LLM Embeddings Improve Test-time Adaptation to Tabular $Y|X$-Shifts

Yibo Zeng[1,*], **Jiashuo Liu**[2,*], **Henry Lam**[1], **Hongseok Namkoong**[1]

[1]Columbia University, [2]Tsinghua University
yibo.zeng@columbia.edu, liujiashuo77@gmail.com
khl2114@columbia.edu, namkoong@gsb.columbia.edu

Abstract

Distribution shifts between the source and target domain pose significant challenges for machine learning, and different types of distribution shifts require distinct interventions. After analyzing 7,650 distribution shift pairs across three real-world tabular datasets, we find that $Y|X$-shifts are more prevalent in tabular data, in contrast to image data, where X-shifts are more dominant. In this work, we conduct a comprehensive and systematic study on leveraging recent large language models to generate improved feature embeddings for backend neural network models. Specifically, we develop a large-scale testbed consisting of **7,650** distribution shift pairs across the `ACS Income`, `ACS Mobility`, and `ACS Public Coverage` datasets, following a standard training-validation-testing protocol. Through an extensive analysis of **20** models and learning strategies across over **261,000** model configurations, we find that while LLM embeddings are inherently powerful, they do not consistently outperform state-of-the-art tree-ensemble methods. Interestingly, even a small number of target samples can have a significant impact for tabular $Y|X$ shifts. Additionally, we explore the influence of target sample size, fine-tuning strategies, and methods of integrating supplementary information.

1 Introduction

Predictive performance degrades when the distribution of target domain shifts from that of source (training) [3, 32, 13, 9, 1]. Distribution shifts can be categorized into shifts in the marginal distribution of covariates (X-shifts) or changes in the relationship between the label and covariates ($Y|X$-shifts). In computer vision, X-shifts are prevalent since high-quality human labels are consistent across different images [25, 22, 29]; in contrast, $Y|X$-shifts are prevalent in tabular data due to missing variables and hidden confounders. There is a large body of work addressing X-shifts due to its dominance in vision and language [19, 35, 34], yet the work on $Y|X$-shifts remain limited [21].

The main challenge with addressing $Y|X$-shifts in tabular tasks is that the source data may provide little insight on the target distribution. Since it is impossible to generalize to a completely new and unknown domain [2, 27], we focus on leveraging few labeled target examples (on the order of 10 to 100) to address small $Y|X$-shifts that negatively impact model performance. Our goal is to build a feature representation $\phi(X)$ such that the difference between $\mathbb{E}_{\text{source}}[Y|\phi(X)]$ and $\mathbb{E}_{\text{target}}[Y|\phi(X)]$ are learnable even based on a few target data.

Using the wealth of world knowledge learned during pre-training, LLMs have the potential to build representations that mitigate the impact of confounders whose distribution changes across source and target. Specifically, we use a LLM encoder (`e5-Mistral-7B-Instruct`) to featurize tabular

[*]Equal contribution

Table Representation Learning Workshop at NeurIPS 2024.

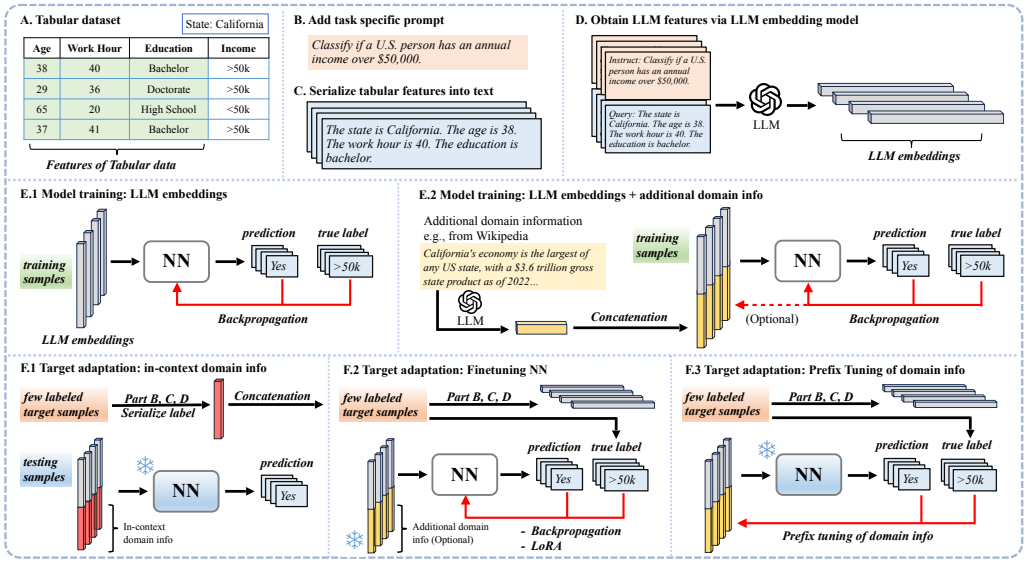


Figure 1: Overview of methods incorporating LLM embeddings.

data—which we referred to as LLM embeddings—and fit a shallow neural network (NN) on these embeddings for tabular prediction (Figure 1). In contrast to classical numerical encoding of tabular data, our approach automatically incorporates the semantics of each covariate using off-the-shelf LLMs, and can include additional contextual domain-level information that can help account for missing variables whose distribution shifts from source to target.

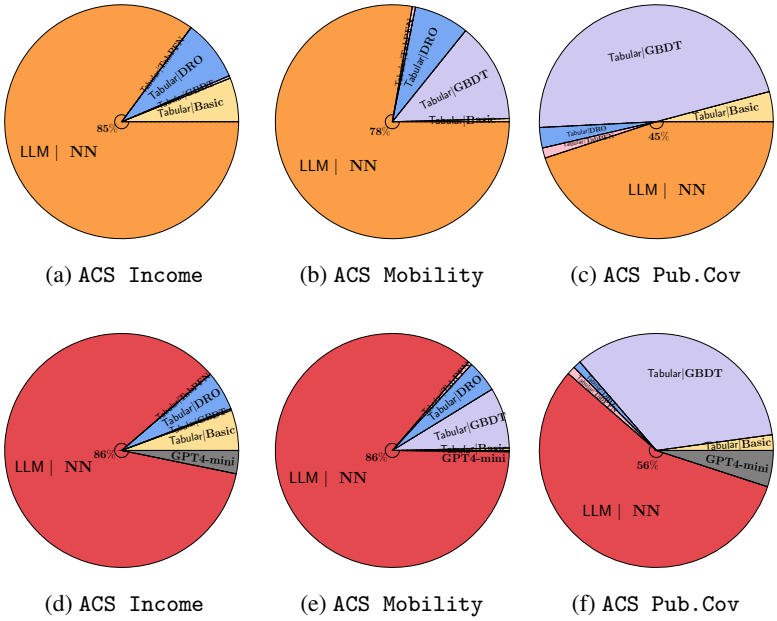
Throughout our investigation, we use the same LLM encoder to extract the LLM embeddings, and we only "finetune" the shallow NNs we use as the main prediction model across target domains. Investigating how different LLM encoders, e.g., LLMs specialized for tabular data [33], affect tabular $Y|X$ shifts is left as a future work. Comparison to other LLM-based tabular classification is discussed in the related work paragraph to come.

For rigorous empirical evaluation, we consider **7,650** natural spatial shifts (source→target) based on three real-world tabular datasets (`ACS Income, Mobility, Pub.Cov` [9]). Our testbed serves as a *large-scale* benchmark for $Y|X$-shifts on tabular data, offering a standardized protocol for training, validation, testing, and finetuning, as well as a consistent hyperparameter selection process. Compared to previous benchmarks on tabular distribution shifts [21], this paper not only explores a significantly greater variety of shift settings but also introduces a series of novel approaches to incorporate LLM embeddings as features. Such a comprehensive evaluation ensures the robustness and adaptability of our findings across diverse scenarios, setting a new standard for future work in this domain. We compare our proposed approach with typical methods on Tabular features, including basic models (LR, SVM, NN), gradient-boosting trees (GBDT; XGB, LGBM, GBM), and distributionally robust methods (DRO; KL-DRO, χ^2-DRO, Wasserstein DRO, CVaR-DRO, and Unified-DRO), as well as recent advanced pre-trained models, including TabPFN [15] and GPT4-mini (refer to Table 2 for summary). In total, we consider 22 algorithms and **261,000** model configurations.

Since it is unrealistic to expect any single method to uniformly dominate over large number of source→target settings, we complement traditional average-case metrics using the *fraction of times each method performs best*. For each method $\mathcal{M}$,

$$\text{FractionBest}(\mathcal{M};\Delta) := \frac{|\mathcal{S}(\Delta)\cap\mathcal{S}_{\mathcal{M}}|}{|\mathcal{S}(\Delta)|}, \tag{1.1}$$

where $\mathcal{S}(\Delta)$ contains all source→target settings where the performance between the best and second best model is larger than Δ (we set $\Delta = 1\%$ in this paper), and $\mathcal{S}_{\mathcal{M}}$ contains all source→target settings where model $\mathcal{M}$ performs the best. FractionBest calculates the proportion of source→target settings where (i) $\mathcal{M}$ outperforms all other methods and (ii) the improvement over the second-best model is meaningful (and significant).



(a) ACS Income (b) ACS Mobility (c) ACS Pub.Cov

(d) ACS Income (e) ACS Mobility (f) ACS Pub.Cov

Figure 2. The FractionBest Ratio in Equation (1.1) (with $\Delta = 1\%$). We compare our proposed methods—(a)-(c): LLM|NN and (d)-(f): LLM|NN (finetuning)—with methods on Tabular features.

First, we consider LLM embeddings without any adaptation to labeled target data[2]. Shallow networks based on LLM embeddings (LLM|NN) outperform all other methods on tabular features in **85%** settings in the ACS Income dataset, and in **78%** in the ACS Mobility dataset. However, for the ACS Pub.Cov dataset, the FractionBest drops to **45%**, which indicates that LLM embeddings do not always offer a perfect solution (see Figure 2 (a)-(c)). We conclude *LLM embeddings sometimes improve robustness*, but *do not consistently surpass* state-of-the-art tree-ensemble methods.

However, we find that *finetuning* the prediction model (shallow NN) *with few target samples* can *make a big difference* even when using identical LLM embeddings. When finetuning with just 32 target samples, the FractionBest ratio (Equation (1.1)) improves from 85% to **86%** on ACS Income, from 78% to **86%** on ACS Mobility, and from 45% to **56%** on ACS Pub.Cov (see Figure 2 (d)-(f)). We find this improvement *surprising*: although the shallow NN has numerous parameters, finetuning with only 32 target samples surprisingly improve target performance by a relatively large margin. More importantly, such improvement is observed under $Y|X$ shifts and holds across a wide range of distributional shift settings. This implies the potential of our novel and lightweight approach, and opens up the door for further investigation of using LLM embeddings in tabular classification tasks. Compared to in-context learning with advanced pre-trained models like TabPFN and GPT4-mini, *our methods are significantly more lightweight while exhibiting superior performance under $Y|X$ shifts*. This suggests a promising new research direction for integrating LLMs into tabular prediction. Theoretical insights are discussed in Section B.

Our method also implies multiple additional benefits (See Section 3.2). With the same amount of target samples, the finetuned NNs significantly outperform in-context learning with GPT4-mini, which can be viewed as the SOTA decoder model (see Figure 2 (d)-(f)). Moreover, *the performance gain brought by target samples is larger under stronger $Y|X$-shifts*, where the level of $Y|X$-shifts is measured by DISDE [7]. Finetuning with 32 target samples yields an average performance gain of 5.4 percentage points across the worst 500 settings on ACS Pub.Cov, compared to no finetuning. This is notably higher than the 1.2% average gain observed across all 2550 settings (**4.5 times**).

Beyond our primary findings, we also conduct ablation studies to better understand our approach in Section A. Given the large number of model parameters and limited labeled target samples, one might expect parameter-efficient methods like Low-Rank Adaptation (LoRA) [16] and Prefix Tuning [20]

[2]We do not conduct any target adaptation; however, we use 32 labeled target samples for validation (selection of hyperparameters, etc.).

to offer a clear advantage for target adaptation. However, we find *the specific finetuning approach has small impact* on target adaptability under tabular $Y|X$-shifts. In Figure 7, all target adaptation methods significantly outperform the non-finetuned version when using LLM embeddings. On the other hand, *incorporating the "right" domain information has an outsize impact* on adaptability to $Y|X$-shifts. For `ACS Pub.Cov`, adding additional domain information from Wikipedia shows minimal improvement alone, but significantly enhances performance under target adaptation.

Another practical question is how to allocate a fixed number of labeled target samples between target adaptation and validation (selection of finetuning method, hyperparameters, etc). In Figure 8 to come, we compare two allocation schemes of 64 labeled target samples: (i) using all 64 samples for validation (solid bar), and (ii) dividing the samples into 32 for validation and 32 for finetuning (shaded bar). For `ACS Mobility` and `ACS Pub.Cov`, *target adaptation provides significant gains over the validation-only approach*, highlighting the need for further investigation into sample allocation.

2 Methods

We introduce a series of methods utilizing LLM embeddings for tabular prediction, as well as different choices to incorporate additional domain information, different model architectures, and target adaptation techniques using a small amount of labeled target samples. This work is the first to comprehensively explore the impact of LLM embeddings on tabular $Y|X$-shifts.

2.1 LLM Embeddings for Tabular Prediction

We first introduce how we transform tabular data into LLM embeddings, where the key idea is to serialize each sample into a natural language format that the LLM can process. There is a substantial body of research on serialization, including using another LLM to rewrite tabular data into natural language [14], adding descriptions of the classification task, training and test examples [14, 31], etc. Among these methods, Hegselmann et al. [14] demonstrate that using a straightforward text template with a task description consistently achieves the best empirical performance.

Using an income prediction problem to illustrate, consider a simple task description such as "`Classify whether US working adults' yearly income is above $50000 in 2018.`" along with a simple serialization template that enumerates all features in the format "`The [feature name] is [value]`". Adopting this serialization approach, we employ the encoder model `e5-Mistral-7B-Instruct` to generate the LLM embedding. Formally, the encoder takes the serialization Serialize(X) of sample X as input and outputs its corresponding embedding $\Phi(X)$ as

$$X \xrightarrow{\text{serialization}} \text{Serialize}(X) \xrightarrow{\texttt{e5-Mistral-7B-Instruct}} \Phi(X).$$

Since `e5-mistral-7b-instruct` requires input data to be formatted in the following template:

```
Instruct:        description of the classification task \n
Query:           description of the data,
```

we provide task description in the "Instruct" part, and use the serilization template to format the tabular data in the "Query" part. An illustrative example is provided in Part A-D of Figure 1, with additional details available in Appendix C.1. Analyzing the impact of different LLM encoders, task descriptions, and serialization methods is left for future work.

2.2 Additional Domain Information

Another advantage of using LLM embeddings is their ability to incorporate additional domain information or prior knowledge, denoted by C. As demonstrated in Section 1, incorporating domain-specific information can help address $Y|X$-shifts and improve target generalization performance.

In this work, we propose a simple yet effective approach for integrating domain knowledge into tabular predictions. Rather than combining the domain information with serialized tabular data and generating a single LLM embedding, we generate separate LLM embeddings for the domain knowledge and the serialized tabular data, and then concatenate them together. The benefits of this approach are twofold: (a) although the domain information may contain significantly more words than the serialized tabular features, our concatenation method ensures a balanced 1:1 ratio between the

two, preventing a single embedding that disproportionately focuses on the longer domain information; (b) by separating the tabular features from the domain information, we can efficiently update the domain information without having to regenerate all the embeddings for the entire dataset.

We explore three sources of domain information: Wikipedia, GPT-4, and labeled target samples. Given that our experiments (see Table 1 and Section 3) focus primarily on socioeconomic factors, we collected "Economy" data for each U.S. state from Wikipedia as C. For GPT-4, we prompt it to provide background knowledge relevant to each prediction task in each state as C. For labeled target samples, we serialize 32 labeled samples from the concerned domain as the prior knowledge C. Further details can be found at Appendix C.2. After obtaining domain information C, we use `e5-mistral-7b-instruct` to generate an LLM embedding for C. As illustrated in Parts E.2 and F.1 of Figure 1, this embedding is then concatenated with the LLM embeddings of the tabular data, which serve as input to the backend neural network models (NN). This approach allows us to generate the LLM embedding for the dataset *just once*, and subsequently concatenate it with embeddings from different prompts as needed. In Section 3, we study whether and how this additional domain information can enhance generalization under $Y|X$-shifts.

In addition, recent works on prompt engineering have focused on incorporating additional domain information to enhance prediction tasks, often through detailed instructions [28, 30]. Our proposed framework introduces a novel approach to leveraging such information and remains fully compatible with these existing methods.

2.3 Model Training and Target Adaptation

Model architecture For the backend model, we use a vanilla neural network (NN) classifier on both tabular features and LLM embeddings for tabular data classification. The NN is a simple feedforward neural network with several hidden layers, dropout layer, and ReLU activation functions.

When adding additional domain information via an embedding layer, the same embedding is applied to all samples from the same domain. Since the output of `e5-mistral-7b-instruct` is a 4096-dimensional vector, we simply concatenate the LLM embeddings with the embeddings of the domain information. This concatenated vector is then passed through the hidden layers, dropout layer, and ReLU activation functions. For all NNs, the final linear layer an output dimension of 2, followed by a softmax layer for binary classification. During training, we use cross-entropy as the loss function, batch size as 128, and use the Adam optimizer. Detailed model architecture and hyper-parameters are provided in the Appendix C.3 and C.4, with a discussion on hyperparameter selection in Section 3.1.

Target Adaptation Even with the incorporation of LLM embeddings and domain information, our model may still experience $Y|X$-shifts. In practice, it is common to have a small set of samples from the target domain, which can be leveraged to better adapt the model to the target domain.

For each (source domain, target domain) pair, we begin by selecting the best training hyperparameter based on a validation criterion, which will be discussed in the Section 3.1. Using this model trained on the source domain, we explore four primary methods for target adaptation: in-context domain info, full-parameter fine-tuning, low-rank adaptation (LoRA), and prefix tuning for domain information.

For in-context domain info (F.1 of Figure 1), we keep the trained model frozen and only update the domain information, switching it from natural language description of labeled sample from the source domain during training to that of target domain during inference phase. For the other three methods, we conduct further training of the model. In full-parameter fine-tuning (F.2 of Figure 1), the entire neural network is fine-tuned using the target samples. For LoRA, we introduce a low-rank adaptation layer to each linear layer by incorporating two smaller matrices, A and B, both with a rank of 16. Specifically, matrix A has dimensions corresponding to the input size and the rank, while matrix B has dimensions corresponding to the rank and the output size. Matrix A is initialized with a mean of 0 and a standard deviation of 0.02, whereas matrix B is initialized with zeros. These matrices are then multiplied together and added to the original weight matrix. We then fine-tune only these LoRA parameters, while keeping the rest of the model unchanged. In prefix tuning (F.3 of Figure 1), the initial domain information embedding serves as a starting point for further refinement. During training, both the the NN and the domain information embedding of the source domain are trained. For target adaptation, we switch the domain information embedding from the source to the target domain. The NN is kept frozen, and only the domain information embedding of the target domain is updated using these samples from the target domain. We refer to this process as prefix tuning.

Table 1. Details of datasets used in this work. "# Source→Target Pair" denotes the number of distribution shift pair for each dataset, and we consider the natural *spatial* shift between US states.

#ID	Dataset	#Samples	#Features	Outcome	#Source Domains	#Target Domains	#Source→Target Pair
1	`ACS Income`	1.60M	9	Income≥50k	51 (US States)	50 (US States)	2550
2	`ACS Mobility`	621K	21	Residential Address	51 (US States)	50 (US States)	2550
3	`ACS Pub.Cov`	1.12M	18	Public Ins. Coverage	51 (US States)	50 (US States)	2550

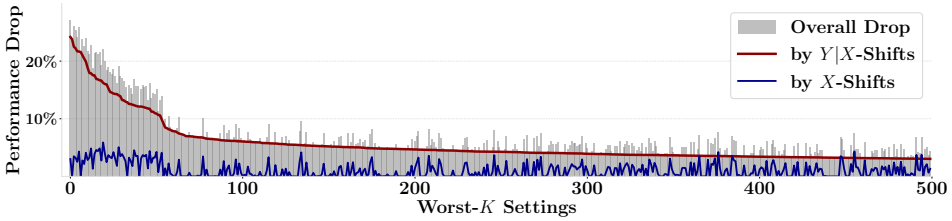


Figure 3. Shift pattern analysis. For the 2550 source→target distribution shift pairs in `ACS Income` dataset, we attribute the performance drop for each source→target pair into $Y|X$-shifts (red curve) and X-shifts (blue curve), and sort all pairs according to the drop introduced by $Y|X$-shifts. We draw the *worst-500* settings in each dataset, and the decomposition method used here is DISDE [7] with XGBoost as the reference model. Results on other datasets are in Figure 9.

As shown in Table 1, we use different hyperparameters for target adaptation. Detailed hyperparameters are provided in Appendix C.4, and the hyperparameter selection process is discussed in Section 3.1.

3 Numerical Experiments

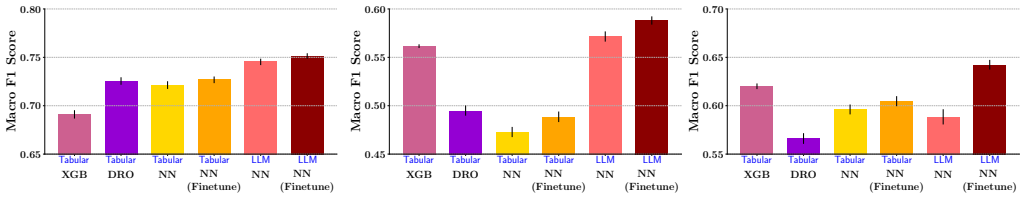
In this section, we conduct a thorough investigation of **7650** natural shift settings (source → target domain) in 3 tabular datasets over **261,000** model configurations and summarize the observations. Our findings highlight the potential of incorporating LLM embeddings to enhance the generalization ability in tabular data prediction tasks.

3.1 Testbed Setup

Dataset In this work, we use the ACS dataset [10] (`ACS Income`, `ACS Pub.Cov`, `ACS Mobility`) derived from the US-wide ACS PUMS data, where the goal is to predict various socioeconomic factors for individuals. The details of datasets are summarized in Table 1.

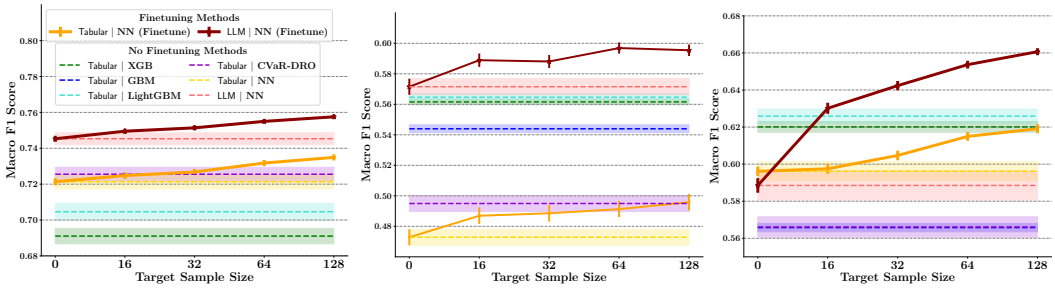
Algorithms As introduced in Section 2, we compare various methods that incorporate LLM embeddings into tabular data prediction, including different finetuning methods (no finetuning, finetuing on full parameters, and low rank adaptation (LoRA)) and different embeddings (w/ or w/o extra information). Besides, in order to fully compare the performances, we also include a wide range of learning strategies that perform on Tabular features, including basic models (LR, SVM, NN), tree ensembles (XGB, LGBM, GBM), robust methods (KL-DRO, χ^2-DRO, Wasserstein DRO, CVaR-DRO, and Unified-DRO), and in-context learning methods (TabPFN, GPT4-mini). All methods are summarized in Table 2.

Experiment Setup We conduct experiments with more than **261,000** model configurations on 2550 source→target shift pairs in `ACS Income`, `ACS Mobility`, and `ACS Pub.Cov` datasets respectively (**7650** settings in total). For each source→target shift pairs, we randomly sample $20,000$ labeled data from the source and target domain respectively, as the training and test dataset. We evaluate the model trained on the source domain, with or without target adaptation, and report the *Macro F1 score* on the testing dataset. Given the numerous training hyperparameters—learning rate, number of training epochs, hidden layer dimension, dropout ratio—we use a validation set of 32 randomly sampled labeled target domain samples to choose the optimal training hyperparameters, based on the highest F1 score in the validation dataset. Since our metric is Macro F1 score, the validation set is set as balanced between positive and negative classes. Note that the hyperparameter selection is *near*-oracle, as it leverages target samples, albeit in a limited quantity. When doing finetuning, we sample another 32 labeled target samples to finetune the model. And we use the same 32-sample validation dataset (for training hyperparameter selection) to select the target adaptation hyperparameters that yield the best Macro F1 score. Note that our testbed allows flexible sample sizes for training, validation, testing, and finetuning. Additionally, we perform an ablation study on different allocations of the



(a) `ACS Income` (b) `ACS Mobility` (c) `ACS Pub.Cov`

Figure 4. Average Macro F1 Score over the worst-500 settings. For each dataset, we sort the 2550 settings according to the magnitude of $Y|X$-shifts and select the **worst-500** settings. We calculate the average Macro F1 Score for each method. For all methods, we select the best hyper-parameters of the basic model according to 32 samples from the target domain. We use CVaR-DRO based on NN here to represent DRO methods. For finetuning methods, we use 32 target samples.



(a) `ACS Income` (b) `ACS Mobility` (c) `ACS Pub.Cov`

Figure 5. Average Macro F1 Score over Worst-500 settings with different #target samples used for finetuning. Dotted lines represent methods that do not require finetuning, while solid lines represent finetuning methods. All three figures share the same legend, and all results use 32 labeled target samples as validation dataset.

overall target samples in validation and finetuning (see Figure 8). See details on hyperparameters in the Appendix C.4.

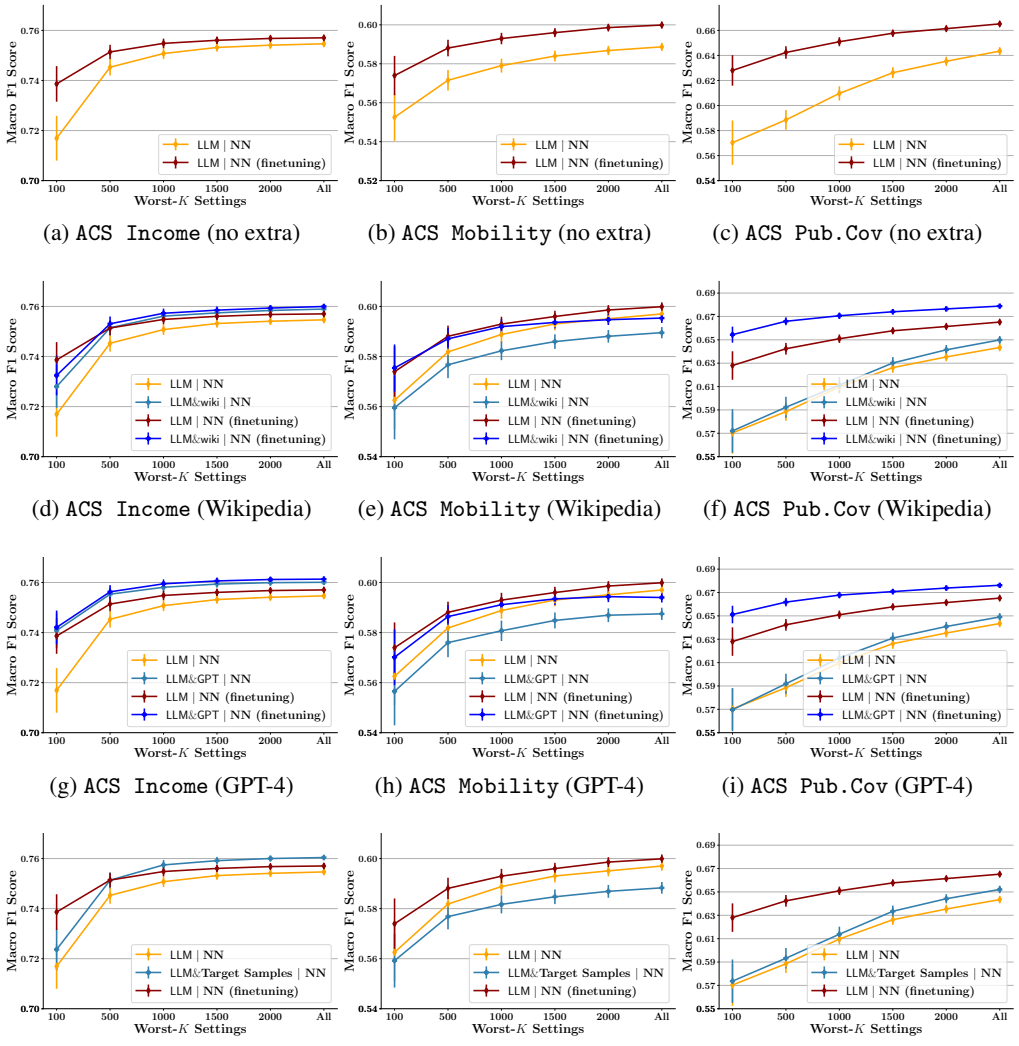
3.2 Primary Findings

We begin by presenting the key observations from our results. In addition to the metric (1.1) introduced in Section 1, we report performance metrics averaged over the source-target pairs.

LLM embeddings improve performance, but when applied alone do not consistently outperform tree-ensembles. To better assess the generalization ability when incorporating LLM embeddings, we select the worst 500 settings (out of 2,550 total settings per dataset) based on the severity of $Y|X$-shifts and report the average Macro F1 Score in Figure 4. Each bar represents the average result across these worst 500 settings, characterized by the most severe $Y|X$-shifts. Thus, even a 1pp improvement is significant, as it implies consistent gains of about 1pp across each of the worst 500 settings. Comparing "NN on LLM embeddings" to "NN on tabular features" (with the backbone model fixed as NN), we observe LLM embeddings significantly enhance generalization under distribution shifts on the `ACS Income` and `ACS Mobility` datasets, with average improvements of 2.4pp and 9.9pp. Notably, "NN on LLM embeddings" even outperforms XGBoost under strong distribution shifts on these datasets. This demonstrates the potential of LLM embeddings in tabular data prediction, where they can contribute to more generalizable models.

A different trend is observed on the `ACS Pub.Cov` dataset, where the inclusion of LLM embeddings results in a performance drop for NN models. This suggests that simply incorporating LLM embeddings does not always resolve distribution shift issues; their effectiveness may vary across datasets, particularly depending on whether the LLM embeddings provide additional relevant information for the specific prediction task.

A small number of target samples can make a big difference. While incorporating LLM embeddings doesn't always yield improvements, we find that even a small number of target samples can have a significant impact. As shown in Figure 4, finetuning the "NN on LLM embeddings" model with just 32 target samples significantly improves the average performance across the worst 500 settings for both `ACS Mobility` and `ACS Pub.Cov`. Similar trends are observed for other worst-K



(a) `ACS Income` (no extra) (b) `ACS Mobility` (no extra) (c) `ACS Pub.Cov` (no extra)

(d) `ACS Income` (Wikipedia) (e) `ACS Mobility` (Wikipedia) (f) `ACS Pub.Cov` (Wikipedia)

(g) `ACS Income` (GPT-4) (h) `ACS Mobility` (GPT-4) (i) `ACS Pub.Cov` (GPT-4)

(j) `ACS Income` (In-context) (k) `ACS Mobility` (In-context) (l) `ACS Pub.Cov` (In-context)

Figure 6. Average performance over the worst-K pairs. For each dataset, we sort the 2550 pairs according to the magnitude of $Y|X$-shifts and select the worst-K settings ($K \in \{100, 500, 1000, 1500, 2000, 2550\}$). For methods requiring finetuning, we use 32 target samples here. (a)-(c): NN based on LLM embeddings without extra information; (d)-(f): With domain information from Wikipedia; (g)-(i): With domain information generated by GPT-4; (j)-(l): With domain information from target-sample serialization (F.1 of Figure 1).

settings, as shown in Figure 6 (a)(b)(c) to come. Notably, for the `ACS Pub.Cov` dataset, where LLM embeddings alone provided no improvement, finetuning with only 32 target samples leads to a 5.4pp gain, even surpassing XGBoost by 2.2pp. This highlights the adaptability of LLM embeddings, making them a promising tool for harnessing their power across various downstream real-world tasks.

Furthermore, in Figure 5, we illustrate how the performance of finetuning methods varies with different numbers of target samples used for finetuning. Our conclusions hold true regardless of the number of target samples used for finetuning.

The performance gain brought by finetuning with target samples is larger under stronger $Y|X$-shifts. As shown in Figure 6 (a)-(c) (LLM | NN v.s. LLM | NN (finetuning)), for NN using LLM embeddings, finetuning with 32 target samples yields an average performance gain of 5.8pp across the worst-100 settings on `ACS Pub.Cov`, compared to no finetuning. This is notably higher than the 1.2pp average gain observed across all 2550 settings (**4.8 times**). Similarly, for `ACS Income`, it is about **9 times**.

References

[1] Evelin Amorim, Marcia Cançado, and Adriano Veloso. Automated essay scoring in the presence of biased ratings. In *Association for Computational Linguistics (ACL)*, pages 229–237, 2018.

[2] Martin Arjovsky, Leon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv:1907.02893 [stat.ML]*, 2019.

[3] Peter Bandi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, et al. From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge. *IEEE transactions on medical imaging*, 38(2):550–560, 2018.

[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79:151–175, 2010.

[5] Jose Blanchet, Yang Kang, Karthyek Murthy, and Fan Zhang. Data-driven optimal transport cost selection for distributionally robust optimization. In *2019 winter simulation conference (WSC)*, pages 3740–3751. IEEE, 2019.

[6] Jose Blanchet, Daniel Kuhn, Jiajin Li, and Bahar Taskesen. Unifying distributionally robust optimization via optimal transport theory. *arXiv preprint arXiv:2308.05414*, 2023.

[7] Tiffany Tianhui Cai, Hongseok Namkoong, and Steve Yadlowsky. Diagnosing model performance under distribution shift. *arXiv preprint arXiv:2303.02011*, 2023.

[8] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *ACM SIGKDD International Conference on Knowledge Discovery*, pages 785–794. ACM, 2016.

[9] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems 34*, 34, 2021.

[10] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in neural information processing systems*, 34:6478–6490, 2021.

[11] John C. Duchi and Hongseok Namkoong. Variance-based regularization with convex objectives. *Journal of Machine Learning Research*, 20(68):1–55, 2019.

[12] Josh Gardner, Zoran Popovic, and Ludwig Schmidt. Subgroup robustness grows on trees: An empirical baseline investigation. *Advances in Neural Information Processing Systems*, 35:9939–9954, 2022.

[13] David J Hand. Classifier technology and the illusion of progress. *Statistical Science*, 21(1):1–14, 2006.

[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. In *International Conference on Artificial Intelligence and Statistics*, pages 5549–5581. PMLR, 2023.

[15] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

[16] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

[17] Zhaolin Hu and L Jeff Hong. Kullback-leibler divergence constrained distributionally robust optimization. *Available at Optimization Online*, 1(2):9, 2013.

[18] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[19] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

[20] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

[21] Jiashuo Liu, Tianyu Wang, Peng Cui, and Hongseok Namkoong. On the need for a language describing distribution shifts: Illustrations on tabular datasets. In *Advances in Neural Information Processing Systems 36*, 2023.

[22] John Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[23] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011.

[25] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[26] R Tyrrell Rockafellar, Stanislav Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[27] Elan Rosenfeld, Pradeep Ravikumar, and Andrej Risteski. The risks of invariant risk minimization. In *Proceedings of the Ninth International Conference on Learning Representations*, 2021.

[28] Timo Schick and Hinrich Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.

[29] Vaishaal Shankar, Achal Dave, Rebecca Roelofs, Deva Ramanan, Benjamin Recht, and Ludwig Schmidt. Do image classifiers generalize across time? *arXiv:1906.02168 [cs.LG]*, 2019.

[30] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*, 2020.

[31] Dylan Slack and Sameer Singh. Tablet: Learning from instructions for tabular data. *arXiv preprint arXiv:2304.13188*, 2023.

[32] Andrew Wong, Erkin Otles, John P Donnelly, Andrew Krumm, Jeffrey McCullough, Olivia DeTroyer-Cooley, Justin Pestrue, Marie Phillips, Judy Konye, Carleen Penoza, et al. External validation of a widely implemented proprietary sepsis prediction model in hospitalized patients. *JAMA Internal Medicine*, 181(8):1065–1070, 2021.

[33] Jiahuan Yan, Bo Zheng, Hongxia Xu, Yiheng Zhu, Danny Chen, Jimeng Sun, Jian Wu, and Jintai Chen. Making pre-trained language models great on tabular prediction. *arXiv preprint arXiv:2403.01841*, 2024.

[34] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4396–4415, 2022.

[35] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

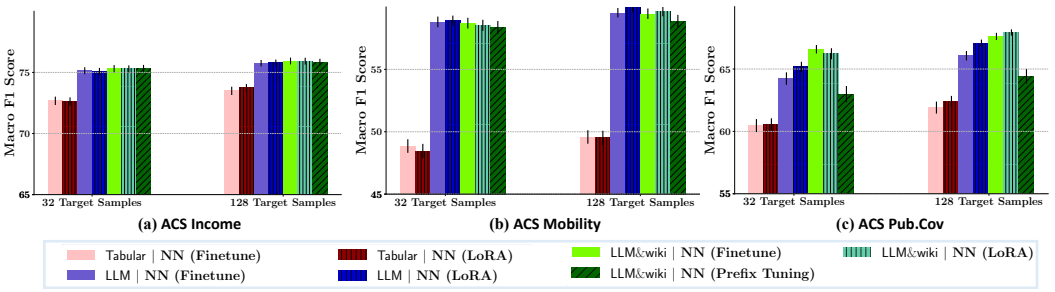


Figure 7. Comparison between full-parameter finetuning, LoRA, and prefix tuning. We show the average Macro F1 Score over the worst-500 settings.

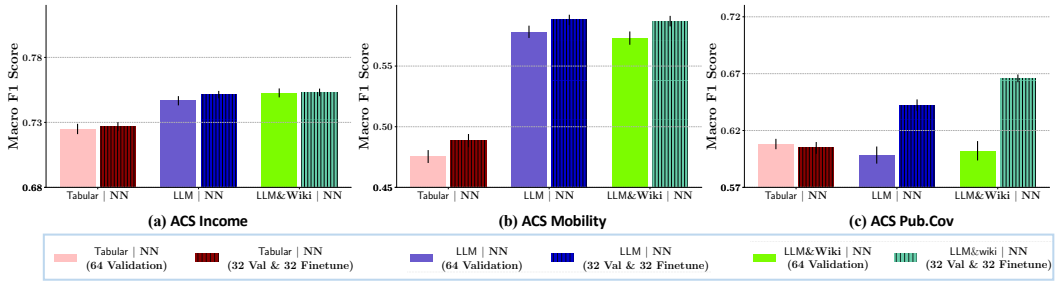


Figure 8. Comparison between different allocation of target samples into validation and finetuning. We report the average Macro F1 Score over the worst-500 settings.

A Auxiliary Findings

In addition to the primary findings, we have several other noteworthy observations.

"Right" domain information matters. In Figure 6 (d)-(l), we study how additional domain information — from Wikipedia, GPT-4, or 32 target samples (via in-context domain information, F.1 of Figure 1) impacts F1 scores in both non-adaptation and target adaptation scenario. We do NOT observe a consistent trend across all three datasets: additional domain information can either improve or reduce F1 scores. In the non-adaptation case, both LLM&wiki | NN and LLM&GPT | NN outperform LLM | NN in `ACS Income` (d)(g), but underperform in `ACS Mobility` (e)(h). When finetuning NNs, LLM&wiki | NN (finetuning) and LLM & GPT | NN (finetuning) beat LLM | NN (finetuning) in `ACS Pub.Cov` (f)(i), but underperform in `ACSMobility`. Moreover, LLM& Target Samples | NN shows comparable performance with LLM | NN (finetuning) in `ACSIncome` (j), but underperform LLM | NN (finetuning) in `ACS Mobility` and `ACS Pub.Cov` (k)(l), and even underperform LLM | NN that has no target adaptation in `ACS Mobility` (k). Our results indicates that "right" domain information can indeed improve tabular data classification under $Y|X$ shifts, yet identifying the best domain information requires non-trivial engineering efforts, which we leave for future work.

For simplicity, we only compare LLM embeddings (without extra domain information) to LLM&wiki in the sequel, as LLM&wiki and LLM&GPT exhibit similar trends.

Specific finetuning approaches are less crucial than expected. Given the large number of model parameters and limited labeled target samples, one might expect parameter-efficient methods like Low-Rank Adaptation (LoRA) and Prefix Tuning to offer a clear advantage. However, as shown in 7, all methods significantly outperform the non-finetuned version when using LLM embeddings, and the choice of finetuning method appears less significant in our setting.

An exception is prefix tuning on the `ACS Pub.Cov` task, where performance was several percentage points lower. While this requires further investigation, our key takeaway is that under $Y|X$ shifts, 1) finetuning models using LLM embeddings can greatly enhance classification performance; 2) popular finetuning methods yield comparable results.

Allocation of labeled target samples matters. In Figure 8, we compare two allocation schemes of 64 labeled target samples. For the solid bars, all 64 samples are utilized as the validation dataset for hyperparameter selection. For the shaded bars, we allocate 32 samples for validation and the remaining 32 samples for finetuning. Based on this, we initially explore and understand the impact of sample allocation. For `ACS Mobility` and `ACS Pub.Cov`, target adaptation using LLM embeddings (the shaded bar) significantly outperforms the validation approach (the solid bar). However, for `ACS Income`, the improvement from target adaptation is only marginal. This shows that although target adaptation is effective, its improvement is highly dependent on the specific distribution and $Y|X$ shift level. This raises further questions about how to optimally allocate resources. Although our findings endorse considering target adaptation, identifying the best allocation strategy is left as future work.

B Discussion based on theoretical insights

We take a brief examination of the theoretical insights that may lie behind our empirical findings. While standard generalization bounds are vacuous for neural networks, we nevertheless find that theoretical results from domain adaptation provide a useful starting point for understanding why finetuning LLM embeddings with small target samples can result in superior generalization performance under substantial $Y|X$ shifts, particularly in comparison to tabular features.

Let $\phi(X)$ denote a feature map and Y a binary label. We let P and Q be the source and target distributions. We consider a model class $\mathcal{H}$ of VC dimension d. For any model $h \in \mathcal{H}$, we use $\epsilon_P(h) := \mathbb{E}_P[\mathbb{I}(h(\phi(X)) \neq Y)]$ to denote the expected 0-1 loss on the source domain and $\hat{\epsilon}_P^m(h)$ its empirical counterpart based on m i.i.d. samples from P. We define $\epsilon_Q(h)$ and $\hat{\epsilon}_Q^m(h)$ on the target.

Although in practice we finetune on the target, we use a mixture problem as a rough approximation. Suppose that we have $(1-\beta)m$ i.i.d. samples from source domain P and (βm) i.i.d. samples from target domain Q. Let $\hat{h}_{\alpha,\beta}$ be the minimizer of the α-weighted empirical error

$$\hat{h}_{\alpha,\beta} := \operatorname*{argmin}_{h\in\mathcal{H}} \left\{ \alpha\hat{\epsilon}_Q^{\beta m}(h) + (1-\alpha)\hat{\epsilon}_P^{(1-\beta)m}(h) \right\}.$$

The following classical result from Ben-David et al. [4, Theorem 3] bounds the generalization error on the target.

Proposition 1. *For any $\delta \in (0,1)$, with probability at least $1-\delta$,*

$$\epsilon_Q(\hat{h}_{\alpha,\beta}) - \inf_{h\in\mathcal{H}} \epsilon_Q(h) \leq 4\sqrt{\frac{\alpha^2}{\beta} + \frac{(1-\alpha)^2}{1-\beta}} \sqrt{\frac{2d\log(2(m+1)) + 2\log\frac{8}{\delta}}{m}} + 2(1-\alpha)\underbrace{d_{\mathcal{H}\Delta\mathcal{H}}(P_X, Q_X)}_{X\text{-shifts}} + 2(1-\alpha)\underbrace{\inf_{h\in\mathcal{H}} \{\epsilon_P(h) + \epsilon_Q(h)\}}_{Y|X\text{-shifts}}, \tag{B.1}$$

where $d_{\mathcal{H}\Delta\mathcal{H}}(\cdot,\cdot)$ denotes the $\mathcal{H}\Delta\mathcal{H}$-distance between two (marginal) distributions.

Since we use limited (32) labeled target samples for finetuning, β is small. Also, we use a smaller learning rate for finetuning than for training on the source, implying that α is even smaller than β. Thus, we expect the constants $\sqrt{\frac{\alpha^2}{\beta} + \frac{(1-\alpha)^2}{1-\beta}}$ and $(1-\alpha)$ to both be close to 1. Although the VC-dimension d can be vacuously large for neural networks (a well-known defect in statistical learning theory), having a large enough sample size m can generally make the first term comparable to the next two terms. For the raw features $\phi_{\text{tabular}}(X) = X$, we have observed empirically that $Y|X$-shifts are salient, implying that the third term (related to $Y|X$-shifts) dominates the second term (related to X-shifts), and it can also dominate the first term when $Y|X$ shifts are particular significant. In comparison, we conjecture that LLM embeddings ϕ_{LLM} can reduce the gap between $\mathbb{E}_P[\mathbb{I}(Y \neq h(z) \mid \phi_{\text{LLM}}(X) = z]$ and $\mathbb{E}_Q[\mathbb{I}(Y \neq h(z)) \mid \phi_{\text{LLM}}(X) = z]$ by *incorporating prior knowledge* encoded during LLM pre-training. Since $\epsilon_P(h) = \int \mathbb{E}_P[\mathbb{I}(Y \neq h(z)) \mid \phi(X) = z] \, \mathrm{d}P_{\phi(X)}(z)$ (and similarly for Q), we thus expect the third term to be much smaller than using raw features $\phi_{\text{tabular}}(X)$. This suggests that when using LLM embeddings ϕ_{LLM}, the generalization bound can be smaller than that of raw features $\phi_{\text{tabular}}(X)$, especially when $Y|X$ shift is more significant under raw features $\phi_{\text{tabular}}(X) = X$; recall Figure 6.

We hope our empirical findings spur future theoretical investigations into the foundations of LLM-based target adaptation.

C Model Training Details

In this section, we outline the serialization scheme, the generation of additional domain information, our model architecture, and the hyperparameters used for training and target adaptation.

C.1 Serialization Scheme

As discussed in Section 2.1, serializing a row of tabular data, such as X, requires two components: a task description and a description of the data.

Task description For the ACS Income, ACS Mobility, and ACS Public Coverage datasets, we consider the same binary classification task as described in Ding et al. [9]. We adopt concise task descriptions as suggested by [14], as follows:

- `ACS Income`: Classify whether US working adults' yearly income is above $50000 in 2018.
- `ACS Mobility`: Classify whether a young adult moved addresses in the last year.
- `ACS Public Coverage` : Classify whether a low-income individual, not eligible for Medicare, has coverage from public health insurance.

Description of the data For all three ACS datasets, we utilize features as shown in [9, Appendix B], along with the domain name (state) to characterize the data. Specifically, for data from the source state, we apply the source domain name, and for data from the target state, we use the target domain name.

As illustrated in Figure 1 and recommended by [14], we adopt a straightforward text template: "The `feature name` is `value`." However, for certain features with less common or more complex feature names, we opt for a template as "The person (appropriate verb) `value`" to more clearly convey the data description. For example, for the feature "Gave birth to child within the past 12 months" with the value "No", the serialization would be "The person did not give birth to a child within the past 12 months." The features that use this alternative template are as follows:

- `ACS Income`: class of worker;
- `ACS Mobility`: class of worker, disability, employment status of parents, citizenship, military service, hearing difficulty, vision difficulty, cognitive difficulty, grandparent living with grandchildren, employment status;
- `ACS Public Coverage`: disability, employment status of parents, citizenship, mobility, military service, hearing difficulty, vision difficulty, cognitive difficulty, gave birth to child within the past 12 months, employment status.

For features without associated values, we omit them during serialization.

C.2 Additional domain information

As shown in Section 2.2, we study three sources of domain information: Wikipedia, GPT-4, and labeled target samples. As we use the `e5-mistral-7b-instruct` to generate an LLM embedding for the domain information, we need to specify both the "Instruct" and "Query" components.

For the "Instruct" part, we apply the same task description as outlined in Section C.1. For the "Query" part, we utilize various descriptions of the additional domain information:

- Wikipedia: For all three datasets, we use the "Economy" section of each state's Wikipedia page as the additional domain information.
- GPT4: For each state, we pose the following question to GPT4, and use its response as the additional domain:

- `ACS Income`: "We aim to develop a classifier to determine whether U.S. individuals earned over $50000 in 2018, using features such as age, sex, educational attainment, race, class of worker, marital status, occupation, and hours worked per week over the past 12 months. Given the unique economic and demographic profile of `state_name`, how might these factors influence income levels differently compared to other U.S. states? Please provide a 2000-word summary detailing these differences."
- `ACS Mobility`: "We aim to develop a classifier to determine whether a young adult moved addresses in the last year, using features such as age, sex, educational attainment, race, class of worker, marital status, occupation, total income, and hours worked per week over the past 12 months. Given the unique economic and demographic profile of `state_name`, how might these factors influence mobility levels differently compared to other U.S. states? Please provide a 2000-word summary detailing these differences."
- `ACS Public Coverage`: "We aim to develop a classifier to determine whether a low-income individual, not eligible for Medicare, has coverage from public health insurance, using features such as age, sex, educational attainment, race, disability, marital status, occupation, citizenship status, mobility status, military service, nativity, total income, and employment status. Given the unique economic and demographic profile of `state_name`, how might these factors influence public coverage levels differently compared to other U.S. states? Please provide a 2000-word summary detailing these differences."

- labeled target samples: We use the following template for 32 labeled target samples:

```
Here are some examples of the data:  \n
description of one target sample \n
Answer:  (Yes or No).  \n \n
description of another target sample \n
Answer:  (Yes or No).  \n \n
...
```

We use the same data descriptions as in Section C.1, with the exception that the state/domain name is omitted, as it is represented by the labeled target samples provided.

Shift Patterns The shift patterns of all settings are shown in Figure 9.

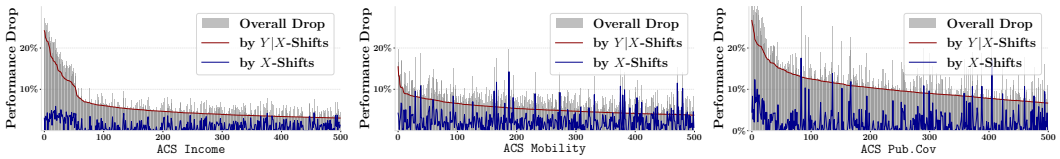


Figure 9. Shift pattern analysis. For the 2550 source→target distribution shift pairs in `ACS Income`, `ACS Pub.Cov`, `ACS Mobility` datasets respectively, we attribute the performance drop for each source→target pair into $Y|X$-shifts (red curve) and X-shifts (blue curve), and sort all pairs according to the drop introduced by $Y|X$-shifts. We draw the *worst-500* settings in each dataset, and the decomposition method used here is DISDE [7] with XGBoost as the reference model.

C.3 Model Architecture

We detail the baselines used in our paper.

Fully-connected Neural Networks (NN) Given the varying input dimensions for Tabular features, LLM embeddings, and LLM embeddings with additional domain information, we employ three neural networks with similar architectures. We then train these networks and conduct target adaptation using Empirical Risk Minimization (ERM).

For all three datasets using Tabular features, we use a hidden layer with output dimension being `hidden layer dim` as a hyperparameter, a dropuput layer with `dropout ratio` being a hyperparameter, ReLu activation. We then have another hidden layer with input and output dimension both being `hidden layer dim`, and then softmax layer with dimension 2 as the output.

For datasets using Tabular features, the network includes a hidden layer where the output dimension is set by the hyperparameter `hidden layer dim`. It is followed by a dropout layer (`dropout ratio` as a hyperparameter), ReLU activation, another hidden layer the input and output dimensions are both equal to `hidden layer dim`. The network concludes with a softmax layer with an output dimension of 2.

For datasets using only LLM embeddings, the input dimension is 4096, which is the output dimension of `e5-mistral-7b-instruct`. The network consists of three hidden layers with fixed dimensions, where the input and output dimensions are (4096, 1024), (1024, 256), and (256, 128), respectively. Each layer uses ReLU activation. Next, there's a hidden layer with an input dimension of 128 and an output dimension set by the `hidden layer dim` hyperparameter. This is followed by a dropout layer (with `dropout ratio` as a hyperparameter), ReLU activation, and another hidden layer where both the input and output dimensions are `hidden layer dim`. A ReLU activation follows this final hidden layer, and the network concludes with a softmax layer that outputs a dimension of 2.

For datasets using LLM embeddings concatenated with additional domain information, the input dimension is 8192. We slightly update the first three hidden layers, with input and out dimensions being (8192, 2048), (2048, 512), and (512, 128), respectively. All other neural network structure and hyperparameters are the same as NNs with LLM embeddings only.

When applying low-rank adaptation (LoRA) for target adaptation, we introduce a low-rank adaptation layer to each linear layer by incorporating two smaller matrices, A and B, both with a rank of 16. Specifically, matrix A has dimensions corresponding to the input size and the rank, while matrix B has dimensions corresponding to the rank and the output size. Matrix A is initialized with a mean of 0 and a standard deviation of 0.02, whereas matrix B is initialized with zeros. These matrices are then multiplied together and added to the original weight matrix.

Tree Ensemble Models Gardner et al. [12] show that several tree-based methods are competitive on tabular datasets. And gradient-boosted trees (e.g., XGB [8], LGBM [18], GBM [23]) are widely considered as the state-of-the-art methods on tabular data. Therefore, we compare XGB, LGBM, and GBM in this work. For GBM, we use the standard implementations in `scikit-learn` [24]. For XGB and LGBM, we use the standard implementations in the `xgboost` package[3] and the `lightgbm` package[4]. All these methods are trained on CPUs.

DRO Methods Distributionally robust optimization (DRO) methods optimize the worst-case loss over an ambiguity set $\mathcal{P}$:

$$\min_{f\in\mathcal{F}} \sup_{Q\in\mathcal{P}} \mathbb{E}_Q[\ell(f(X), Y)]. \tag{C.1}$$

The ambiguity set is typically chosen as a "ball" around the training distribution P_{tr}

$$\mathcal{P}(d, \epsilon) = \{Q : d(Q, P_{\text{tr}}) \leq \epsilon\}, \tag{C.2}$$

where $d(\cdot, \cdot)$ is a distance metric between probability measures and ϵ is the radius of set. When d is set as the generalized f-divergence (including CVaR) as:

$$d(P, Q) = E_Q\left[f(\frac{dP}{dQ})\right], \tag{C.3}$$

for the KL-DRO method [17], we use $f(x) = x\log x - (x-1)$; for the χ^2-DRO method [11], we use $f(x) = (x-1)^2$; for the CVaR-DRO problem [26], we use $f(x) = 0$ if $x \in [\frac{1}{\alpha}, \alpha]$ and ∞ otherwise, and α controls the worst-case ratio.

For Wasserstein DRO [5], we choose $d(\cdot, \cdot)$ as the Wasserstein distance. Unified-DRO [6] combines Wasserstein distance and KL-divergence as $d(\cdot, \cdot)$, and we follow their initial Github codebases and hyperparameter selection when implementing these methods.

C.4 Hyperparameters for Training and Target Adaptation

For each algorithm, we maintain a grid of candidate hyperparameters as shown in Tables 3 and 4. and perform hyperparameter selection as described in Section 3.1. When the number of hyperparameter

[3] `https://pypi.org/project/xgboost/`

[4] `https://pypi.org/project/lightgbm/`

Table 2. Summary of methodologies: As baselines, we use basic models (LR, SVM, and NN), GBDTs (XGB, LGBM, and GBM), robust learning methods (KL-DRO, χ^2-DRO, etc.), and recent advanced LLM in-context learning models (TabPFN and GPT4-mini). For methods utilizing LLM embeddings, we consider different ways to incorporate domain information and different model architectures and finetuning techniques.

Name	Feature	Domain Info	Model	Adaptation	# of HPs	Part of Fig. 1
LR	Tabular	-	LR	No Finetuning	34	-
SVM	Tabular	-	SVM	No Finetuning	34	-
GBDT	Tabular	-	XGB, LGBM, GBM	No Finetuning	200	-
KL-DRO	Tabular	-	SVM	No Finetuning	138	-
χ^2-DRO	Tabular	-	SVM	No Finetuning	138	-
Wasserstein-DRO	Tabular	-	SVM	No Finetuning	17	-
Unified-DRO	Tabular	-	SVM	No Finetuning	150	-
CVaR-DRO	Tabular	-	NN	No Finetuning	200	-
Tabular | NN	Tabular	-	NN	No Finetuning	96	-
				Finetuning	12	-
				LoRA	15	-
LLM In-Context Learning	Tabular	-	TabPFN	No Finetuning	-	-
		State Name	GPT4-mini	No Finetuning	-	-
LLM | NN	LLM Embeddings	-	NN	No Finetuning	96	E.1
				Finetuning	12	F.2
				LoRA	15	F.2
LLM & Wiki/GPT4 | NN	LLM Embeddings	Wikipedia or GPT4	NN	No Finetuning	48	E.2
				Finetuning	12	F.2
				LoRA	15	F.2
				Prefix Tuning	18	F.3
LLM & In-Context Domain Info | NN	LLM Embeddings	Labeled Target Samples	NN	No Finetuning	96	F.1

configurations exceeds 200, we randomly select 200 configurations to reduce computational cost and maintain fairness in the comparison across all algorithms.

Table 3. Training hyperparameter grids used in all experiments. $\diamond$: for methods with the total grid size above 200, we randomly sample 200 configurations for fair comparisons.

Model	Feature	Domain Info	# of HPs	Hyperparameter	Value Range
SVM	Tabular	-	96	C Kernel Loss γ	$\{1e^{-2}, 1e^{-1}, 1, 1e^{1}, 1e^{2}, 1e^{3}\}$ {linear, RBF} Squared Hinge $\{0.1, 0.3, 0.5, 1, 1.5, 2,$ scale, auto$\}$
LR	Tabular	-	23	L_2 penalty	$\{1e^{-3}, 3e^{-3}, 5e^{-3}, 7e^{-3}, 1e^{-2}, 3e^{-2}, 5e^{-2}, \ldots, 1.3, 1.7, 5\}$ $1e^{1}, 5e^{1}, 1e^{2}, 5e^{2}, 1e^{3}, 5e^{3}, 1e^{4}\}$
Tabular | NN	Tabular	-	96	Learning Rate Hidden Layer Dim Dropout Ratio Train Epoch	$\{0.001, 0.003, 0.005, 0.01\}$ $\{16, 32, 64, 128\}$ $\{0, 0.1\}$ $\{50, 100, 200\}$
GBM	Tabular	-	1680$^\diamond$	Learning Rate Num. Estimators Max Depth Min. Child Samples	$\{1e^{-2}, 1e^{-1}, 5e^{-1}, 1\}$ $\{32, 64, 128, 256\}$ $\{2, 4, 8, 16\}$ $\{1, 2, 4, 8\}$
LGBM	Tabular	-	1680$^\diamond$	Learning Rate Num. Estimators L_2-reg. Min. Child Samples Column Subsample Ratio (tree)	$\{1e^{-2}, 1e^{-1}, 5e^{-1}, 1\}$ $\{64, 128, 256, 512\}$ $\{0, 1e^{-3}, 1e^{-2}, 1e^{-1}, 1\}$ $\{1, 2, 4, 8, 16, 32, 64\}$ $\{0.5, 0.8, 1.\}$
XGB	Tabular	-	1944$^\diamond$	Learning Rate Min. Split Loss Max. Depth Column Subsample Ratio (tree) Column Subsample Ratio (level) Max. Bins Growth Policy	$\{0.1, 0.3, 1.0, 2.0\}$ $\{0, 0.1, 0.5\}$ $\{4, 6, 8\}$ $\{0.7, 0.9, 1\}$ $\{0.7, 0.9, 1\}$ $\{128, 256, 512\}$ {Depthwise, Loss Guide}
KL-DRO	Tabular	-	117	Uncertainty Set Size ϵ	$\{1e^{-4}, \ldots, 0.01, \ldots, 0.99\}$
χ^2-DRO	Tabular	-	117	Uncertainty Set Size ϵ	$\{1e^{-4}, \ldots, 0.01, \ldots, 0.99\}$
Wasserstein-DRO	Tabular	-	138	Uncertainty Set Size ϵ	$\{1e^{-4}, \ldots, 0.01, \ldots, 0.99, \ldots, 3\}$
CVaR-DRO	Tabular	-	1620$^\diamond$	Worst-case Ratio α Underlying Model Class	$\{0.01, 0.1, 0.2, 0.3, 0.5, 1.0\}$ NN
Unified-DRO	Tabular	-	180	Distance Type Uncertainty Set Size ϵ θ_1	L_{inf} $\{1e^{-3}, \ldots, 9e^{-1}\}$ $\{1.001, 1.01, 1.1, 1.5, 2, 3, 5, 10, 50, 100\}$
LLM | NN	LLM	-	48	Learning Rate Hidden Layer Dim Dropout Ratio Train Epoch	$\{0.001, 0.01\}$ $\{32, 64, 128\}$ $\{0, 0.1\}$ $\{100, 200, 300, 500\}$
LLM & In-Context Domain Info/Wiki/GPT4 | NN	LLM	Labeled Target Samples or Wikipedia or GPT4	48	Learning Rate Hidden Layer Dim Dropout Ratio Train Epoch	$\{0.001, 0.01\}$ $\{32, 64, 128\}$ $\{0, 0.1\}$ $\{100, 200, 300, 500\}$

Table 4: Target adaptation hyperparameter grids used in all experiments.

Model	Target Adaptation Method	# of HPs	Hyperparameter	Value Range
Tabular | NN or LLM | NN or LLM & Wiki/GPT4 | NN	Finetuning	-	12	Learning Rate Target adaptation Epoch	$\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$ $\{25, 50, 100\}$
Tabular | NN or LLM | NN or LLM & Wiki/GPT4 | NN	LoRA	-	12	Learning Rate Target adaptation Epoch	$\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 0.01\}$ $\{25, 50, 100\}$
LLM & Wiki/GPT4 | NN	Prefix Tuning	-	18	Learning Rate Target adaptation Epoch	$\{10^{-5}, 10^{-4}, 10^{-3}, 0.01, 0.05, 0.1\}$ $\{25, 50, 100\}$