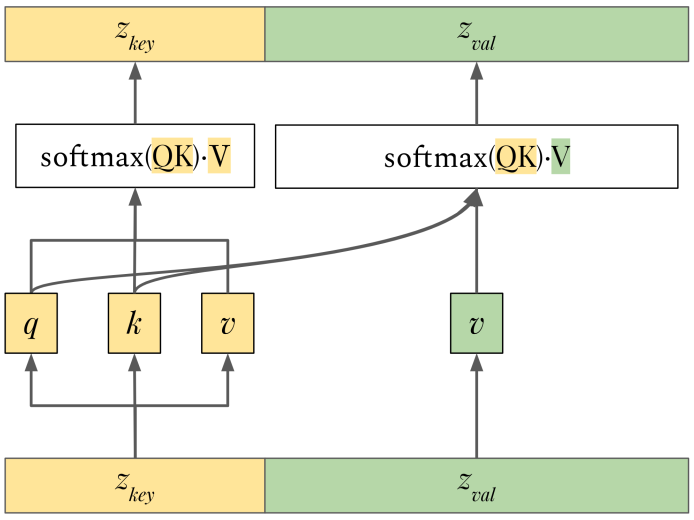


Figure 3: `MapAttention` layer. The user can mask out a feature value embedding without masking out the corresponding feature key embedding. Thus, the feature key embedding can be used to compute attention weights used to reconstruct the masked feature value using other feature values during masked input prediction.

Note that if there is only one MapAttention layer, it would not be different from a learned convex sum of feature value embeddings, since the attention weights cannot vary according to the feature values. But in the next MLP block, the z_k embeddings and z_v embeddings are mixed together, which means that the attention weights in the subsequent MapAttention layers can vary with the feature value embeddings in the inputs.

3.3 Learning objective

To enable information extraction from a variety of datasets (that might be labeled or unlabeled for the given tasks), we propose self-supervised pretraining for MapTransformer, as commonly used for language modeling. At each training step, we randomly mask out some feature values in the input, and use the output embeddings of the MapTransformer to reconstruct the masked input. In tabular modeling, the output space corresponding to each position of the output sequence is not fixed (unlike language modeling, where the output space is always a fixed set of word piece tokens). Each position represents a feature, which could be a categorical or a numeric feature. A naive approach would be to maintain an embedding table for each categorical feature, but this would (i) result in a massive embedding table and an unwieldy number of prediction heads when we attempt to ingest many diverse datasets, and (ii) preclude transfer learning. Another approach would be to directly minimize some distance between the masked and the output embedding, but this would suffer from the curse of dimensionality and embeddings not being trained with a distance-based objective. We focus on hybrid contrastive learning and regression objective, as summarized in Fig. 2.

For categorical features, the output layer's z_v embedding is used to classify the masked categorical feature value. As shown in Fig. 2, we grab the embedding from the corresponding position, and pass it through a Shared Classification Head to obtain z. On the right side, we grab the masked categorical feature value, and we find one other category for that feature within a batch, compute their text embeddings using the same embedding LLM, and pass them through a Shared Projection. We denote the output vector for the "correct" category as z^+, and the output vector for the "incorrect" category as z^-. The loss to be minimized is $l = -\log(\exp z \cdot z^+ / (\exp z \cdot z^+ + \exp z \cdot z^-))$. Intuitively, we can think of this as minimizing the softmax cross-entropy loss where $z \cdot z^+$ is the likelihood of the correct prediction, and $z \cdot z^-$ is the likelihood of the incorrect prediction. Alternatively, we can think of this as maximizing the log probability of making the correct prediction.

For numeric features, the output layer's z_v embedding is used to reconstruct the input value. We use Shared Regression Head to map the z_v embedding to a scalar, and minimize the mean squared error from the quantile normalized input value. Note that since the inputs were already quantile normalized, we do not need to worry about different numeric feature being on different scales, or having different skewness.

It is worth emphasizing that there are no per-task or per-dataset prediction heads – all weights are shared. The only additional parameters that scale with the number of datasets is the quantiles for the numeric features. Due to the flexibility of this learning setup, the same set of MapTransformer weights can be trained on multiple datasets. For simplicity, we select the training batch from the available datasets in a round-robin fashion.

4 Experiments

4.1 Datasets

Clinical Trials Outcome Dataset: Following [19, 17], we the use clinical trial data that contain drug, disease, and eligibility information for 17K clinical trials, that constitute the target task. The task is binary classification for the trial outcome, whether it is success or failure. To build the foundation model, pretraining is performed on the trial database containing 220K clinical trials with information about the trial setup (such as title, phase, enrollment, conditions, etc.) from 'ClinicalTrials.gov'. Both datasets cover phases I, II, and III.

Airbnb Datasets: To demonstrate IngesTables's knowledge transfer capabilities, we experiment with Airbnb-2019 [20] and Airbnb-2023 [21] datasets. Seven common-sensible features are used from each dataset, such as "room_type" and "number_of_reviews. Six features are common to both datasets, and the task is to predict the rental price of a given Airbnb unit. Airbnb-2019 contains 49K samples, and Airbnb-2023 contains 10K samples. Thus, Airbnb-2023 contains a distribution shift from Airbnb-2019 both temporally and in terms of which features are present. We experiment with various experimental settings to illuminate `IngesTables`'s knowledge transfer capabilities.

4.2 Results

Table 1: Test performances on the Clinical Trial Outcome Datasets. We report the mean of three runs with different random seeds for the `IngesTables` figures, while the rest of the figures are taken from [17], which does not averaged results over multiple runs. Note that similar to MediTab, `IngesTables` is only trained as a single model for all three datasets; unlike HINT, SPOT, and the rest, the `IngesTables` model is not individually tuned for each dataset.

Trial Data	Metrics	XGBoost	FFNN	DeepEnroll	COMPOSE	HINT	SPOT	MediTab	`IngesTables`
Phase I	AUROC	0.518	0.550	0.575	0.571	0.576	0.660	0.699	0.647 ± 0.009
	PRAUC	0.513	0.547	0.568	0.564	0.567	0.689	0.726	0.683 ± 0.007
Phase II	AUROC	0.600	0.611	0.625	0.628	0.645	0.630	0.706	0.655 ± 0.003
	PRAUC	0.586	0.604	0.600	0.604	0.629	0.685	0.733	0.700 ± 0.002
Phase III	AUROC	0.667	0.681	0.699	0.700	0.723	0.711	0.734	0.720 ± 0.006
	PRAUC	0.697	0.747	0.777	0.782	0.811	0.856	0.881	0.874 ± 0.003

Table 2: Transfer learning experiments for `IngesTables` using Airbnb-2019 for pretraining and Airbnb-2023 for fine tuning, which have different sets of features. We focus on the MSE metric on the test set (lower is better). The goal is to test how knowledge can be transferred from Airbnb-2019 to the Airbnb-2023 dataset, which exhibits a distribution shift, under three settings: warm start adaptation (pretraining on 2019 and finetuning on 2023), cold start standard training on 2019 only, and zero-shot evaluation on the 2023 dataset using a checkpoint pretrained on the 2019 data. These results show that pretraining on 2019 notably improves the training outcome on 2023, and that the zero-shot performance for a 2019-pretrained checkpoint on 2023 actually exceeds standard training on 2023 directly. In other words, `IngesTables`is able to positively transfer knowledge from one dataset to another dataset with a distribution shift.

Pretraining data	Tuning data	Test data	Test MSE
Airbnb-2019	Airbnb-2023	Airbnb-2023	0.0996 ± 0.011
	Airbnb-2023	Airbnb-2023	0.123 ± 0.018
Airbnb-2019		Airbnb-2023	0.101 ± 0.010

Table 1 overviews the comparisons of different modeling approaches on Clinical Trials Outcome. We note that `IngesTables` significantly outperforms generic tabular learning models like XGBoost and FFNN baselines by $15 - 20\%$ on all three tasks, and is competitive with deep neural network architectures with preprocessing setups that specialize on medical trial datasets (DeepEnroll, COMPOSE, HINT, and SPOT). On the other hand, our model underperforms MediTab, a fine-tuned BioBERT model that makes extensive use of GPT-3.5 for its preprocessing step. Despite this, as we will discuss in our next section, once cost and latency is considered, the `IngesTables` approach is still overall a more practical choice.

Table 2 shows the transfer learning capability of `IngesTables` with experiments on the Airbnb dataset. We observe that even with partially overlapping features between pretraining and tuning data, there is strong information transfer enabled by `IngesTables`. Interestingly, the zero-shot performance from the large-scale pretraining can even outperform standard tuning setting with smaller-scale data, indicating the strength of `IngesTables` in taking advantage of the information from large-scale pretraining data.

4.3 Latency and cost comparisons with other LLM-based approaches

Other LLM-based approaches have the fundamental drawback of employing LLM calls at every step, that causes significantly high training cost and inference latency. To further highlight the importance of our particular design choices and computational optimizations, we illustrate key inference latency and training cost comparison aspects below.

When comparing experimental results with MediTab, it is important to note that MediTab's input preprocessing requires at least 5 forward passes of the LLM for every example during training (not to mention the retries when the generated output fails the sanity check), and at least one forward pass of the LLM for every example during inference. The number of tokens decoded by the LLM for each inference call is also proportional to the number of features. To illustrate our point concretely, an inference call on MediTab for an example with 100 features would require decoding roughly 400 tokens, it would take a staggering $\sim$10 seconds *for each example* for GPT 3.5 decoding! By contrast, once `IngesTables` is done with the input preprocessing it needs to do for training, at inference, unless there are out-of-vocabulary categorical features encountered, `IngesTables` only needs to query its prepopulated vector store to prepare its inputs for a forward pass of `MapTransformer`, whose latency is negligible compared to that of an LLM.

To exemplify the cost difference, consider 10 datasets, each with 10 million rows and 100 features (50 categorical with up to 100 categories, 50 numeric), assuming each cell contains 4 tokens, and using the pricing page from OpenAI [22]. We compare three ways of using LLMs:

- Decoder-only LLM (GPT-3.5 Turbo 4K): $10 \cdot 10$ million $\cdot 100 \cdot 4 \cdot$ (\$0.002 / 1000) = \$80,000
- Embedding model (Ada v2, no caching): $10 \cdot 10$ million $\cdot 100 \cdot 4 \cdot$ (\$0.0001 / 1000) = \$4000
- Embedding model (Ada v2, with caching): $10 \cdot (100 \cdot 4 + 50 \cdot 100) \cdot$ (\$0.0001 / 1000) < \$0.01

Such significant difference highlights the importance of employing embedding models to encode key table attributes with caching, as the fundamental design choice of `IngesTables`.

5 Conclusion

We propose a novel architecture, `MapTransformer`, that is designed to be trained with the `IngesTables` learning setup on multiple datasets with different schemas. We present promising experimental results on a clinical trial dataset, and demonstrate its superior efficiency and scalability in terms of the number of LLM forward passes needed.

We leave more demonstrations of the proposed `MapTransformer` for the scenarios of adaptation to new tasks and domains, as well as scaling to many pretraining datasets, to future work.

References

[1] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2022.

[2] Zhuofan Zong, Guanglu Song, and Yu Liu. Detrs with collaborative hybrid assignments training, 2023.

[3] Hamid Gharagozlou, Javad Mohammadzadeh, Azam Bastanfard, and Saeed Shiry Ghidary. RLAS-BIABC: A reinforcement learning-based answer selection using the BERT model boosted by an improved ABC algorithm. *Computational Intelligence and Neuroscience*, 2022:1–21, may 2022.

[4] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, et al. Notes from the ai frontier. *McKinsey Global Institute*, 4 2018.

[5] Vadim Borisov, Tobias Leemann, Kathrin Sessler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022.

[6] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022.

[7] Zifeng Wang and Jimeng Sun. Transtab: Learning transferable tabular transformers across tables, 2022.

[8] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022.

[9] Sercan O. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning, 2020.

[10] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training, 2021.

[11] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings, 2020.

[12] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks, 2019.

[13] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, Olivier Hénaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver IO: A General Architecture for Structured Inputs & Outputs, 2022.

[14] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models, 2023.

[15] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Qiuyi Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers, 2022.

[16] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second, 2023.

[17] Zifeng Wang, Chufan Gao, Cao Xiao, and Jimeng Sun. Meditab: Scaling medical tabular data predictors via data consolidation, enrichment, and refinement, 2023.

[18] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning, 2023.

[19] Tianfan Fu, Kexin Huang, Cao Xiao, Lucas M. Glass, and Jimeng Sun. Hint: Hierarchical interaction network for trial outcome prediction leveraging web data, 2022.

[20] New york city airbnb open data. `https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data`.

[21] New york city airbnb 2023, public data. `https://www.kaggle.com/datasets/godofoutcasts/new-york-city-airbnb-2023-public-data`.

[22] Open AI pricing page. `https://web.archive.org/web/20230928125454/https://openai.com/pricing`. Accessed: 2023-09-28.