# LIFT: Language-Interfaced FineTuning for Non-Language Machine Learning Tasks

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Finetuning pretrained language models (LMs) has become a norm for learning various downstream tasks. While it is feasible to finetune LMs for language downstream tasks without making any architectural changes, most existing finetuning approaches for non-language tasks (*e.g.,* tabular data classification and regression) rely on task-specific designs for input, output layers, and loss functions. A natural question arises: can language model finetuning solve non-language downstream tasks without changing the model architecture or loss function? To answer this, we study the efficacy and limitations of **Language-Interfaced Fine-Tuning (LIFT)** for non-language tasks by conducting an extensive empirical study on a suite of classification and regression tasks. LIFT does not make *any* changes to the model architecture or loss function, and it solely relies on the *natural language interface*, enabling "no-code machine learning with language models". We find that LIFT performs comparably well to the best models in many cases, across a wide range of low-dimensional classification and regression tasks. We report experimental results on fundamental properties of LIFT, including its decision boundaries, sample efficiency, ability to extrapolate, and robustness. We also analyze a few unique properties specific to LIFT, *e.g.*, calibrated predictions and how to use them, and sample-efficient context-aware learning via appropriate prompting or two-stage finetuning. We provide discussions on limitations and open questions toward making LIFT more effective and efficient.

## 1 Introduction

Deep neural networks have been extremely successful across a multitude of domains, from computer vision [1, 2] and natural language processing [3, 4], to game playing [5, 6]. Most advances in deep learning have come with a variety of domain-specific designs for network architectures, such as convolutional filters [7, 8, 9] for vision tasks, or recurrent modules [10, 11] and attention mechanisms [12, 13] in the context of naturallanguage processing. A domain-and-modality agnostic model that can be adapted to solve tasks across different modalities and domains has become a desideratum [14], motivating great efforts in transfer learning [15] and multi-modal learning [16]. Recently, transformer-based language models (LMs) [13, 17, 18, 19] exhibited impressive versatility across different domains and modalities. They have been adapted far beyond language domains [20], such as sentiment analysis [21, 22], question answering [23], or commonsense reasoning [24], but also have been applied to non-language modalities [18]. For instance, GPT2 [17] pretrained on language data can be efficiently finetuned to perform image classification and numerical computation [18].

Adapting pretrained LMs to downstream language tasks can be achieved without modifying the models' architecture. Typically this adaptation is enabled via simple finetuning [25, 26, 27, 28] or in-context few-shot learning methods [29, 30]. However, not altering the architecture may pose a limitation for transferring to non-language tasks, such as tabular data classification and regression. Though achieving tremendous success, deep learning still faces difficulties on standard learning tasks, especially on tabular data, likely because of its heterogeneous nature with features being dense numerical and weaker in correlation compared to natural image-language data [31].

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

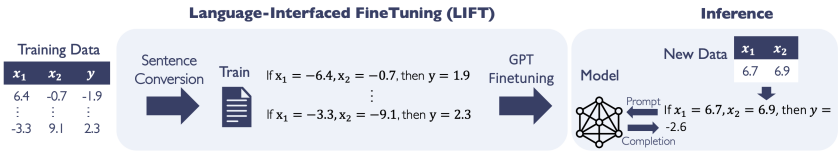



Figure 1: **A high-level illustration of Language-Interfaced FineTuning (LIFT).** The LIFT framework consists of two phases: dataset conversion and language model finetuning. To obtain a model for a given task, all you need with LIFT is to design proper sentence templates and convert the labeled data to a sentence dataset. No changes to architecture or loss functions are needed.

As their input and output formats are not necessarily in some language form, adapting LMs to these domains usually requires architectural changes. It is typical to re-design the input/output layers, and loss function to accommodate a different predictive task. For instance, to adapt GPT2 [23] to other modalities, the frozen pretrained transformer [18] adds an input and an output layer for learning the input and output transformations before finetuning. Such changes require a good understanding of the underlying principles of LMs and then some modifications to the implementation of such model. A natural question that arises is whether such changes are necessary. In other words,

| |
|---|
| Can LM finetuning solve non-language tasks<br>**without** changing the architecture or loss function? |

To answer to this question, we consider a simple finetuning procedure for pretrained LMs, which we refer to as **Language-Interfaced FineTuning (LIFT)**. The LIFT framework can be used to learn a predictor for any classification or regression task. LIFT runs in two phases: (1) it converts labeled samples into sentences and (2) it finetunes pretrained LMs on the sentence dataset without altering the decoder architecture or loss function.

Fig. 1 illustrates how we finetune GPT with LIFT to solve a regression task. Here, input data has two features $\mathbf{x}_1, \mathbf{x}_2$ and the goal is to predict $\mathbf{y}$. LIFT first converts each labeled sample into a sentence. In this example, we use a specific template "`If` $\mathbf{x}_1 = \text{value}_1, \mathbf{x}_2 = \text{value}_2$, `then` $\mathbf{y} = \text{value}_3$" for input sentences. We then finetune LMs without changing either architecture or loss function. Once the model is finetuned, we perform inference as follows. LIFT first convert test samples into sentences using the same template, while leaving the $\mathbf{y}$ value empty. It then feeds the converted sentences as prompts to the finetuned model. The output tokens are parsed to provide our prediction.

We empirically show that LIFT can provide high accuracy solutions for a variety of non-language tasks. Fig. 2 shows examples of real functions learned by GPT-J models finetuned using LIFT. As LIFT does not require any architectural changes, this shows the potential that finetuning LMs may not always require adapting their architecture/loss function, even when the target task is non-language. Thus, LIFT can be almost perceived as a "no-code machine learning" framework as the data-to-sentence conversion is extremely straightforward even without extensive programming skill.

We investigate the efficacy and limitations of LIFT on non-language tasks by conducting an extensive empirical study on a suite of classification and regression tasks. *First*, we observe that LIFT performs well across a wide range of low-dimensional classification and regression tasks. Although its accuracy is not the best overall, it nearly matches the performance of the best baseline models in most cases. To further understand LIFT, we provide experiments testing fundamental learning properties, e.g., its inductive bias, sample efficiency, ability to extrapolate, outlier, worst- and average-case noise robustness. *Thirdly*, we study a few unique properties specific to LIFT, *i.e.,* calibrated predictions and how we can use them, and sample-efficient context-aware learning via appropriate prompting. *Lastly*, to improve upon basic finetuning, we employ the two-stage finetuning technique with synthetic pretext tasks, and data augmentation with mixup. We find that two-stage finetuning improves the performance of LIFT in the regime of low training samples. We discuss the current limitations of LIFT and future investigations towards making LIFT more effective and efficient.

**Scope of study** The goal of this work is to understand: (i) what LIFT can and cannot, (ii) properties of finetuned models obtained via LIFT, and (iii) several techniques to improve LIFT. We emphasize that LIFT is not meant to achieve state-of-the-art performance across all tasks, but to explore the extent to which we can achieve good performance with minimal additional code and effort.

## 2 Related Work

**Pretraining and finetuning large-scale language models.** Modern large-scale language models have allowed for significant advances in field of natural language processing [4]. Starting from

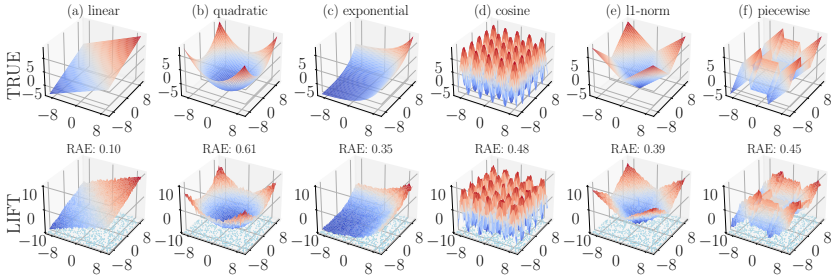



Figure 2: **Approximating various functions with LIFT using GPT-J.** We visualize the target functions (first row) and the predictor functions learned by LIFT on GPT-J (second row). Blue dots show the 1000 training samples. One can observe that LIFT well approximates the target functions.

BERT [32], pretraining LMs and finetuning on various downstream tasks has become the standard practice, achieving astonishing performance across multiple tasks [33, 34, 21, 23, 35, 36]. Finetuning LMs also enables high accuracy solutions for downstream tasks across different domains [37]. So far, finetuning LMs, without architecture changes, is limited to NLP tasks. In this work, we explore whether finetuning LMs can be beneficial for non-language tasks across different modalities. Finetuning methods for LMs have been widely studied, ranging from simple architecture modifications: adding linear layers [38, 39], and freezing parts of the network [18, 40, 41]; to more advanced methods such as adapter networks [26, 27, 28], low-rank adaptation of network parameters [25], and intermediate finetuning [42, 43, 44]. These methods can be adopted for non-language tasks with modifications over the input/output layers and the loss functions. However, these modifications might steer pretrained parameters into unwanted parts of the parameter space, leading to undesired behaviors like catastrophic forgetting. The in-context, few-shot learning paradigm [19, 45, 29, 30, 46] suggests modifying only LM inputs by adding a few examples of the downstream task. These models work well for NLP downstream tasks, but it is unclear how to apply them to downstream tasks of other modalities. Our work studies the feasibility of directly finetuning LMs for non-language tasks, further pushing the application boundaries of large-scale language models.

**Adaptation of LMs to non-language tasks.** Transformer-based architectures have been successfully applied to non-language domains and modalities, such as computer vision [47, 2], protein sequences [48], reinforcement learning [49], and tabular data [31, 50, 51]. Multiple works study the transferring of these models for downstream tasks in the same modality, such as language [52], vision [2] or different modalities, such as CLIP for language-image association [53] or between English and programming languages [54]. However, directly finetuning LMs for other modalities is just recently visited [55, 18]. Recent work [18] empirically shows that GPT2 [23] pretrained on language tasks and data can be efficiently finetuned for different modalities and domains. By freezing most pretrained parameters except the layer normalization layers and adding input and output layers, they can achieve competitive performance on various datasets. We note that these works require some changes in the architecture and losses to adapt to different data representations [18], while LIFT requires no such changes. To the best of our knowledge, we are one of the first to study LLM adaptation without architecture changes on **tabular data**,. Furthermore, transformer-based language models are observed not to work well on complex numeric tasks [56, 57, 58] and are sensitive to different number formats [56]. These transformer-based models for tabular data [50, 51] have different architecture designs from the LMs. Nevertheless, standard learning algorithms (Random Forest [59] or XGBoost [60]) are found to outperform deep learning models on several tabular datasets [31].

## 3 Methodology and Experimental Setup

We describe the methodology used in our empirical study. Our code is publicly available here. [1]

**LIFT.** To finetune a pretrained LM with LIFT on a supervised dataset, we apply two steps: (1) convert each sample into a sentence with a fixed template, (2) perform standard finetuning on LMs with sentence datasets. We use the default cross-entropy loss for token prediction in LMs. Our template is "`When we have` $\mathbf{x}_1 = v_1, \ldots, \mathbf{x}_p = v_p$, `what should be the` $\mathbf{y}$`?### y =`$v_{\mathbf{y}}$`@@@`", where $v_1, \ldots, v_p$ are feature values and $v_{\mathbf{y}}$ is the response value. For the testing sample, we leave the value $v_{\mathbf{y}}$ blank. The predicted output is parsed from the output tokens generated from finetuned LMs.

[1] `https://anonymous.4open.science/r/LanguageInterfacedFineTuning`

Table 1: **Summary of main findings on LIFT.**

| Property | Findings |
|---|---|
| Overall performance | For classification, LIFT can achieve comparable performances to strong baselines (Table 2) on various datasets. For regression, LIFT can approximate different function types in the low-dimensional cases (Fig. 2), but does not work well in the high-dimensional cases. |
| Robustness | In general, LIFT does not show clear robustness over label corruption, adversarial examples, etc. For regression, LIFT is robust to outliers (Fig. 23). |
| Context-aware learning | Performance of LIFT can be further improved by providing the context (feature names) of tasks. The improvement is more significant in the datasets in which common knowledge helps (Tables 21, 5). |
| Two-stage finetuning | When applying synthetic pretext tasks with LIFT, the number of required samples for achieving high performance is reduced (Fig. 6). |

**Pretrained LMs.** We use GPT-J [61] (6B parameters) and GPT-3 [19] (2.7B to 175B parameters). For GPT-J, we apply Lora [25] for efficient finetuning that essentially approximates the weight update of each layer by a low-rank matrix. For experiments on GPT-J, we used `p3.8xlarge` and `p3.2xlarge` instances from AWS and RTX3090 GPUs in the local server. Since GPT-3 is not fully publicly available, we use the API provided by OpenAI to perform black-box GPT-3 finetuning.

**Evaluation.** We evaluate the performance of LIFT on various non-language classification/regression tasks. Note that GPT may produce *invalid outputs*, such as non-numerical outputs. For classification, we consider invalid outputs as misclassifications; for regression, we increase the generation randomness and regenerate five times to obtain valid outputs. If valid outputs are not available, we measure the regression error by replacing the invalid outputs with the mean of the observed outcome of the training dataset. In terms of *evaluation metrics*, we measure accuracy on classification tasks, and RMSE, RAE errors on regression tasks, where $\text{RAE} = \frac{\sum_{i=1}^{n} |\hat{\mathrm{y}}_i - \mathrm{y}_i|}{\sum_{i=1}^{n} |\bar{\mathrm{y}} - \mathrm{y}_i|}$ and $\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{\mathrm{y}}_i - \mathrm{y}_i)^2}{n}}$ on each dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathrm{y}_i)\}_{i=1}^{n}$ with $n$ samples, features $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^p$, and outcome y.

**Datasets.** We design and select a wide range of datasets to better understand the behavior of LIFT. For *classification*, we use three types of non-language data: low-dimensional synthetic datasets, real tabular datasets in OpenML [62] and vision datasets (MNIST [63], Fashion-MNIST [64] and their permuted variant [65]). For *regression*, we use both synthetic and real datasets. For synthetic datasets, we defined samples $(\mathbf{x}_i, \mathrm{y}_i)$ of input-output pair as $\mathrm{y}_i \sim f(\mathbf{x}_i) + \mathcal{N}(0, \sigma^2)$, where $\sigma^2 \geq 0$ is the noise level. Unless otherwise stated, we sample the feature $\mathbf{x}_i$ uniformly from a hypercube $[L, U]^p$, where $L$ and $U$ are minimum/maximum feature values, and $p$ is the number of features. Following the suggestion by [66], we consider various functions $f$ for regression tasks: (i) linear function, (ii) quadratic function, (iii) exponential function, (iv) cosine function, (v) $\ell 1$-norm function, and (vi) piece-wise function. We also use three real datasets: Medical Insurance [67], Combined Cycle Power Plant (CCPP) [68], Servo [69] and student [70]. More details are included in Appendix.

**Baselines.** We consider standard machine learning algorithms [71, 72]. The classification baselines are logistic regression (*LogReg*), decision tree (*DT*), k-nearest neighbor (*KNN*), support vector machine with Gaussian kernel (*SVM*), random forest (*RF*), and XGBoost (*XG*). We also compared with an optimal deterministic classifier (*ODC*) that outputs the most dominant class. For regression, we use polynomial regression (*PR*), kernel ridge regression (*KR*) with radial basis function kernel, $k$-nearest neighbors (*KNN*), a three-layer ReLU neural network (*MLP*) with 50 hidden neurons at each layer, Gradient Boosting Trees (*GBT*), random forest (*RF*), and Gaussian process (*GP*).

# 4 Basic Findings of LIFT

We study LIFT on standard non-language classification and regression tasks. Table 1 summarizes our main findings on LIFT. We provide further observations of different LMs on LIFT in Appendix C.2.

## 4.1 How well does LIFT perform on standard ML tasks?

**Classification.** Table 2 compares classification accuracies between algorithms on a wide range of tasks. We observe that LIFT achieves comparable performance to most baselines. In most cases,

Table 2: **Accuracies (↑) on various classification datasets.** Full comparison with all baselines are in Table 20. Overall, LIFT/GPTs perform comparably well to baselines across tasks and can adapt well to non-linear datasets (circles, two circles), beyond the capacity of logistic regression. On the OpenML data, LIFT/GPTs achieve competitive performances with the best methods, such as XGBoost or RBF-SVM. The performance of LIFT degrades as the number of classes is large. Also, LIFT/GPTs achieve competitive accuracies on all vision datasets. *p, c: number of features, classes.*

| Dataset (ID) | $p$ / $c$ | ODC | LogReg | DT | RBF-SVM | XG | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|
| circles (3) | 2 / 2 | 50.00 | 48.58±1.94 | 77.42±0.24 | **83.08±0.59** | 81.42±0.31 | 79.95±1.53 | 81.17±0.42 |
| two circles (6) | 2 / 2 | 50.00 | 49.83±4.18 | 75.50±0.20 | 80.00±0.54 | 79.25±0.35 | 75.92±1.65 | **81.42±0.82** |
| blobs (2) | 2 / 4 | 25.00 | **96.75±0.00** | 96.08±0.82 | **96.75±0.00** | 96.17±0.12 | 96.17±0.59 | 96.67±0.24 |
| moons (4) | 2 / 4 | 50.00 | 88.58±0.12 | 99.25±0.41 | **100.00±0.00** | 99.83±0.12 | 99.58±0.42 | **100.00±0.00** |
| 9Clusters (1) | 2 / 9 | 11.25 | 100.00±0.00 | 100.00±0.00 | **100.00±0.00** | **100.00±0.00** | 99.75±0.00 | **100.00±0.00** |
| Customers (1511) | 8 / 2 | 68.18 | **87.12±0.54** | 85.98±0.53 | 86.36±0.00 | 85.23±0.00 | 85.23±1.61 | 84.85±1.42 |
| Pollution (882) | 15 / 2 | 50.00 | 58.33±11.79 | **77.78±3.93** | 58.33±6.81 | 63.89±7.86 | 63.89±3.93 | 63.89±7.86 |
| Spambase (44) | 57 / 2 | 60.59 | 93.27±0.00 | 90.7±0.14 | 93.70±0.00 | **95.87±0.00** | 94.03±0.54 | 94.90±0.36 |
| Hill-Valley (1479) | 100 / 2 | 49.79 | 77.78±0.00 | 56.38±0.89 | 68.72±0.00 | 59.26±0.00 | **100.00±0.20** | **99.73±0.19** |
| IRIS (61) | 4 / 3 | 33.33 | 96.67±0.00 | 97.77±3.85 | **100.00±0.00** | **100.00±0.00** | 96.67±0.00 | 97.0±0.00 |
| TAE (48) | 5 / 3 | 35.48 | 45.16±4.56 | 65.59±5.49 | 53.76±6.63 | **66.67±8.05** | 61.29±6.97 | 65.59±6.63 |
| CMC (23) | 9 / 3 | 42.71 | 49.49±0.83 | 56.72±0.32 | 56.50±0.97 | 52.43±0.42 | 49.83±0.28 | **57.74±0.89** |
| Wine (187) | 13 / 3 | 38.89 | **100.00±0.00** | 93.52±2.62 | **100.00±0.00** | 97.22±0.00 | 93.52±1.31 | 92.59±1.31 |
| Vehicle (54) | 18 / 4 | 25.88 | 80.39±1.00 | 63.92±2.37 | **81.18±0.48** | 73.14±0.28 | 64.31±2.37 | 70.20±2.73 |
| LED (40496) | 7 / 10 | 11.00 | 68.67±0.94 | 66.33±2.87 | 68.00±0.82 | 66.00±0.82 | 65.33±0.47 | **69.33±2.05** |
| OPT (28) | 64 / 10 | 10.14 | 96.53±0.22 | 89.8±1.09 | 97.95±0.00 | 97.48±0.17 | 98.22±0.11 | **98.99±0.30** |
| Mfeat (12) | 216 / 10 | 10.00 | 97.67±0.12 | 87.67±1.05 | **98.83±0.24** | 96.75±0.00 | 94.17±1.75 | 93.08±0.24 |
| Margin (1491) | 64 / 100 | 0.94 | 81.35±0.15 | 43.86±1.21 | **81.98±0.30** | 70.21±0.29 | 50.23±1.33 | 59.37±0.92 |
| Texture (1493) | 64 / 100 | 0.94 | 81.67±0.97 | 46.88±1.93 | **83.44±0.89** | 70.73±1.41 | 50.32±2.18 | 67.50±1.42 |
| MNIST | 784 / 10 | 11.35 | 91.95±0.69 | 87.42±0.64 | 97.70±0.97 | 97.69±0.04 | 97.01±1.15 | **98.15±0.67** |
| Permuted MNIST | | 11.35 | 92.58±0.04 | 87.87±0.69 | **98.06±0.31** | 97.62±0.09 | 95.80± 0.07 | 96.25±0.35 |
| Fashion MNIST | | 10.00 | 85.59±0.09 | 80.52±0.40 | **90.59±0.02** | 90.19±0.04 | 85.10 ± 0.19 | 90.18 ±0.12 |
| Permuted F-MNIST | | 10.00 | 84.95±0.84 | 79.91±0.93 | 88.04±1.69 | **89.93±0.14** | 82.25±0.27 | 88.92±0.71 |

LIFT/GPT ranks highly in the top three best-performing methods. We find that LIFT can learn non-linear relationships between features and the responses: LIFT achieves $80.17\%$ accuracy on the circle dataset while logistic regression failed to perform better than the ODC ($50\%$). As the difficulty of tasks varies, which can be estimated by the average performance of baselines, LIFT also suffers from the performance degradation. LIFT can perform relatively well even when the number of features is as large as hundreds, though the limited number of tokens as inputs to LMs restricts the number of features LIFT can input. However, when the number of classes is large (say 100s), both LIFT/GPT-J and LIFT/GPT-3 have lower accuracies than many baselines, even they manage to be better than ODC. For instance, the accuracy gap between LIFT/GPT-J and the best algorithm, RBF-SVM, on 100-class dataset (OpenML ID=1491) is nearly 30%. While most baselines require feature scaling and normalization for good performance, LIFT can directly use raw values.

**Regression.** See Sec. C.1.4 for the *function approximation* comparison. For the low-dimensional cases, LIFT achieves comparable performance to baselines but fails to beat the strongest baselines such as GP. This is as expected as GPT models measure the error by comparing tokens instead of measuring how close the prediction values are to true values. We conjecture our performance can be further improved by level encoding, *i.e.*, representing the numerical values as binary values. However, this is beyond the scope of our current work. We also investigate the *interpolation and extrapolation* of LIFT, the results of which are in Sec. C.1.4. All methods fail to extrapolate and interpolate well for all functions. It turns out that LIFT is not having good interpolation performance except in the linear regression case. An interesting observation we have made is that LIFT tends to output seen values (from training data) for extrapolation.

### 4.2 Can we understand the inductive biases of language models via LIFT?

We investigate the inductive biases of models via their decision boundaries. In doing so, we train a simple neural network to construct a set of decision boundaries with different levels of complexity, shown in the first column of Fig. 3, and observe how LIFT behaves for different datasets. Fig. 3 visualizes decision boundaries learned by different algorithms. Both GPT-J and GPT-3 models finetuned with LIFT are able to adapt to different boundaries. Interestingly, their boundary shapes are axis-parallel, similar to those of tree-based classifiers. We conjecture that this phenomenon may come from the encoding and tokenization schemes of GPT for numeric values. In particular, GPT may decide the output token by sequentially considering each set of digits in the numbers. We also observe the fragmentation behavior of decision boundaries of GPT, similar to the behavior of convolution neural networks [73]. See Table 9 in the Appendix for more detailed measurements that

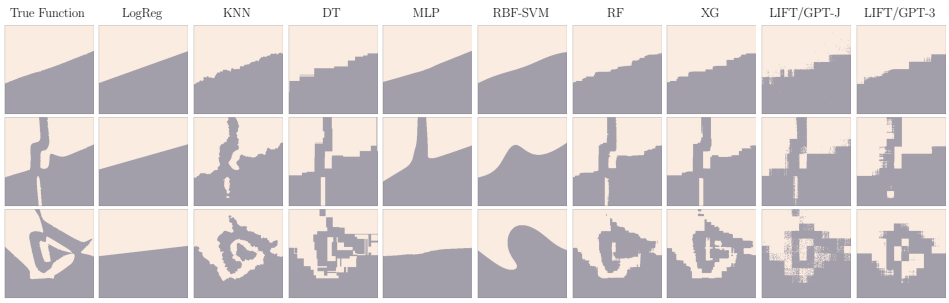



Figure 3: **Decision boundary of algorithms on synthetic datasets.** We first train a neural network on rolls dataset (Fig. 7), and take three snapshots of the trained neural network (shown in the first column). Then, for each snapshot, we define a dataset having labels as the network's prediction. Top to bottom: snapshots with more training epochs, corresponding to harder-to-classify datasets. We find that GPTs finetuned by LIFT adapts well on different boundaries.

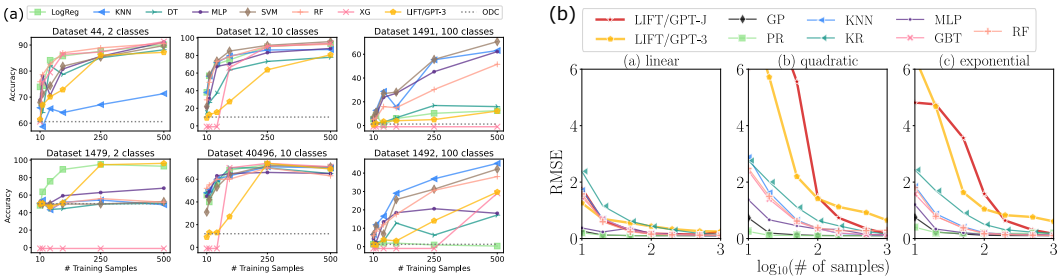



Figure 4: **Sample complexity of algorithms.** *(a) for classification,* LIFT/GPT-3 achieves comparably good sample complexity among algorithms. When tasks have more classes, LIFT needs more samples to start adapting to the task. *(b) for regression (function approximation),* LIFT/GPT needs slightly larger sample complexity to achieve similar performance. One can notice that LIFT/GPT achieves very competitive or even better performance when around 1000s of samples are given.

quantify the similarity between decision boundaries. Our results indicate that finetuning GPT using LIFT provably extracts useful features from labeled datasets to adapt to different decision boundaries, which is partly consistent with recent findings on inductive biases of language models [74].

### 4.3 How many samples does LIFT need?

We investigate if LIFT is sample efficient. Fig. 4(a) and Fig. 4(b) show results on classification and regression tasks, respectively. We find that the GPT model can be quickly finetuned to new tasks when LIFT is applied. For *classification*, when the number of classes increases, LIFT does need more samples for adaptation. This is probably because GPT requires more samples to learn the more complex input and output spaces of the data. For *regression*, we find that $n = 1000$ samples are sufficient for LIFT to have small RMSE, similar to other baselines. There exist some functions (quadratic, cosine, piece-wise) where LIFT has lower sample complexity than popular baselines.

Table 3: **Comparing in-context learning and finetuning with LIFT.** "LIFT/Full-Data" and "LIFT/Subset" columns are results of LIFT on the full dataset and on the subset of data (used in in-context setting). We focus on the comparison between in-context learning and LIFT/subset and put bold to the higher accuracy. The number of prompts for each dataset is selected to fit in the maximum context length of given LMs. We can see that LIFT/Full-data achieves the best performances overall. In-context learning or LIFT/Subset can be worse than ODC given limited training data.

| Dataset (ID) | #Prompts | ODC | GPT-J | | | GPT-3 | | |
|---|---|---|---|---|---|---|---|---|
| | | | In-Context | LIFT/Subset | LIFT/Full-data | In-Context | LIFT/Subset | LIFT/Full-data |
| Breast (13) | 35 | 70.69 | 56.90±19.51 | **58.62±2.44** | 64.94±11.97 | 62.07±1.41 | **70.69±0.00** | 71.26±1.62 |
| TAE (48) | 50 | 35.48 | **34.33±1.47** | 32.26±9.50 | 61.29±4.56 | **37.64±4.02** | 33.33±1.52 | 65.59±6.63 |
| Vehicle (54) | 14 | 25.88 | **25.49±0.55** | 26.04±1.69 | 64.31±2.37 | **28.82±2.10** | 23.73±2.27 | 70.20±2.73 |
| Hamster (893) | 43 | 53.33 | 48.89±3.14 | **60.00±10.88** | 55.55±16.63 | **57.78±6.29** | 53.33±0.00 | 53.33±0.00 |
| Customers (1511) | 29 | 68.18 | 56.06±17.14 | **59.85±2.84** | 85.23±1.61 | 60.61±1.42 | **63.26±6.96** | 84.85±1.42 |
| LED (40496) | 33 | 68.67 | 10.00±0.82 | **13.04±3.27** | 65.33±0.47 | 8.00±1.63 | **11.33±2.62** | 69.33±2.05 |

**Finetuning vs. in-context learning.** Compared to in-context learning, finetuning is a more appropriate solution when the unseen task requires large training data due to the limited context length

of LMs. However, it is unclear if this holds given smaller numbers of samples. We investigate this question by comparing LIFT with in-context learning on various data (see Table 3 in the Appendix). In many cases, LIFT/GPT achieve comparable or even better accuracies than the in-context learning using the same subsets of training data.

### 4.4 How robust is LIFT?

We investigate to which extent LIFT is robust to practical settings of training data: label corruption, noisy features, outlier samples, and adversarial examples. We provide further observations of robustness to class-imbalance in Appendix C.1.6.

**Robustness to label corruption.** We randomly corrupt a percentage of labels in the training data by assigning a new label under two different regimes. For *random errors,* we assign a new label in the label space with equal probability. For *systematic errors,* we assign all errors in a specific class to a singular other class as described in [75]. See Fig. 22(c) for results. We find that LIFT/GPT-3 follows the general trend of the baselines, not particularly outperforming or underperforming in terms of robustness, while LIFT/GPT-J is more sensitive to the corruption, which we also observe on KNN. We suspect that the current finetuning scheme of GPT-J is more sensitive than the black-box finetuning implementation of GPT-3. Nevertheless, we attribute the robust behavior of LIFT/GPT-3 to the ensemble effect of the underlying language models. See the Appendix for detailed results and visualization of the learned decision boundaries.

**Robustness to outliers.** We investigate the robustness of LIFT to outliers in regression tasks whose y is not consistent in terms of fitting $(\boldsymbol{X}, \mathbf{y})$. We inject four outliers into the training data of 200 samples. Fig. 22 compares the RAE values of LIFT and baselines with and without outliers when there are 2% outliers in the training set. Due to the space limit, we defer the full version in Sec. C. We observe that LIFT with both GPT-3 and GPT-J are the most robust among all eight methods. To be more specific, the performance of LIFT is almost not affected, while all the baselines suffer from a huge performance drop. To further study how robust LIFT is to outliers, we evaluate the performance of LIFT with datasets including 1%, 2%, 5%, 10%, and 20% outliers, respectively. We include median-3NN and median-5NN that serve as the optimal robust baselines [76], which make the prediction by taking the median of nearest neighbors, instead of mean. Fig. 24 shows that LIFT with GPT-3 is almost as robust as median-3NN and median-5NN, while LIFT with GPT-J is more vulnerable when the percentage of outliers increases.

**Robustness to adversarial samples.** We test how LIFT is robust against adversarial attacks. Since we don't have full access to the GPT-3 model, we tried the transfer attack. Table 4 shows the robustness results of LIFT and baselines for the MNIST classification problem. We tested on PGD attacks transferred from LeNet-5 and MLP. Given MNIST images having pixel values within [0,1], the perturbation radius is set to $\varepsilon \in [0, 0.01, 0.1, 0.3]$ and the adversarial examples are generated within $L_\infty$ ball. We compared the results for three networks: LeNet-5, MLP (having 2 hidden layers, each with 300 neurons and 100 neurons), and LIFT/GPT-3. It is shown that the adversarial attack on LeNet-5 and MLP are transferred to LIFT/GPT-3.

Table 4: **Accuracies (↑) under the perturbation on the input feature of MNIST data.**

| Source | **PGD attack on LeNet-5** | | | **PGD attack on MLP** | | |
|---|---|---|---|---|---|---|
| Target | LeNet-5 | MLP | LIFT/GPT-3 | LeNet-5 | MLP | LIFT/GPT-3 |
| $\varepsilon = 0$ | 99.22 | 98.09 | 98.15 | 99.22 | 98.09 | 98.15 |
| $\varepsilon = 0.01$ | 97.27 | 97.77 | 44.88 | 99.15 | 96.89 | 44.46 |
| $\varepsilon = 0.1$ | 26.80 | 93.99 | 33.66 | 96.98 | 23.12 | 23.62 |
| $\varepsilon = 0.3$ | 0.00 | 36.62 | 20.31 | 41.51 | 0.00 | 20.29 |

## 5 Evaluation of LIFT-specific Learning Properties

In this section, we study the behavior of LIFT in a more fine-grained manner.

### 5.1 Is LIFT calibrated?

We investigate whether LIFT is *calibrated*, where prediction logits/samples reflects the confidence, by exploring how LIFT performs for different levels of noise. We apply LIFT/GPT per each synthetic regression dataset of 1000 training samples. We then perform inferences with varying temperatures 20 times to calculate the standard deviation $\sigma$ for each prediction. Also, we increase the noise level from 0 to 2 on feature value $x$ with $\sigma(x) = (x + 10)/10$ to show how the noise level influences the model's prediction behavior. Fig. 5 shows our results on six synthetic datasets. We observe that the standard deviation of LIFT's prediction aligns well with that of the observations across tasks. These results imply that LIFT/GPT is calibrated.

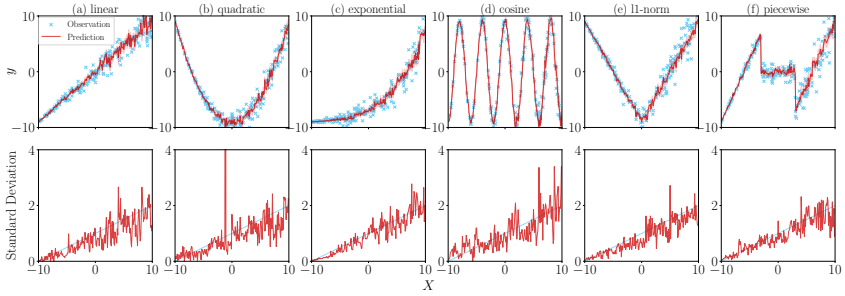



Figure 5: **Investigating calibrated prediction effect of LIFT.** Prediction standard deviations of LIFT/GPT align well to the observations (top), across datasets, implying the well calibration.

Table 5: **The effect of using feature names on LIFT**. We compare classification accuracy (↑) of LIFT/GPT-3 when provided feature names *are and are not* incorporated to the prompts on `CMC (23)`, `TAE (48)`, `Vehicle (54)`, `German`, and `Census`. We provide five versions of LIFT when feature names are correctly incorporated (Correct-Names) or randomly shuffled (Shuffled-Names). As a result, all LIFT models achieve better performance than ODC. LIFTs with correct feature names achieve the best accuracies on `TAE`, `Vehicle`, and `German` datasets while achieving the comparable accuracies to the best model on the `CMC` and `Census` dataset. **Two prompt designs result in the same template for the* `Vehicle`.

| Dataset (ID) | ODC | LIFT | | | | | |
|---|---|---|---|---|---|---|---|
| | | W/o Names I | W/o Names II | Shuffled-Names I | Shuffled-Names II | Correct-Names I | Correct-Names II |
| CMC (23) | 42.71 | **57.74±0.89** | 57.40±1.37 | 56.27±2.06 | 57.06±4.24 | 57.40±1.09 | 56.27±2.22 |
| TAE (48) | 35.48 | 65.59±6.63 | 66.67±5.48 | 60.22±6.72 | 64.52±8.53 | **69.89±9.31** | **69.89±6.72** |
| Vehicle (54) | 25.88 | 70.20±2.73 | 71.96±3.09 | 70.20±5.34 | 69.22±2.72 | **75.29±2.04*** | |
| German | 70.00 | 71.33 ± 5.20 | 67.83 ± 2.72 | 73.00 ± 1.87 | 71.67 ± 0.94 | 72.33 ± 1.70 | 74.17 ± 1.25 |
| Census | 75.43 | 81.41 ± 0.24 | - | 81.68 ± 0.21 | 81.73 ± 0.12 | 81.89 ± 0.32 | 81.56 ± 0.14 |

### 5.2 Does LIFT benefit from incorporating feature names?

Unlike standard machine learning algorithms, LIFT can be provided with the context information by incorporating the feature names and task descriptions in the prompts. Intuitively, this incorporation may improve the sample complexity of LIFT as the prior knowledge already learned in the pretraining phase may help LIFT predict better. We empirically verify this intuition and show our results in Table 5 for several classification tasks using pretrained GPT-3 models. We design seven prompts templates to assess how incorporating feature names affects the performance of LIFT (see more details in Sec. B.4). Shown in Table 5 is our evaluation. At first, all LIFT models outperform ODC with large accuracy gaps, indicating that they are all properly trained. Overall, we observe that correctly incorporating feature names helps boost the performances of LIFT across the datasets. On the `CMC` and `Census` datasets, the performance of LIFT remains almost similar after adding feature names. Also, if we use similar prompts with shuffled feature names (`Shuffled-Names I, II`), then the performance of LIFT drops by a significant margin. Thus, this confirms that the aforementioned performance improvements are indeed due to proper prompting with correct feature/value association. Properly incorporating the feature names into LIFT may improve the performance.

## 6 Improving LIFT with Existing Techniques

Previous sections illustrate the effectiveness of standard LIFT in learning non-language tasks. We provide two techniques for improving LIFT.

### 6.1 Two-stage finetuning for LIFT with synthetic pretext tasks

In Sec. 4.3, we observe that LMs need a sufficient number of samples to start adapting to the non-language tasks. We suspect that LMs need early samples to learn the task description: spaces of input and label, and the template [46].

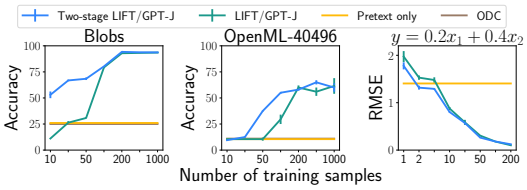



Figure 6: **Two-stage finetuning.** The inter-finetuning (blue) applies LIFT first on synthetic pretext data before the real datasets, outperforming finetuning (green) when training data is small.

We suggest utilizing synthetic data instead of real training data to describe the task for LMs, thus reducing the sample complexity. Motivated by recent works in pretraining for few-shot learning [77], we propose a two-stage finetuning approach. Given a task, we generate a simple mixture of Gaussian synthetic data with the same number of features and classes as the real dataset. We first apply LIFT on the LM with this synthetic dataset for a few epochs, then continue the procedure with the real training dataset. Fig. 6 shows our comparison between two-stage LIFT (inter-finetuning) with the original LIFT (finetuning). We find that intermediate finetuning improves the performances of LIFT on both classification and regression tasks when the number of training samples is small.

### 6.2 Data augmentation

In various classification tasks, data augmentation [79] has been considered a simple tool for improving the generalization performance. We focus on mixup [78], a popular data augmentation that makes virtual samples by mixing the features/labels of real samples. We test whether mixup helps in the LIFT framework. Specifically, we finetune GPTs using the prompt of mixup samples and compare the accuracy of GPTs finetuned with original real samples. Given each sample $(x, y)$ having feature $x$ and label $y \in \{0, 1\}$, we convert it to $(x, \tilde{y})$ for $\tilde{y} = 10y$, so that the label is either 0 or 10. Given $(x_1, \tilde{y}_1)$ and $(x_2, \tilde{y}_2)$, we make mixup sample $(x^{\text{mix}}, y^{\text{mix}}) = (\lambda x_1 + (1-\lambda)x_2, \lambda\tilde{y}_1 + (1-\lambda)\tilde{y}_2)$, for randomly chosen convex combination coefficient $\lambda \in \{0, 0.1, 0.2, \cdots, 1.0\}$. We fine-tune GPTs using the input-output pair $(x^{\text{mix}}, y^{\text{mix}})$. Table 6 shows our results for various OpenML datasets. We observe that mixup does not help LIFT on both GPT-3 and GPT-J, while mixup guarantees sufficient gain for logistic regression.

Table 6: **Classification accuracy (%) on various OpenML datasets, with/without mixup [78].** For both GPT-J and GPT-3, mixup is not helping LIFT, while improving the accuracy in logistic regression training.

| Setting / OpenML ID | | 467 | 683 | 736 | 855 | 859 | 874 | 911 | 1005 |
|---|---|---|---|---|---|---|---|---|---|
| LogReg | no mixup | 85.0 | 76.7 | 72.4 | 63.3 | 95.7 | 97.3 | 67.4 | 60.0 |
| | **mixup** | 90.0 | 81.1 | 76.5 | 66.0 | 98.3 | 100.0 | 70.1 | 64.9 |
| LIFT/GPT-J | **no mixup** | 90.9 | 66.7 | 60.9 | 83.0 | 93.3 | 30.0 | 88.0 | 53.5 |
| | mixup | 45.5 | 25.0 | 65.2 | 34.0 | 93.3 | 10.0 | 40.0 | 25.58 |
| LIFT/GPT-3 | **no mixup** | 90.9 | 63.9 | 65.2 | 81.0 | 100.0 | 76.7 | 86.0 | 73.6 |
| | mixup | 78.8 | 47.2 | 59.4 | 68.67 | 95.56 | 63.33 | 82.0 | 65.12 |

## 7 Discussion and Conclusion

We study the Language-Interfaced FineTuning (LIFT) framework for solving non-language downstream tasks without changing models' architecture or loss function. LIFT first converts labeled samples into sentences and then finetune pretrained LMs on the sentence dataset using the standard finetuning method and loss function. Via an extensive empirical study on a suite of non-language tasks, we show that LIFT/GPT performs relatively well on low-dimensional classification and regression tasks. We find LIFT/GPT robust against outliers and label corruption and can properly calibrate the predictions. We further improve LIFT with the use of in-context feature names and two-stage finetuning methods. While LIFT still suffers from the high dimensionality of features in regression tasks and the large numbers of classified categories, the findings of LIFT open a few interesting directions for future research investigations. Nevertheless, our work is arguably one of the first to thoroughly study the efficacy of LIFT with pretrained language models on standard regression and classification tasks, paving the way for enabling "no-code machine learning with language models."

**Limitations and open questions.** Despite promising performances on various tasks and settings, we observe some limitations of LIFT to basic learning tasks. LIFT/GPT do not perform well if the features have high dimensions (for regression), or when the output space is large (for classification). Also, the context length of LIFT is restricted to the context length of LMs and LIFT/GPT is memory-inefficient. To address this issue, one can combine LIFT with memory-efficient LMs such as LinTransformer [80]. Besides, our works open some interesting questions for future works. First, based on the similarity of decision boundaries, do LMs and LIFT/GPT share similar behaviors to ensemble methods or decision tree? Secondly, are LMs universal models that can adapt well to any modalities and domains? Lastly, can LIFT/GPTs adapt better for regression task using more sophisticated encoding schemes for numeric features?

**Social impacts.** Future research should also investigate potential fairness issues of applying LIFT. Based on large language models, LIFT might have embedded bias targeting at certain social groups. Especially when feature names are included in the training prompts, it is possible that the models will be more sensitive to social biases and thus might make unfair and even harmful predictions. We leave measuring the embedded bias in LIFT as one of interesting future directions.

# References


[1] David Forsyth and Jean Ponce. *Computer vision: A modern approach.* Prentice hall, 2011.

[2] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An image is worth 16x16 words: Transformers for image recognition at scale. 2021.

[3] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[5] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[6] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 314–315, 2019.

[7] Shih-Chung B Lo, Heang-Ping Chan, Jyh-Shyan Lin, Huai Li, Matthew T Freedman, and Seong K Mun. Artificial convolution neural network for medical image pattern recognition. *Neural networks*, 8(7-8):1201–1214, 1995.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[10] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[13] David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

[14] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[15] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[16] Dhanesh Ramachandram and Graham W Taylor. Deep multimodal learning: A survey on recent advances and trends. *IEEE signal processing magazine*, 34(6):96–108, 2017.

[17] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[18] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*, 2021.

[19] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[20] Belinda Z Li, Jane Yu, Madian Khabsa, Luke Zettlemoyer, Alon Halevy, and Jacob Andreas. Quantifying adaptability in pre-trained language models with 500 tasks. *arXiv preprint arXiv:2112.03204*, 2021.

[21] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[22] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR, 2021.

[23] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[24] Roy Schwartz, Maarten Sap, Ioannis Konstas, Leila Zilles, Yejin Choi, and Noah A Smith. Story cloze task: Uw nlp system. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 52–55, 2017.

[25] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[26] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[27] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30, 2017.

[28] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Guihong Cao, Daxin Jiang, Ming Zhou, et al. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*, 2020.

[29] Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

[30] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heinz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *arXiv preprint arXiv:2111.01243*, 2021.

[31] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[33] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[34] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[35] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[36] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[37] Alex Tamkin, Trisha Singh, Davide Giovanardi, and Noah D. Goodman. Investigating transferability in pretrained language models. In *FINDINGS*, 2020.

[38] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[39] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[40] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[41] Ting Chen, Mario Lucic, Neil Houlsby, and Sylvain Gelly. On self modulation for generative adversarial networks. *arXiv preprint arXiv:1810.01365*, 2018.

[42] Jason Phang, Thibault Févry, and Samuel R Bowman. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv preprint arXiv:1811.01088*, 2018.

[43] Raphael Rubino and Eiichiro Sumita. Intermediate self-supervised learning for machine translation quality estimation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4355–4360, 2020.

[44] Nikita Moghe, Mark Steedman, and Alexandra Birch. Cross-lingual intermediate fine-tuning improves dialogue state tracking. *arXiv preprint arXiv:2109.13620*, 2021.

[45] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[46] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[47] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020.

[48] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Kathryn Tunyasuvunakool, Olaf Ronneberger, Russ Bates, Augustin Žídek, Alex Bridgland, et al. High accuracy protein structure prediction using deep learning. *Fourteenth Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*, 22:24, 2020.

[49] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34, 2021.

[50] Sercan O Arık and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *AAAI*, volume 35, pages 6679–6687, 2021.

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

[52] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[53] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[54] Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

[55] Wei-Tsung Kao and Hung-Yi Lee. Is bert a cross-disciplinary knowledge learner? a surprising finding of pre-trained models' transferability. *arXiv preprint arXiv:2103.07162*, 2021.

[56] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

[57] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2019.

[58] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5307–5315, Hong Kong, China, November 2019. Association for Computational Linguistics.

[59] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[60] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, et al. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4, 2015.

[61] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[62] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[63] Yann LeCun. The mnist database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[64] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[65] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[66] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon Shaolei Du, Ken-Ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. In *International Conference on Learning Representations*, 2021.

[67] Medical insurance dataset (kaggle). `https://www.kaggle.com/datasets/mirichoi0218/insurance`.

[68] Pınar Tüfekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

[69] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. World Scientific, 1992.

[70] Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. 2008.

[71] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[72] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. October 2021.

[73] Gowthami Somepalli, Liam Fowl, Arpit Bansal, Ping Yeh-Chiang, Yehuda Dar, Richard Baraniuk, Micah Goldblum, and Tom Goldstein. Can neural nets learn the same model twice? investigating reproducibility and double descent from the decision boundary perspective. *arXiv preprint arXiv:2203.08124*, 2022.

[74] Charles Lovering, Rohan Jha, Tal Linzen, and Ellie Pavlick. Predicting inductive biases of pre-trained models. In *International Conference on Learning Representations*, 2020.

[75] Yanyao Shen and Sujay Sanghavi. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*, pages 5739–5748. PMLR, 2019.

[76] Peter J Huber. Robust statistics. In *International encyclopedia of statistical science*, pages 1248–1251. Springer, 2011.

[77] Mingda Chen, Jingfei Du, Ramakanth Pasunuru, Todor Mihaylov, Srini Iyer, Veselin Stoyanov, and Zornitsa Kozareva. Improving in-context few-shot learning via self-supervised training. *arXiv preprint arXiv:2205.01703*, 2022.

[78] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[79] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[80] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[81] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[82] Emily Sheng, Kai-Wei Chang, Premkumar Natarajan, and Nanyun Peng. The woman worked as a babysitter: On biases in language generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3407–3412, Hong Kong, China, November 2019. Association for Computational Linguistics.

[83] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. A reductions approach to fair classification. In *International Conference on Machine Learning*, pages 60–69. PMLR, 2018.

[84] Yuji Roh, Kangwook Lee, Steven Euijong Whang, and Changho Suh. Fairbatch: Batch selection for model fairness. *arXiv preprint arXiv:2012.01696*, 2020.

# Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The contribution and scope of this paper has been summarized in the abstract and the last part of Introduction.
   (b) Did you describe the limitations of your work? [Yes] We wrote the limitations in Sec.7.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] This has been discussed in Sec.7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our paper conforms to the ethics guideline.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code, data, and instructions are provided in the shared anonymous GitHub repository.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] All the training details are discussed in the paper and they can be found at the anonymous GitHub repository.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All the experiments are run multiple times and at the tables mean and standard deviation of those runs are presented.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] This has been described in Sec.3.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our work uses public datasets/models, and has been cited properly, in the paper and GitHub repo.
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# Appendix

We first describe details of our datasets in (A) and the implementation of all models in (B). For the experimental results, we provide the detailed and extended results for all findings presented in the main paper in (C), with more visualizations, score tables, and additional evaluations on more functions and datasets. In (D), we present additional findings which have not been discussed so far in the main paper, including the robustness to Gaussian noise, effects of different numerical types, and contradicting prompts on LIFT and investigate whether LIFT can perform ridge regression. We provide a deeper discussion on the social impact of our work in (E).


**A Datasets** **17**

A.1 Classification datasets . . . . . 17

A.2 Regression datasets . . . . . 17

**B LIFT and Baseline Implementation** **18**

B.1 Pretrained language models and baselines . . . . . 18

B.2 Computing resources . . . . . 18

B.3 Hyperparameter selection . . . . . 18

B.4 Prompt templates . . . . . 28

**C Detailed and Extended Results of Findings in the Main Paper** **28**

C.1 Results for basic findings of LIFT (Section 4) . . . . . 28

C.1.1 Robustness to adversarial examples . . . . . 28

C.1.2 Inductive bias and decision boundary . . . . . 28

C.1.3 Classification: varying the difficulty of synthetic data . . . . . 29

C.1.4 Regression: performances, interpolation, and extrapolation . . . . . 30

C.1.5 Sample complexity . . . . . 31

C.1.6 Robustness to class-imbalance . . . . . 31

C.1.7 Robustness to outliers . . . . . 33

C.1.8 Robustness to label corruption . . . . . 33

C.2 How do different language models affect LIFT? . . . . . 36

C.3 Results for LIFT-specific learning properties (Section 5) . . . . . 37

C.3.1 Does LIFT benefit from incorporating feature names? . . . . . 37

C.4 Results for improving techniques of LIFT (Section 6) . . . . . 37

C.4.1 Two-stage intermediate finetuning for LIFT . . . . . 37

**D Additional Experiments and Findings (NOT Discussed in the Main Paper)** **39**

D.1 Robustness to Gaussian noises . . . . . 39

D.2 How do different numerical types affect LIFT/GPT? . . . . . 40

D.3 Can LIFT perform Ridge regression? . . . . . 40

D.4 The effect of contradicting prompts on finetuning . . . . . 40

**E Discussion of Social Impacts** **42**

# A Datasets

## A.1 Classification datasets

Table 7 summarizes the datasets used for classification tasks. Besides, we use two additional types of synthetic datasets: **neural-net-based datasets** used for understanding the inductive biases of algorithms (in Sec. 4.2) and **Gaussian pretext datasets** for the two-stage finetuning experiments (in Sec. 6.1). We train a 2-layer neural network with Tanh activation functions using the *rolls* dataset shown in Fig. 7, for generating *neural-net-based datasets*. We use the model checkpoint at different epochs (10, 40, 80, 210, 320, 490) as the labeling function to generate the synthetic dataset (with 2,000 samples). The decision boundaries of networks trained on more epochs are more visually complex, and hence the corresponding classification tasks are more complicated. Given a target dataset of $n$ classes and $d$ features, *Gaussian pretext datasets* is generated as follows: using `scikit-learn`[2], we randomly generate datasets of $n$ clusters, where each cluster has 100 normally distributed samples in the $d$-dimensional space.

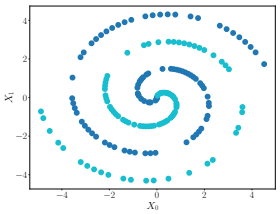



Figure 7: **Rolls dataset used for generating neural-net-based datasets.** This dataset consists of 300 samples from 2 classes.

| Data Type | Dataset | ID | No. Features | No. Classes | No. Instances | Note |
|---|---|---|---|---|---|---|
| Synthetic | 9Gaussians | 1 | 2 | 9 | 2000 | - |
| | Blobs | 2 | 2 | 4 | 2000 | - |
| | Circle | 3 | 2 | 2 | 2000 | non-linear boundary |
| | TwoCircles | 6 | 2 | 2 | 2000 | non-linear boundary |
| | Moons | 4 | 2 | 2 | 2000 | - |
| Tabular (OpenML) | wholesale-customers | 1511 | 9 | 2 | 440 | Imbalance |
| | pollution | 882 | 16 | 2 | 60 | 1 symbolic feature |
| | spambase | 44 | 58 | 2 | 4601 | 1 symbolic feature |
| | hill-valley | 1479 | 101 | 2 | 1212 | 1 symbolic feature |
| | tae | 48 | 6 | 3 | 151 | Categorical data |
| | cmc | 23 | 10 | 3 | 1473 | Meaningful feature Names |
| | wine | 187 | 13 | 3 | 178 | Integral features |
| | vehicle | 54 | 19 | 4 | 846 | Meaningful feature Names |
| | LED-display-domain-7digit | 40496 | 8 | 10 | 500 | 1 symbolic feature |
| | optdigits | 28 | 65 | 10 | 5620 | 1 symbolic feature |
| | mfeat-factors | 12 | 217 | 10 | 2000 | 1 symbolic feature |
| | one-hundred-plants-margin | 1491 | 54 | 100 | 1600 | 1 symbolic feature |
| | one-hundred-plants-shape | 1492 | 65 | 100 | 1600 | 1 symbolic feature |
| | one-hundred-plants-texture | 1493 | 65 | 100 | 1599 | 1 symbolic feature |
| Vision | MNIST | - | 784 | 10 | 70k | - |
| | Permuted MNIST | - | 784 | 10 | 70k | - |
| | Permuted Fashion MNIST | - | 784 | 10 | 70k | - |

Table 7: **Classification datasets.** We have three non-language types of data: synthetic data, real tabular data and vision data. We use five synthetic datasets. For the real tabular data, we select datasets from OpenML with a wide range of number of features, types of features, number of classes, number of training samples. For the vision datasets, we use MNIST, Fashion MNIST and their permuted variants.

## A.2 Regression datasets

Our regression tasks consists of both synthetic and real datasets.
(**Synthetic datasets**) To assess the regression performance of LIFT in different datasets, we generate synthetic datasets based on six different functions types, including smooth functions, non-smooth functions, and non-continuous functions:

1. Linear functions $y = f(\boldsymbol{x}) = \boldsymbol{x}^\top \mathbf{1}/p$
2. Quadratic functions $y = f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{I} \boldsymbol{x}/p$, where $\boldsymbol{I}$ is the identity matrix
3. Continuous exponential function $y = f(\boldsymbol{x}) = \sum_{i=1}^{p} e^{0.2x_i}/p$
4. Cosine functions $y = f(\boldsymbol{x}) = \sum_{i=1}^{p} \cos(0.5\pi x_i)/p$

[2]https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

5. (Non-smooth) $\ell 1$-norm function $y = f(\boldsymbol{x}) = \|\boldsymbol{x}\|_1 / p$
6. (Non-continuous) Piecewise function

$$f(\mathbf{x}) = \frac{1}{p}\sum_{i=1}^{p} \tilde{f}(x_i) := \left\{ \begin{array}{cc} x_i - 1 & -10 \leq x_i < -3, \\ 0 & -3 \leq x_i < 3, \\ x_i + 1 & 3 \leq x_i \leq 10. \end{array} \right\}.$$

We let $\mathrm{x}_i \sim \mathrm{Unif}(-10, 10)$ for each coordinate $i$, and $\sigma = 0.1$ by default. We normalize all the functions above for fair comparison among different functions so that their $y \in [-9, 9]$ when $\boldsymbol{x} \in [-10, 10]^p$. In particular, to assess whether LIFT is better at dealing with positive numbers or integers, we generate additional datasets by further manipulating the $(\mathbf{x}, \mathrm{y})$ distribution of exponential and cosine functions. For datasets with only positive numbers, for each $p \in \{1, 2, 50, 120\}$, we generate the dataset $\mathrm{x}_i \sim \mathrm{Unif}(0, 10)$ for $i \in [p]$. For datasets with only integers, for all $i \in [p]$, we further expand the range of the $\mathrm{x}_i$ from $(0, 10)$ to $(0, 300)$ for positive datasets, from $(-10, 10)$ to $(-150, 150)$ otherwise, and then round down all inputs to integers.

(**Real datasets**) We consider four different types of real datasets: Medical Insurance dataset [67] with 1,338 samples and 6 features, Combined Cycle Power Plant (CCPP) dataset [68] with 9,568 samples and 4 features, Servo [69] dataset with 167 samples and 4 features, and Student [70] dataset with 649 samples and 33 features. In particular, the Medical Insurance dataset and Student dataset contain feature names that can be interpreted using common knowledge (see feature lists in Table 21), while CCPP and Servo do not.

# B LIFT and Baseline Implementation

This section provides details of our models and implementation. We describe our pretrained language models and the baseline implementations (B.1), the computing resources used for running experiments (B.2), and how to finetune and select the hyperparameters (B.3).

## B.1 Pretrained language models and baselines

**Pretrained language models** Our main results are with two pretrained language models: GPT-J [61] and GPT-3 [19]. We mainly focus on GPT-J for the reproducibility purpose and provide additional results on GPT-3 as reference. For GPT-J, we use a quantized version[3] of 6 billion parameters with 8-bit weights. For GPT-3, we use the GPT-3 OpenAI API[4]. In Sec. C.2, we compare two previous mentioned models with three bigger versions of GPT-3 (`Baggage`, `Curie`, `Davinci`). The largest one is `Davinci`-GPT-3 containing approximately 175 billions of parameters.

Since GPTs can output any language token, the output might not be appropriate for the desired task. For example, GPT might output non-numerical words for regression task approximating a function $f : \mathbb{R}^n \to \mathbb{R}$. In such a case, we categorize this as *invalid output*.

**Baselines** For XGBoost, we use the open-source `XGBoost` module[5]. For other baselines, we implement them using `scikit-learn` module[6].

## B.2 Computing resources

For experiments on GPT-J, we use GPU computing from two 24GB-RTX3090 GPUs and AWS EC2 instances[7] (`p3.8xlarge`, `p3.2xlarge`). For other models, we run experiments on CPU instances.

## B.3 Hyperparameter selection

In finetuning GPT-J, we use Adam-8bit optimizer implementation[8] with weight decay of 0.01 and 6 warm-up steps. The learning rate is set to `1e-4` for the synthetic/ OpenML datasets, and `1e-5` for

[3] https://huggingface.co/hivemind/gpt-j-6B-8bit
[4] https://openai.com/api/
[5] https://xgboost.readthedocs.io/
[6] https://scikit-learn.org/
[7] https://aws.amazon.com/ec2/
[8] https://huggingface.co/hivemind/gpt-j-6B-8bit

the vision datasets. We use a linear learning scheduler for the optimizer. For classification, the batch size depends on the number of features of the datasets. We set the batch size to be 128, 32, 16, 2 for datasets with the number of features being no greater than 2, between 2 and 6, between 6 and 20, and greater than 20, respectively. For regression, we set batch size as 4 by default, and reduce it to 1 to avoid the memory issue when the number of features increase. For GPT-3, we use the API provided by OpenAI to perform black-box GPT-3 finetuning with the default setting. Our implementation is with PyTorch framework[9].

We perform hyperparameter selection based on validation results for all methods for a fair comparison. The hyperparameter tuning scheme for all methods is detailed as follows:

**Classification methods**

- LIFT/GPT-J: number of epochs $\in \{5, 10, 15\}$
- LIFT/GPT-3: learning rate multiplier $\in \{0.05, 0.1, 0.2\}$
- Random Forest (*RF*): maximum depth $\in \{3, 5, 10\}$, minimum number of samples required to split an internal node $\in \{2, 5, 10\}$
- Decision Tree (*DT*): maximum depth of the tree $\in \{3, 5, 20\}$ and criterion $\in \{$Gini impurity, Shannon information gain$\}$
- Support Vector Machine (*SVM*): kernel $\in \{$linear kernel, radial basis function$\}$, regularization parameter $\in \{1, 10, 100\}$
- Multilayer Perceptron (*MLP*): initial learning rate $\in \{0.001, 0.01, 0.1\}$
- Logistic regression (*LogReg*): inverse of regularization strength $\in \{1, 10, 100\}$
- K-Nearest Neighbor (*KNN*): number of neighbors to use $\in \{1, 3, 5\}$, power parameter for the Minkowski metric $\in \{1, 2\}$
- XGBoost (*XG*): maximum depth $\in \{3, 5, 10\}$

**Regression methods**

- LIFT/GPT-J: number of epochs $\in \{2, 6, 10\}$
- LIFT/GPT-3: learning rate multiplier $\in \{0.05, 0.1, 0.2\}$
- Polynomial Regression (*PR*): no hyperparameter selection but fixed the degree at 3 since higher-order polynomial regression introduces out-of-memory error especially for high-dimensional datasets
- K-Nearest Neighbor (*KNN*): number of neighbors $\in \{2, 5, 8\}$
- Kernel Regression (*KR*): Gamma parameter of Radial Basis Kernel $\in \{0.01, 0.1, 1\}$
- Multilayer Perceptron (*MLP*): initial learning rate $\in \{0.0001, 0.001, 0.01\}$
- Gradient Boosting Tree (*GBT*): learning rate $\in \{0.001, 0.01, 0.1\}$
- Random Forest (*RF*): maximum depth $\in \{4, 6\}$
- Gaussian Process (*GP*): the number of optimizer restarts used to find parameters of the kernel that maximize the log marginal likelihood $\in \{5, 10\}$.

[9]https://pytorch.org/

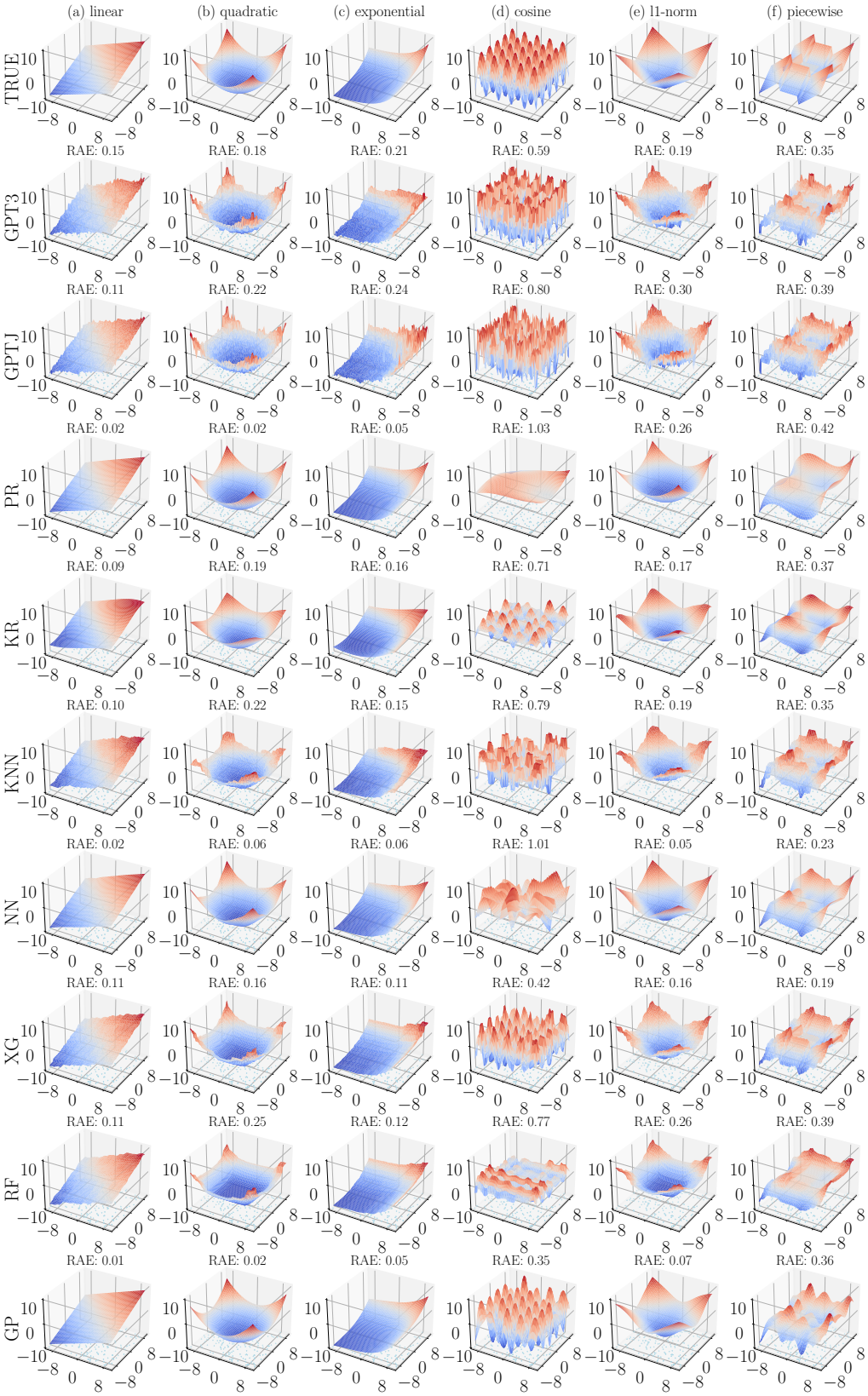



Figure 8: **Performance of LIFT/GPTs and baselines in approximating various functions.** The first row visualizes the true values of the functions, and the second & third rows visualize the predicted values of LIFT/GPTs after finetuning for the corresponding regression tasks with **200 training samples**. We compared with other baselines with same training samples.

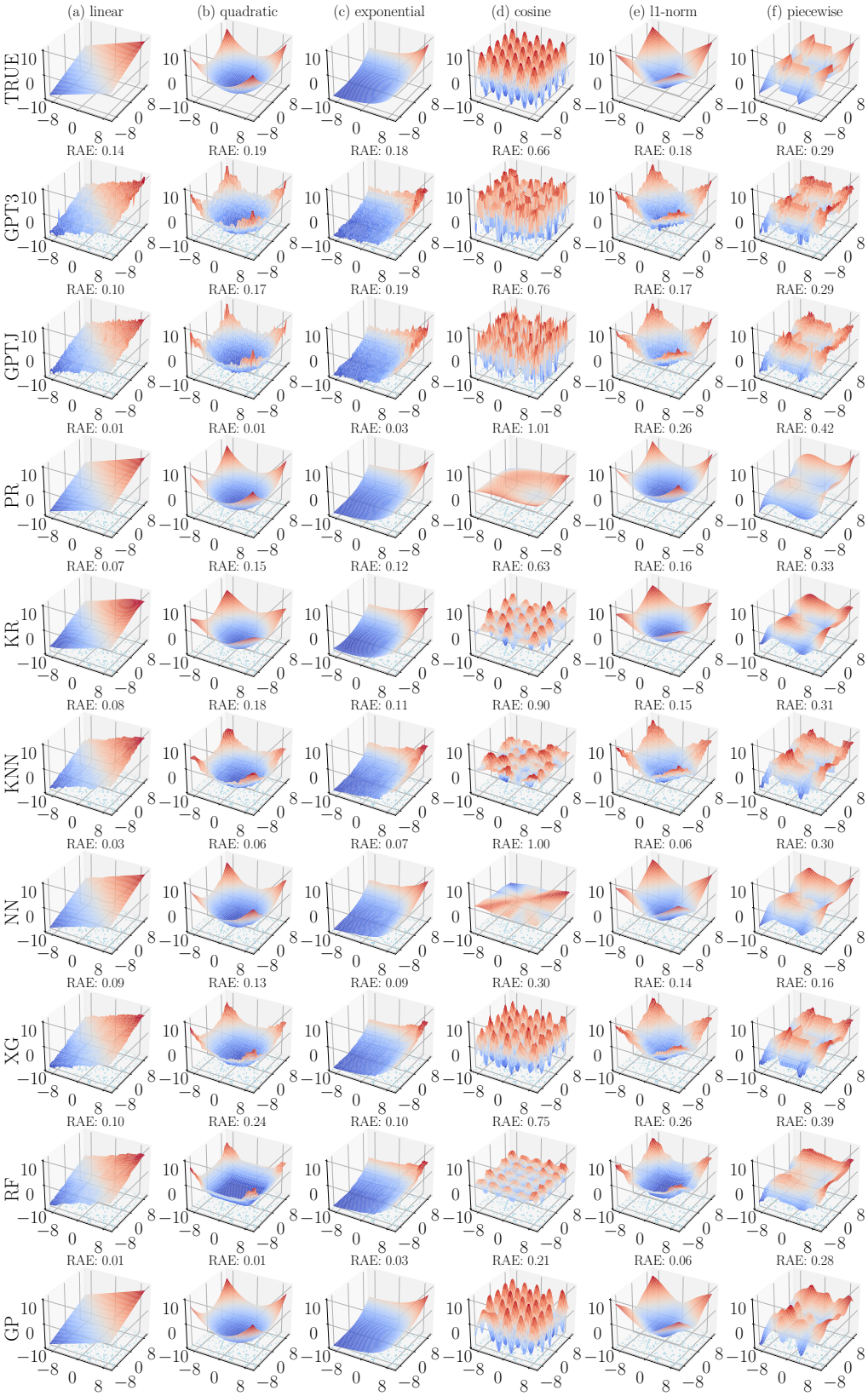



Figure 9: **Performance of LIFT/GPTs in approximating various functions.** The first row visualizes the true values of the functions, and the second & third rows visualize the predicted values of LIFT/GPTs after finetuning for the corresponding regression tasks with **300 training samples**. We compared with other baselines with same training samples.

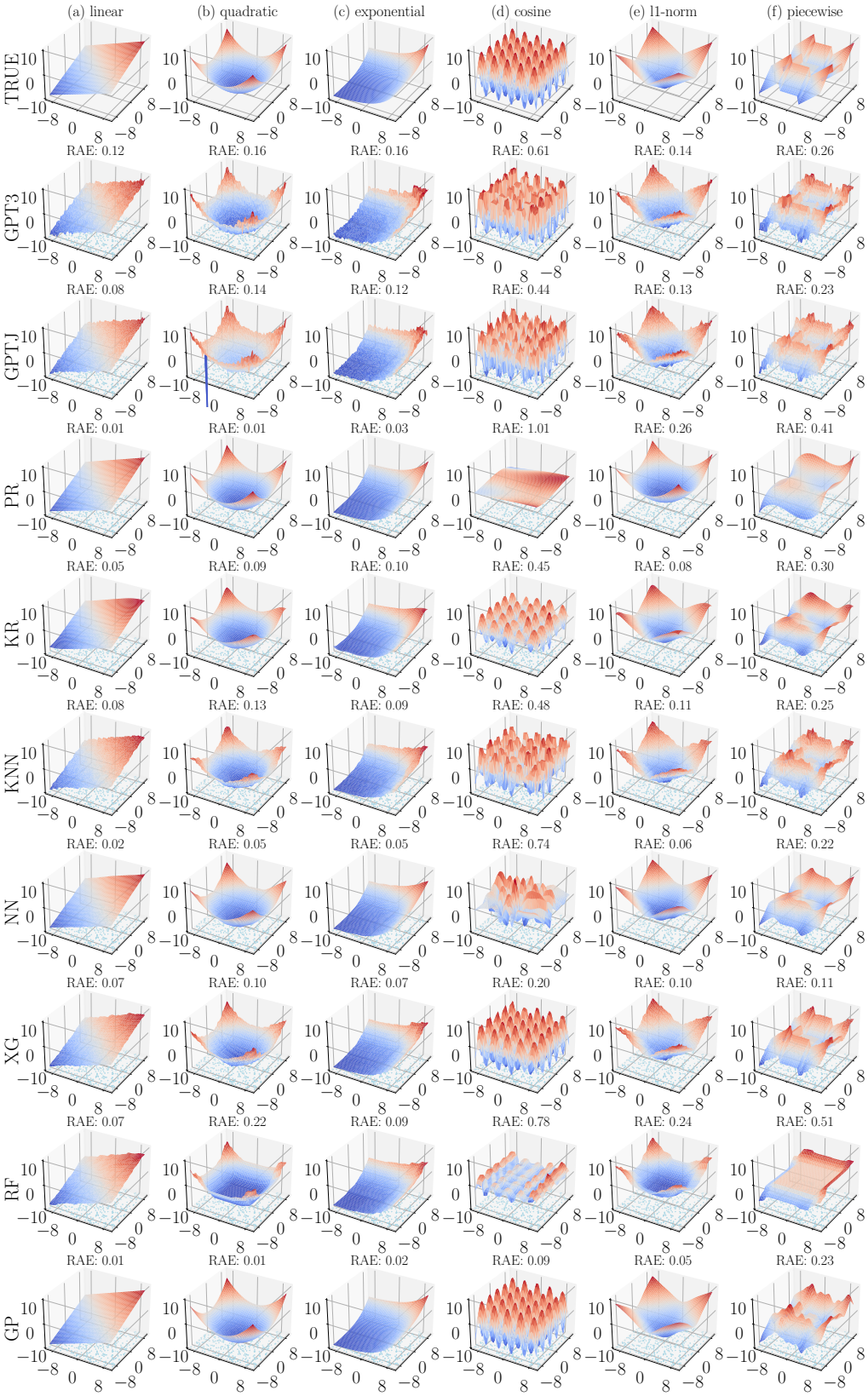



Figure 10: **Performance of LIFT/GPTs in approximating various functions.** The first row visualizes the true values of the functions, and the second & third rows visualize the predicted values of LIFT/GPTs after finetuning for the corresponding regression tasks with **500 training samples**. We compared with other baselines with same training samples.

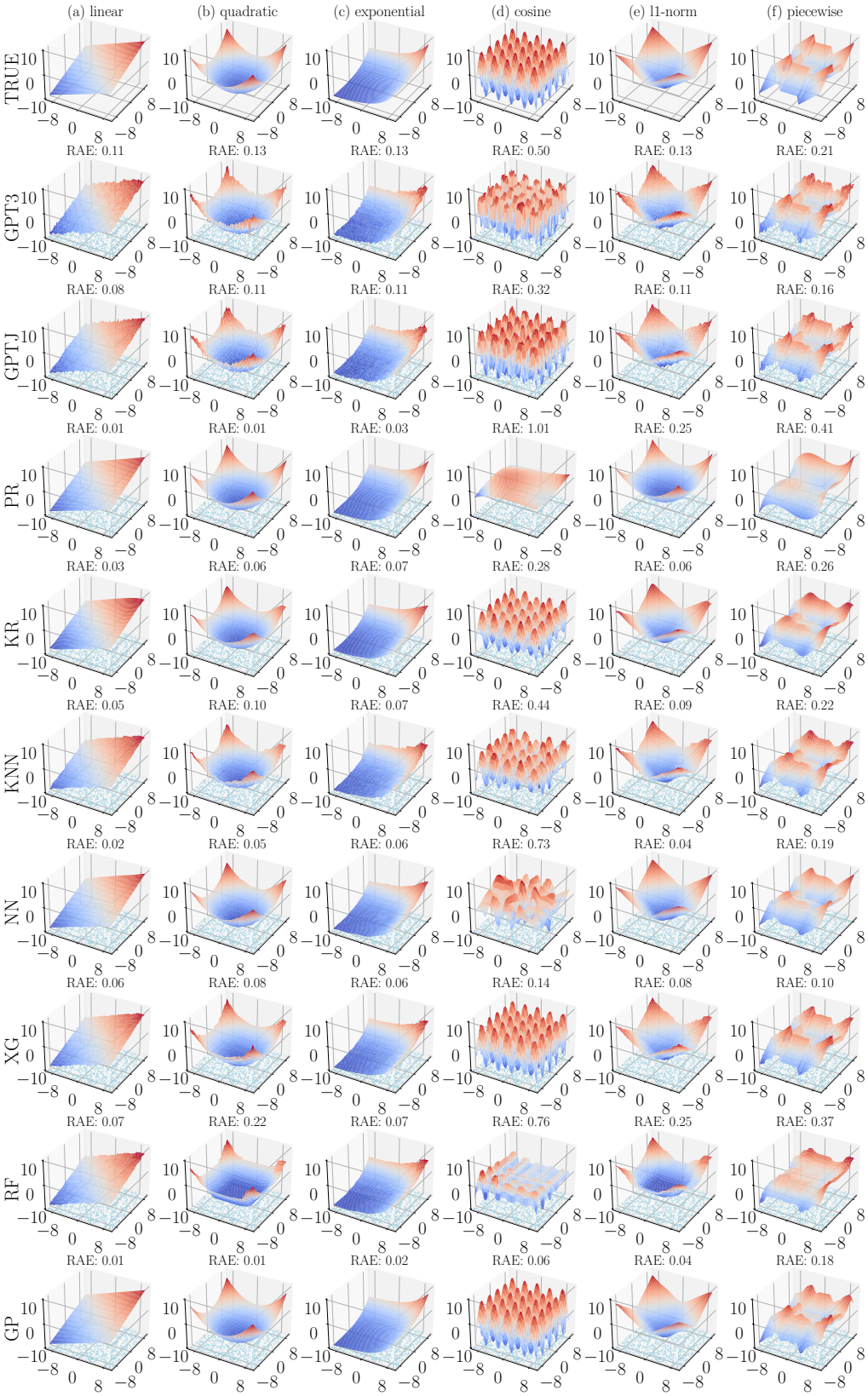



Figure 11: **Performance of LIFT/GPTs in approximating various functions.** The first row visualizes the true values of the functions, and the second & third rows visualize the predicted values of LIFT/GPTs after finetuning for the corresponding regression tasks with **1000 training samples**. We compared with other baselines with same training samples.

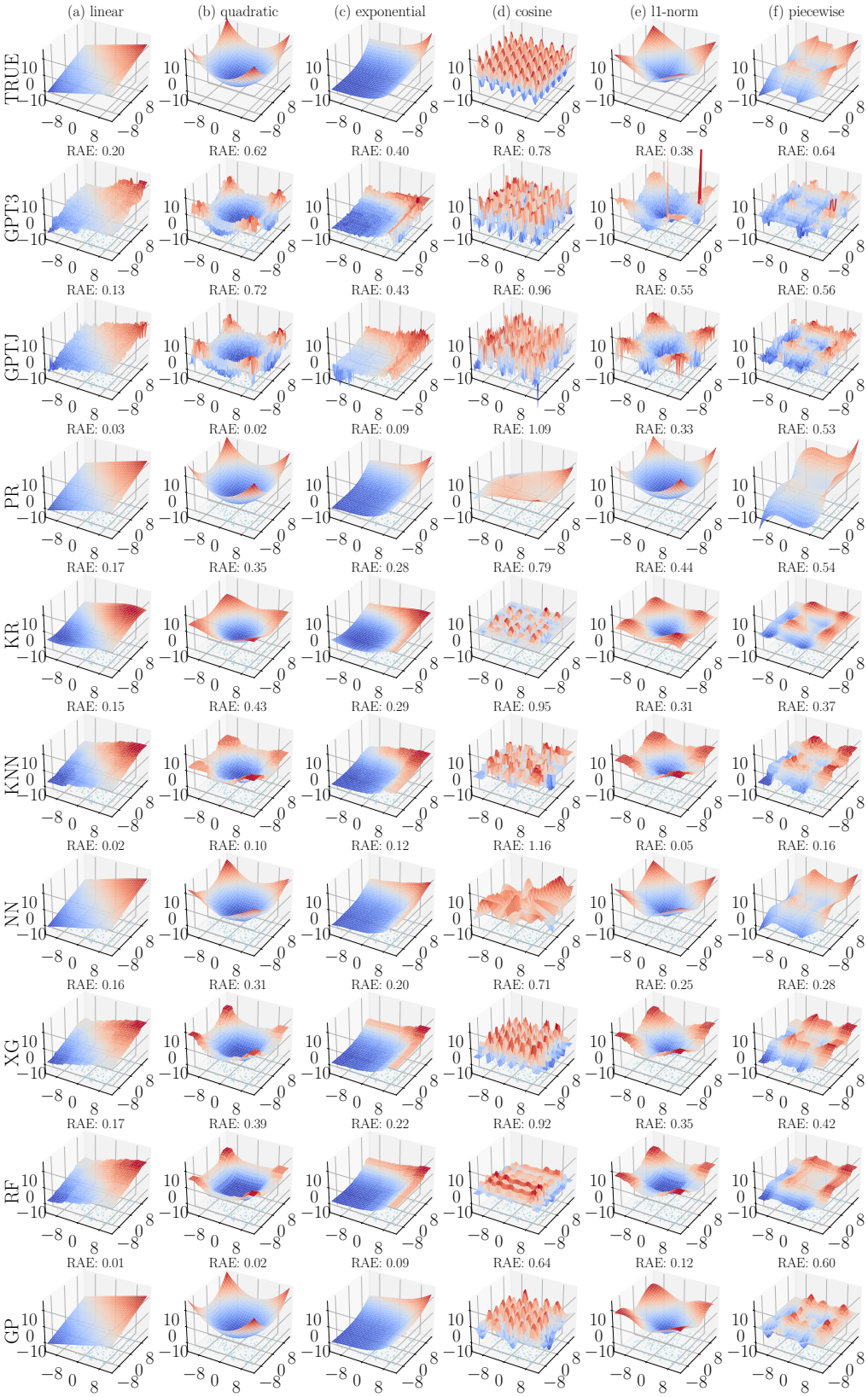



Figure 12: **Extrapolation performance of LIFT/GPTs in approximating various functions.** The dataset is generated by $\mathrm{Unif}([-10, 10]^2)$ with **200 training samples**, and we test the performance of various methods in interval $[-12, 12]^2$.

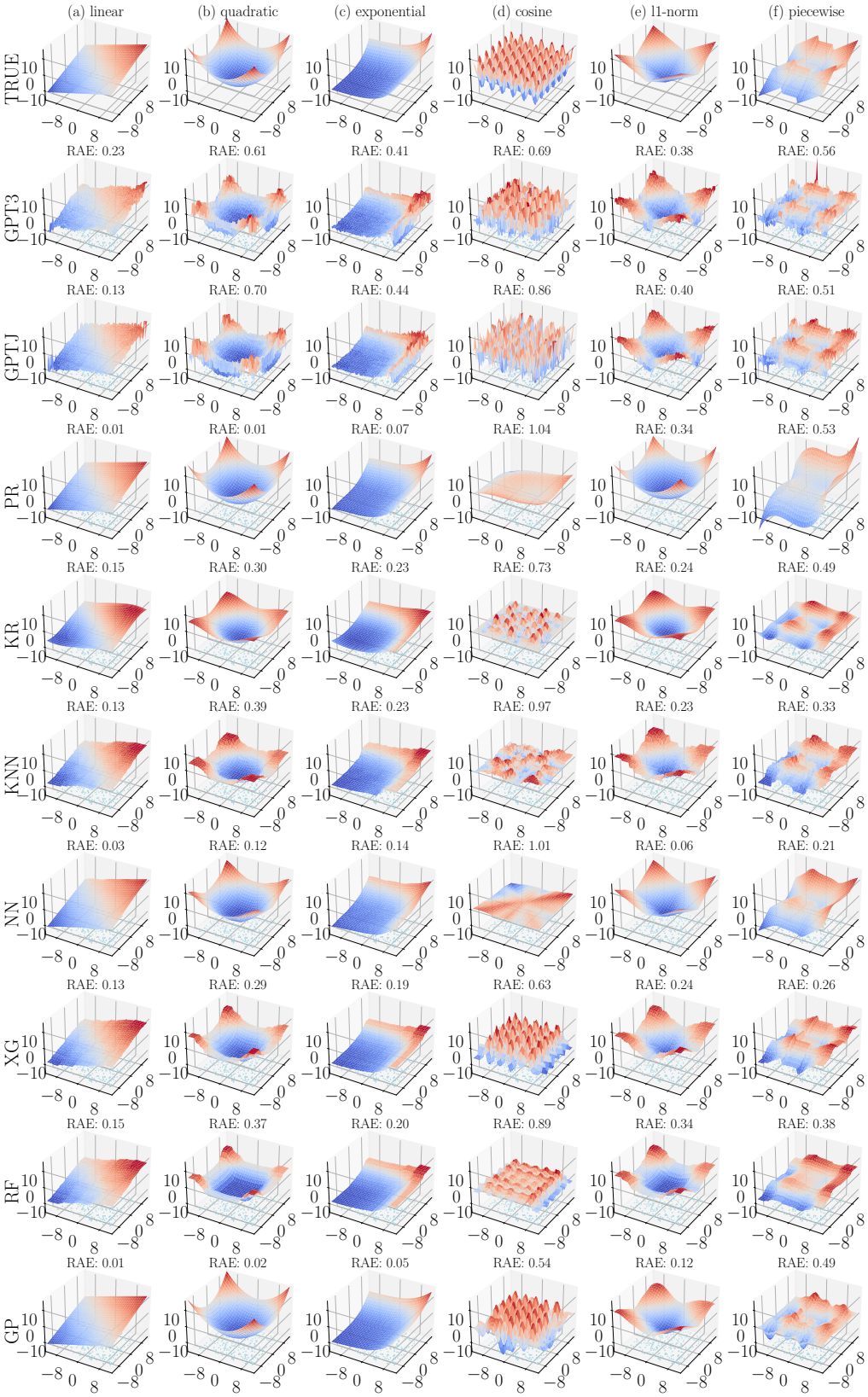



Figure 13: **Extrapolation performance of LIFT/GPTs in approximating various functions.** The dataset is generated by $\mathrm{Unif}([-10, 10]^2)$ with **300 training samples**, and we test the performance of various methods in interval $[-12, 12]^2$.

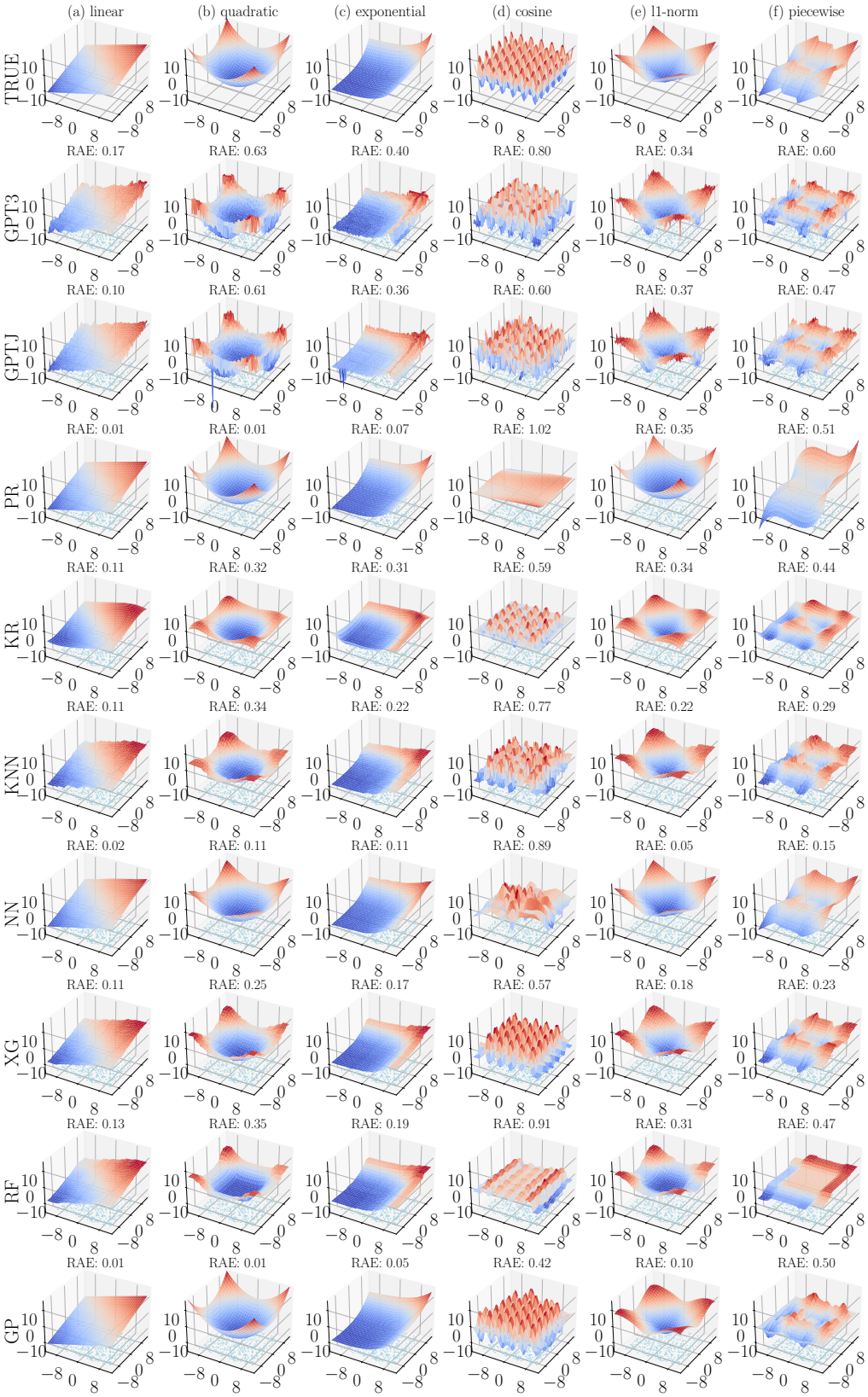



Figure 14: **Extrapolation performance of LIFT/GPTs in approximating various functions.** The dataset is generated by $\mathrm{Unif}([-10, 10]^2)$ with **500 training samples**, and we test the performance of various methods in interval $[-12, 12]^2$.

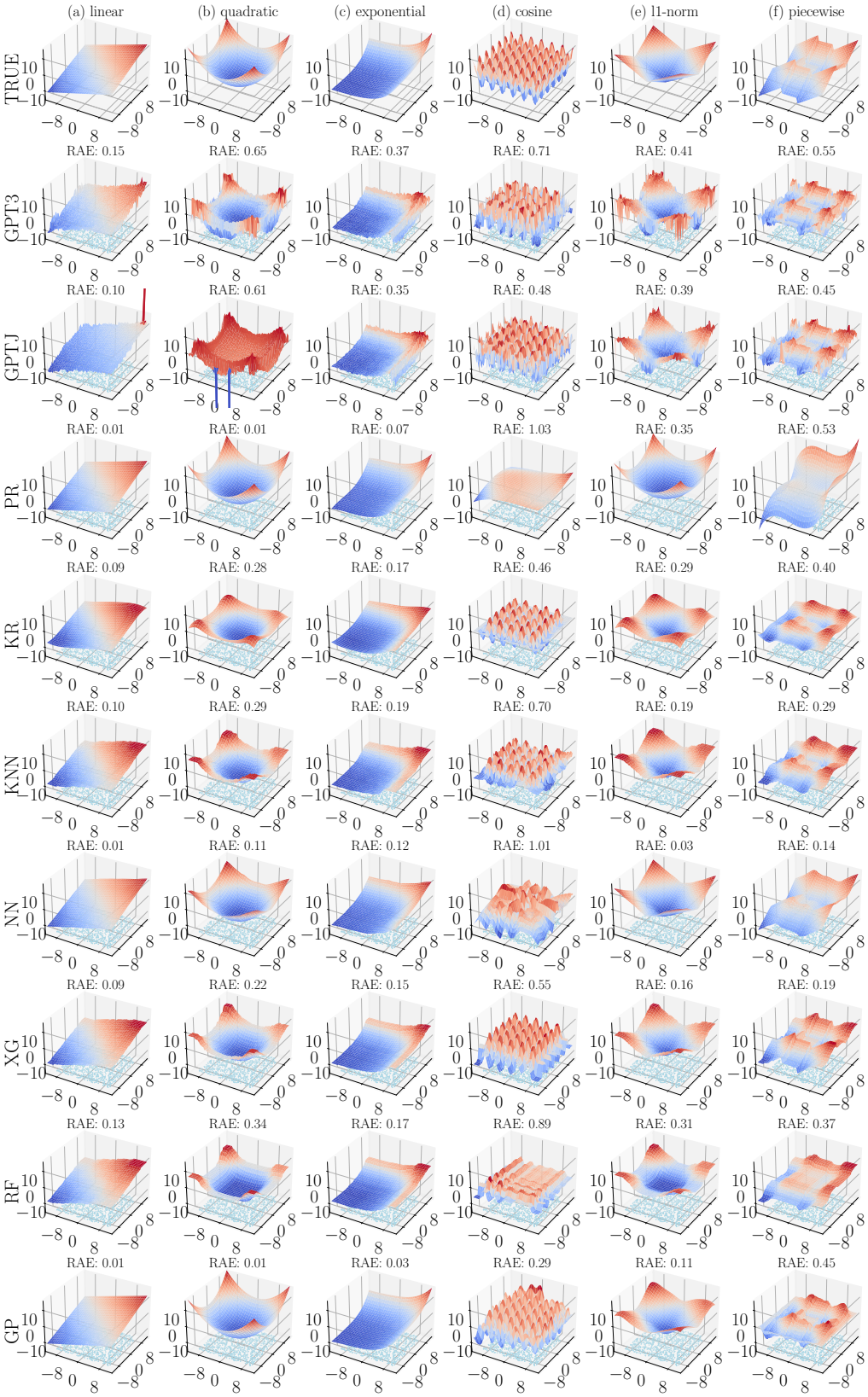



Figure 15: **Extrapolation performance of LIFT/GPTs in approximating various functions.** The dataset is generated by $\mathrm{Unif}([-10, 10]^2)$ with **1000 training samples**, and we test the performance of various methods in interval $[-12, 12]^2$.

### B.4 Prompt templates

In addition to the basic prompts we introduced in Sec. 1, we design seven prompts templates to assess how incorporating feature names and different templates affects the performance of LIFT. For instance, consider a data sample "$\boldsymbol{x} = (\texttt{English speaker}, 23, 3, \texttt{summer}, 19)$, $y = 3$" from TAE dataset where the feature names are "`native speaker, instructor, course, semester, class size`", and the target attribute is `teaching performance`. We can incorporate the contextual information by either simply replacing the "$x_i$" in the prompts with the corresponding feature names or converting this sample into a coherent sentence. Meanwhile, we also investigate how shuffled feature names and misleading questions affect the performance of LIFT by designing the prompts accordingly. For illustration purposes, we provide the example of the five prompt templates as below.

- `(W/O Names)` "`When we have` $x_1 = 1, x_2 = 23, x_3 = 3, x_4 = 1, x_5 = 19$`, what should be` $y$ `value?`"
- `(W/O Names)` "`When we have` $x_1 = 1, x_2 = 23, x_3 = 3, x_4 = 1, x_5 = 19$`, what should the` $y$ `be?`"
- `(Correct-Names I)` "`When we have native speaker=English speaker, course instructor=`$23$`, course=`$3$`, semester=summer, class size=19, how is the teaching performance?`"
- `(Correct-Names II)` "`In the course 3 offered in the summer semester, there was a native English-speaking teaching assistant and an instructor whose ID is 23. How is the teaching performance?`"
- `(Shuffled-Names I)` "`When we have semester=English speaker, class size=`$23$`, semester=`$3$`, course instructor=summer, native speaker=19, how is the teaching performance?`"
- `(Shuffled-Names II)` "`In the course summer offered in the 3 semester, there was a 19 teaching assistant and an instructor whose ID is summer. How is the teaching performance?`"
- `(Misleading)` "`In the course 3 offered in the summer semester, there was a native English-speaking teaching assistant and an instructor whose ID is 23. What is the color of it?`"

We note that the sentence generated using the `(Shuffled-Names II)` template can be incoherent.

## C Detailed and Extended Results of Findings in the Main Paper

In this section, we provide the extended version of experimental results presented in the main paper, including the primary findings of LIFT (C.1), the specific learning properties of LIFT (C.3), and improving techniques for LIFT (C.4).

### C.1 Results for basic findings of LIFT (Section 4)

#### C.1.1 Robustness to adversarial examples

Table 8 shows the results on the robustness of LIFT against adversarial examples for the MNIST classification problem. We applied transfer attack to LIFT, i.e., generated adversarial examples from a source network (LeNet-5) and tested those samples on LIFT. We measured the adversarial robustness under PGD attack with different attack radius $\varepsilon$, and tested on three networks for comparison: MLP (having 2 hidden layers, each with 300 neurons and 100 neurons), LIFT/GPT-J and LIFT/GPT-3. It is shown that LIFT is vulnerable to transfer attacks with even a small attack radius ($\varepsilon = 0.01$) but maintains nonzero robust accuracy as $\varepsilon$ increases.

#### C.1.2 Inductive bias and decision boundary

**Quantifying the similarity of decision boundaries** Based on the qualitative evaluation of decision boundaries, we conjecture that LIFT/GPT-3 can adapt to different boundaries and share similar

| Network name | Source<br>LeNet-5 | Transfer<br>MLP | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|
| Clean acc. | 99.23 | 97.93 | 96.88 | 98.15 |
| Robust acc. ($\varepsilon = 0.01$) | 97.27 | 97.77 | 27.25 | 44.88 |
| Robust acc. ($\varepsilon = 0.1$) | 26.80 | 93.99 | 20.12 | 33.66 |
| Robust acc. ($\varepsilon = 0.3$) | 0.00 | 36.62 | 14.66 | 20.31 |

Table 8: **The result of transfer attack on LIFT/GPT, where the adversarial examples are generated from LeNet-5 trained on clean MNIST samples.** The adversarial examples are generated by PGD attack [81], and we reported the robust accuracy (%) for various attack radii $\varepsilon$. LIFT/GPTs are more vulnerable to adversarial examples than multi-layer perceptron (MLP).

| Dataset ID | SVM (kernel) | | | LogReg | KNN ($k$) | | | DT (depth $D$) | | MLP (width $W$) | | | XG | RF (# estimators $E$) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | poly | rbf | sigmoid | | K=1 | K=3 | K=5 | D=3 | D=5 | W=10 | W=100 | W=200 | | E=20 | E=50 | E=100 |
| 1 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 76.00 | 100.00 | 100.00 | 100.00 | 100.00 | 97.50 | 100.00 | 100.00 | 100.00 |
| 2 | 98.50 | 97.50 | 92.00 | 97.50 | 94.00 | 95.50 | 96.00 | 94.50 | 91.00 | 97.00 | 97.00 | 97.00 | 93.50 | 94.50 | 94.00 | 94.00 |
| 3 | 54.00 | 93.50 | 48.00 | 51.00 | 85.50 | 88.50 | 88.50 | 67.00 | 80.00 | 84.00 | 92.50 | 92.50 | 87.50 | 85.00 | 87.50 | 89.00 |
| 4 | 90.50 | 97.50 | 74.50 | 87.50 | 99.00 | 99.00 | 99.00 | 91.00 | 98.50 | 92.50 | 98.50 | 98.50 | 97.50 | 95.50 | 94.50 | 96.00 |
| 6 | 63.00 | 62.50 | 48.50 | 62.50 | 59.50 | 58.00 | 59.00 | 57.50 | 59.50 | 60.50 | 64.00 | 65.00 | 58.50 | 64.00 | 63.50 | 65.00 |
| 23 | 59.50 | 61.00 | 68.00 | 63.50 | 50.50 | 52.50 | 51.00 | 65.00 | 72.00 | 62.50 | 53.50 | 53.50 | 63.00 | 56.00 | 55.50 | 59.00 |
| 871 | 66.00 | 74.50 | 66.00 | 67.50 | 60.00 | 63.00 | 64.50 | 69.50 | 66.00 | 60.00 | 59.50 | 58.00 | 63.50 | 64.50 | 62.50 | 67.00 |
| 1467 | 100.00 | 100.00 | 100.00 | 94.50 | 92.50 | 98.50 | 100.00 | 94.50 | 91.00 | 98.00 | 97.50 | 97.50 | 94.50 | 99.50 | 99.50 | 100.00 |
| 40496 | 88.00 | 87.00 | 82.50 | 91.50 | 74.00 | 76.50 | 84.50 | 69.00 | 84.50 | 85.50 | 94.50 | 91.50 | 92.00 | 91.50 | 91.50 | 90.50 |
| Average | 79.94 | **85.94** | 75.5 | 79.5 | 79.4 | 81.28 | 82.50 | 76.00 | 82.50 | 82.22 | **84.11** | 83.72 | 83.06 | 83.39 | 81.17 | **84.5** |

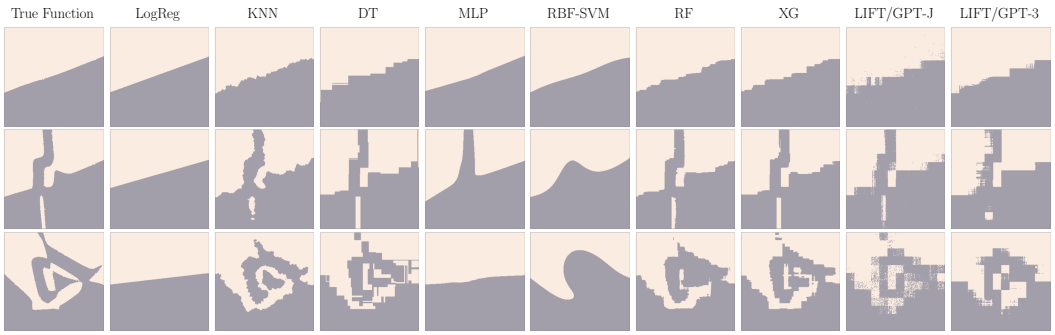
Table 9: **Quantifying the similarity between decision boundaries of LIFT/GPT-3 and those of various baselines.** Each score is a percentage similarity of both LIFT/GPT-3 and the baseline classifying a point with the same class. For example, a score of 100 for XG signifies that LIFT/GPT-3 classified all sampled test points in the same manner as XG, regardless of their true dataset accuracy.

behaviors to tree-based ensemble methods. We carry out experiments to assess this conjecture by quantifying the similarity between the decision boundaries of LIFT/GPT-3 and those of the baselines. In Table 9, each score is a percentage similarity of both LIFT/GPT-3 and the baseline classifying testing points as the same class. We randomly sample 200 data points from the original dataset to train both baselines and LIFT/GPT-3. Then, we randomly sample another 200 from the remaining data points for evaluation. The predictions of LIFT/GPT-3 are then compared with respect to each baseline to generate the final accuracies. For instance, a score of 100 for XG signifies that LIFT/GPT-3 classifies all sampled test points in the same manner as XG, regardless of their true dataset accuracy. The LIFT/GPT-3 models are trained with a learning rate multiplier of 0.1 for 10 epochs. In terms of accuracy, while we observe no similar discernible pattern between LIFT/GPT-3 and the baselines, we find that RBF-SVM, Random Forest (E=100), and MLP (W=100) appear to share the most similar behavior pattern to LIFT/GPT-3.

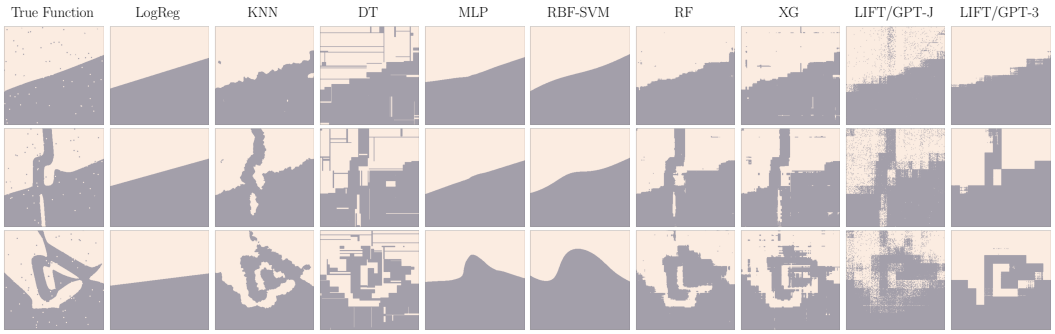
**Decision boundary under label corruption** Fig. 16 visualizes the decision boundaries of LIFT/GPTs and baselines on neural-net-based synthetic datasets with label corruptions (0%, 5%, 20%). This is a full version of the visualization given in Fig. 3 in the main body. We observe that LIFT can capture the rough shapes of the decision boundaries in all three settings. When the level of corruption increases, decision trees and XGboost are the most affected methods. While roughly capturing the boundary, LIFT/GPT-J shows more noisy predictions. In contrast, LIFT/GPT-3 displays great robustness against the corrupted labels while capturing the correct boundary shapes. We conjecture the difference in behaviors of LIFT/GPT-J and LIFT/GPT-3 come from the practical implementation of GPT-J and GPT-3. Nevertheless, this experiment shows the adaptability of LIFT/GPT-J to different settings of decision boundaries.

### C.1.3 Classification: varying the difficulty of synthetic data

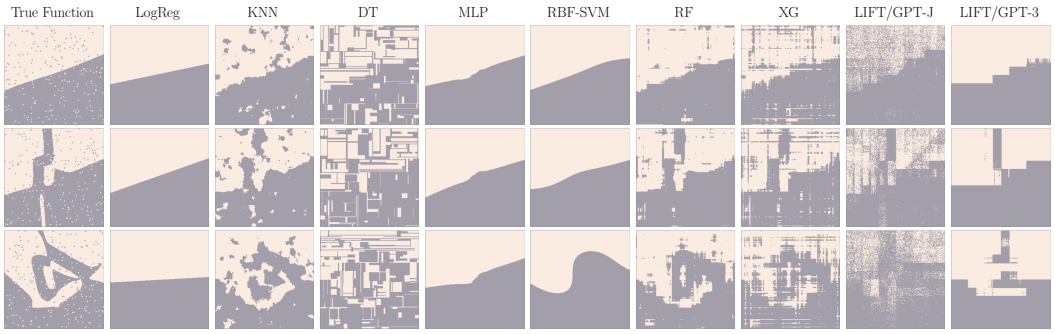
We assess how well LIFT/GPTs adapt to different shapes of decision boundaries using synthetic datasets generated by neural networks. Fig. 17 shows the comparison between evaluated methods when the classification difficulty varies by using different network checkpoints at different epochs. As the difficulty increases or the level of corruption increases, all methods tend to decrease classification accuracy. We provide three settings of training data: clean data, 5% label corruption, and 20% label




(a) Clean data




(b) Label corrupted data (corruption ratio is 5%)




(c) Label corrupted data (corruption ratio is 20%)

Figure 16: **Decision boundary of methods under label corruption.** We provide a full version of Fig. 3, the decision boundary for neural-net-based synthetic dataset. Here we consider three corruption settings: (a) clean data, (b) 5% of the training data are label corrupted, (c) 20% of the training data are label corrupted. Similar to Fig. 3, each sub-figure contains three rows, and from the top row to the bottom row, the classification problem is becoming more challenging. We can see that LIFT/GPTs can roughly estimate the true decision boundaries. Note that LIFT/GPT-3 is one of the most robust methods under the label corruption.

corruption. We observe that LIFT/GPT-3 outperforms logistic regression and decision tree, especially when the label corruption is up to 20%.

#### C.1.4 Regression: performances, interpolation, and extrapolation

We investigate the interpolation and extrapolation performance of LIFT in terms of regression tasks. Fig. 18 and Fig. 19 visualize the interpolation and extrapolation of various methods. Table 10 and Table 11 report the performance of LIFT/GPT-J and LIFT/GPT-3 for 1D synthetic datasets and 2D synthetic datasets with 200 training samples, respectively. Since handling high-dimensional datasets requires large memory, due to the limitation of the resources, we fail to get the results of LIFT/GPT-J

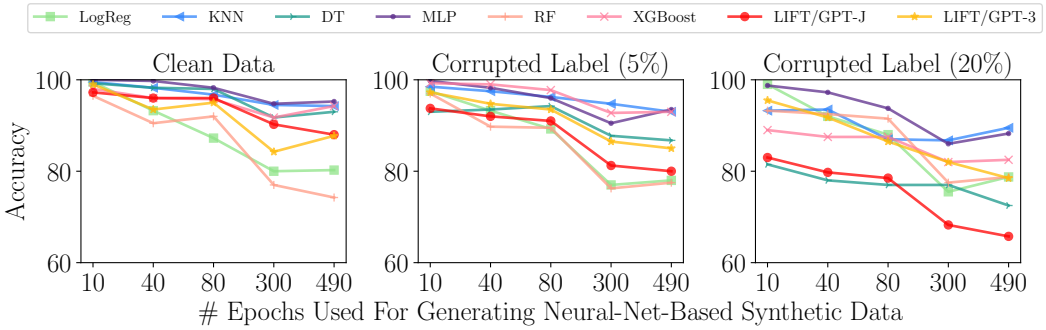



Figure 17: **How accuracy(%) changes as the target classification problem becomes more complex, i.e., the ground-truth decision boundary becomes more complex.** The x-axis shows the number of epochs we used to train the neural network on the `roll` dataset. The network becomes more complex as the number of epochs increases. We illustrate how we measure the complexity in Sec. C.1.3. (Left to right) Performances on three cases: clean data and label-corrupted training data with probabilities of 5% and 20%.

for 50D and 120D synthetic datasets. Table 12 and Table 13 report the performance of LIFT/GPT-3 for 50D and 120D synthetic datasets.

| RAE | PR | KR | KNN | MLP | GBT | RF | GP | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|---|
| Linear | $0.01 \pm 0.0$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.03 \pm 0.0$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.01 \pm 0.0$ | $0.08 \pm 0.0$ | $0.06 \pm 0.0$ |
| Quadratic | $0.01 \pm 0.0$ | $0.05 \pm 0.0$ | $0.05 \pm 0.0$ | $0.03 \pm 0.0$ | $0.06 \pm 0.0$ | $0.05 \pm 0.0$ | $0.01 \pm 0.0$ | $0.11 \pm 0.0$ | $0.13 \pm 0.0$ |
| Exponential | $0.04 \pm 0.0$ | $0.07 \pm 0.0$ | $0.05 \pm 0.0$ | $0.02 \pm 0.0$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.01 \pm 0.0$ | $0.11 \pm 0.0$ | $0.09 \pm 0.0$ |
| Cosine | $1.05 \pm 0.0$ | $0.12 \pm 0.0$ | $0.14 \pm 0.0$ | $0.38 \pm 0.1$ | $0.15 \pm 0.0$ | $0.35 \pm 0.0$ | $0.04 \pm 0.0$ | $0.38 \pm 0.1$ | $0.44 \pm 0.1$ |
| L1norm | $0.23 \pm 0.0$ | $0.06 \pm 0.0$ | $0.05 \pm 0.0$ | $0.03 \pm 0.0$ | $0.06 \pm 0.0$ | $0.06 \pm 0.0$ | $0.03 \pm 0.0$ | $0.10 \pm 0.0$ | $0.09 \pm 0.0$ |
| Piecewise | $0.45 \pm 0.0$ | $0.17 \pm 0.0$ | $0.08 \pm 0.0$ | $0.08 \pm 0.0$ | $0.06 \pm 0.0$ | $0.07 \pm 0.0$ | $0.10 \pm 0.0$ | $0.15 \pm 0.0$ | $0.17 \pm 0.0$ |

Table 10: **Comparison of regression methods in approximating various functions when $p = 1, \sigma = 0.1, (\mathbf{x}, \mathrm{y}) \in \mathbb{R}$.**

| RAE | PR | KR | KNN | MLP | GBT | RF | GP | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|---|
| Linear | $0.03 \pm 0.0$ | $0.09 \pm 0.0$ | $0.12 \pm 0.0$ | $0.04 \pm 0.0$ | $0.12 \pm 0.0$ | $0.12 \pm 0.0$ | $0.01 \pm 0.0$ | $0.12 \pm 0.0$ | $0.19 \pm 0.0$ |
| Quadratic | $0.03 \pm 0.0$ | $0.16 \pm 0.0$ | $0.17 \pm 0.0$ | $0.06 \pm 0.0$ | $0.15 \pm 0.0$ | $0.25 \pm 0.0$ | $0.02 \pm 0.0$ | $0.28 \pm 0.1$ | $0.22 \pm 0.0$ |
| Exponential | $0.04 \pm 0.0$ | $0.15 \pm 0.0$ | $0.13 \pm 0.0$ | $0.07 \pm 0.0$ | $0.09 \pm 0.0$ | $0.11 \pm 0.0$ | $0.04 \pm 0.0$ | $0.19 \pm 0.0$ | $0.20 \pm 0.0$ |
| Cosine | $1.04 \pm 0.0$ | $0.74 \pm 0.0$ | $0.83 \pm 0.1$ | $1.06 \pm 0.0$ | $0.41 \pm 0.0$ | $0.80 \pm 0.0$ | $0.31 \pm 0.0$ | $0.82 \pm 0.2$ | $0.65 \pm 0.1$ |
| L1norm | $0.24 \pm 0.0$ | $0.17 \pm 0.0$ | $0.19 \pm 0.0$ | $0.06 \pm 0.0$ | $0.15 \pm 0.0$ | $0.29 \pm 0.0$ | $0.07 \pm 0.0$ | $0.24 \pm 0.0$ | $0.20 \pm 0.0$ |
| Piecewise | $0.39 \pm 0.0$ | $0.34 \pm 0.0$ | $0.33 \pm 0.0$ | $0.20 \pm 0.0$ | $0.19 \pm 0.0$ | $0.38 \pm 0.0$ | $0.29 \pm 0.0$ | $0.40 \pm 0.1$ | $0.40 \pm 0.1$ |

Table 11: **Comparison of various regression methods in approximating various functions when $p = 2, \sigma = 0.1, (\mathbf{x}, \mathrm{y}) \in \mathbb{R}$.**

Recall that Fig. 2 showed the regression performance of LIFT and existing learning algorithms when $n = 1000$. Figs. 8, 9 and 10 show the results of the same experiments for smaller training samples: when $n = 200, 300, 500$, respectively. Figs. 12, 13 and 14 visualize the extrapolation performance of baselines and LIFT for $n = 200, 300, 500$, respectively.

### C.1.5 Sample complexity

Fig. 20 and Fig. 21 provide the complete and clear version of sample complexity comparisons between evaluated methods in the regression and classification settings.

### C.1.6 Robustness to class-imbalance

We also investigate LIFT on class-imbalanced binary classification (OpenML datasets having IDs 1444, 1467, and 1511). Though evaluated datasets all have high class-imbalance values (ratio of the majority-class-size and the minority-class-size), we find that LIFT can perform well, achieving high F1, precision, and recall scores across the tasks. For instance, on the OpenML-1511 dataset (class-imbalance value is nearly 8), ODC gets 0 scores for both precision and recall while LIFT/GPT-J achieves the best recall ($82.61 \pm 7.10$) and F1 scores ($84.43 \pm 1.43$) along with XGBoost enjoying the highest precision.

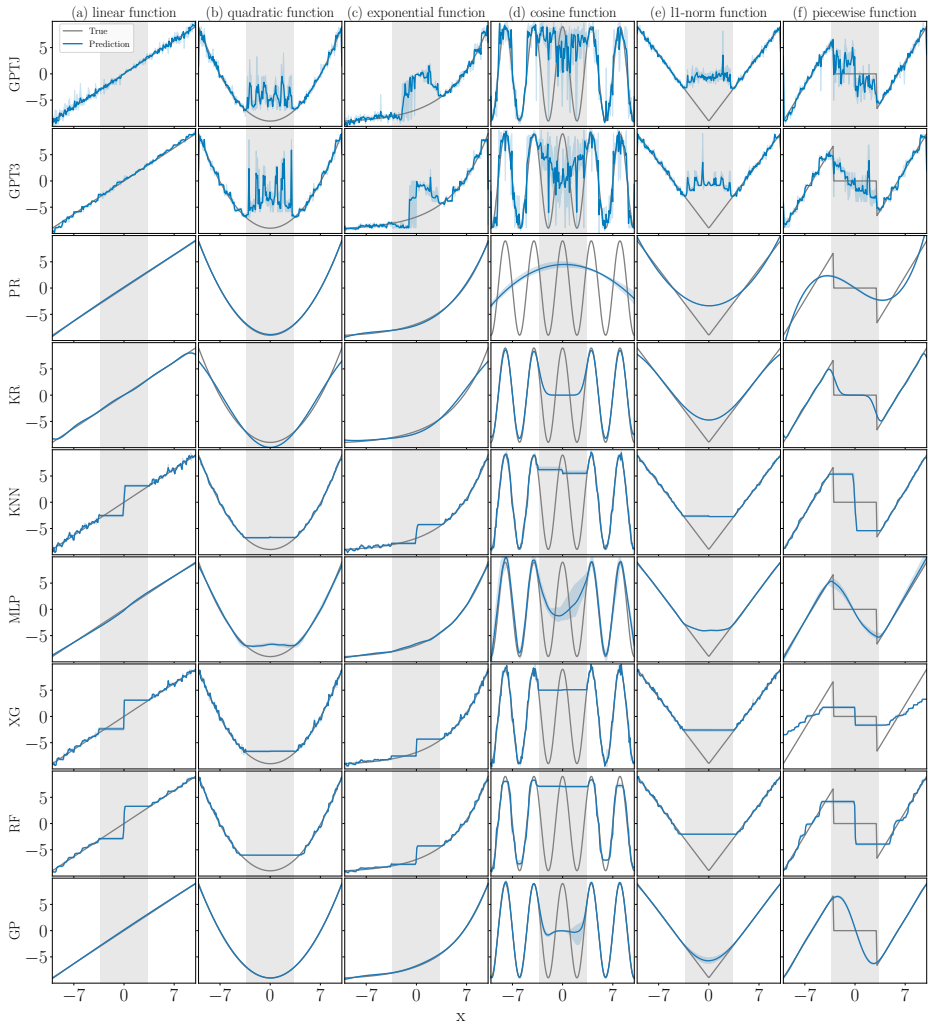



Figure 18: **Interpolation performance on synthetic regression tasks.** Each algorithm is trained with samples in the white background region ($3 \leq |x| \leq 10$), and tested on the interpolation area $|x| \leq 3$. GPT is having worse interpolation performance compared with existing methods.

| RAE | PR | KR | KNN | MLP | GBT | RF | GP | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|
| Linear | $0.71 \pm 0.0$ | $1.02 \pm 0.0$ | $0.78 \pm 0.0$ | $1.85 \pm 0.1$ | $0.97 \pm 0.0$ | $0.87 \pm 0.0$ | $0.13 \pm 0.0$ | $1.18 \pm 0.2$ |
| Quadratic | $1.12 \pm 0.0$ | $5.19 \pm 0.0$ | $1.33 \pm 0.0$ | $2.28 \pm 0.0$ | $0.98 \pm 0.0$ | $0.96 \pm 0.0$ | $0.69 \pm 0.0$ | $0.99 \pm 0.2$ |
| Exponential | $0.94 \pm 0.0$ | $10.23 \pm 0.0$ | $1.04 \pm 0.0$ | $3.18 \pm 0.2$ | $1.05 \pm 0.0$ | $0.96 \pm 0.0$ | $0.53 \pm 0.0$ | $1.15 \pm 0.0$ |
| Cosine | $1.01 \pm 0.0$ | $1.01 \pm 0.0$ | $1.00 \pm 0.0$ | $1.59 \pm 0.0$ | $1.00 \pm 0.0$ | $0.99 \pm 0.0$ | $1.01 \pm 0.0$ | $1.25 \pm 0.1$ |
| Linorm | $1.09 \pm 0.0$ | $1.00 \pm 0.0$ | $1.28 \pm 0.0$ | $1.97 \pm 0.1$ | $0.98 \pm 0.0$ | $0.94 \pm 0.0$ | $0.96 \pm 0.0$ | $1.12 \pm 0.1$ |
| Piecewise | $0.93 \pm 0.0$ | $1.00 \pm 0.0$ | $0.97 \pm 0.0$ | $2.11 \pm 0.0$ | $1.00 \pm 0.0$ | $0.94 \pm 0.0$ | $0.93 \pm 0.0$ | $1.35 \pm 0.1$ |

Table 12: **Comparison of various regression methods in approximating various functions when $p = 50, \sigma = 0.1, (\mathbf{x}, \mathbf{y}) \in \mathbb{R}$.**

Table 14 reports the performance of LIFT on three imbalanced datasets. Here, we evaluate the performance with additional measures: F1, precision, and recall, which are better used for imbalanced data than normal accuracy. Higher scores indicate better performance. Here, LIFT/GPT perform comparably well. For instance, on data OpenML-1444, LIFT/GPT-J achieves 35.83 in terms of F1, achieving the best score among evaluated algorithms. These results partially demonstrate the robustness of LIFT/GPT against the class imbalance.

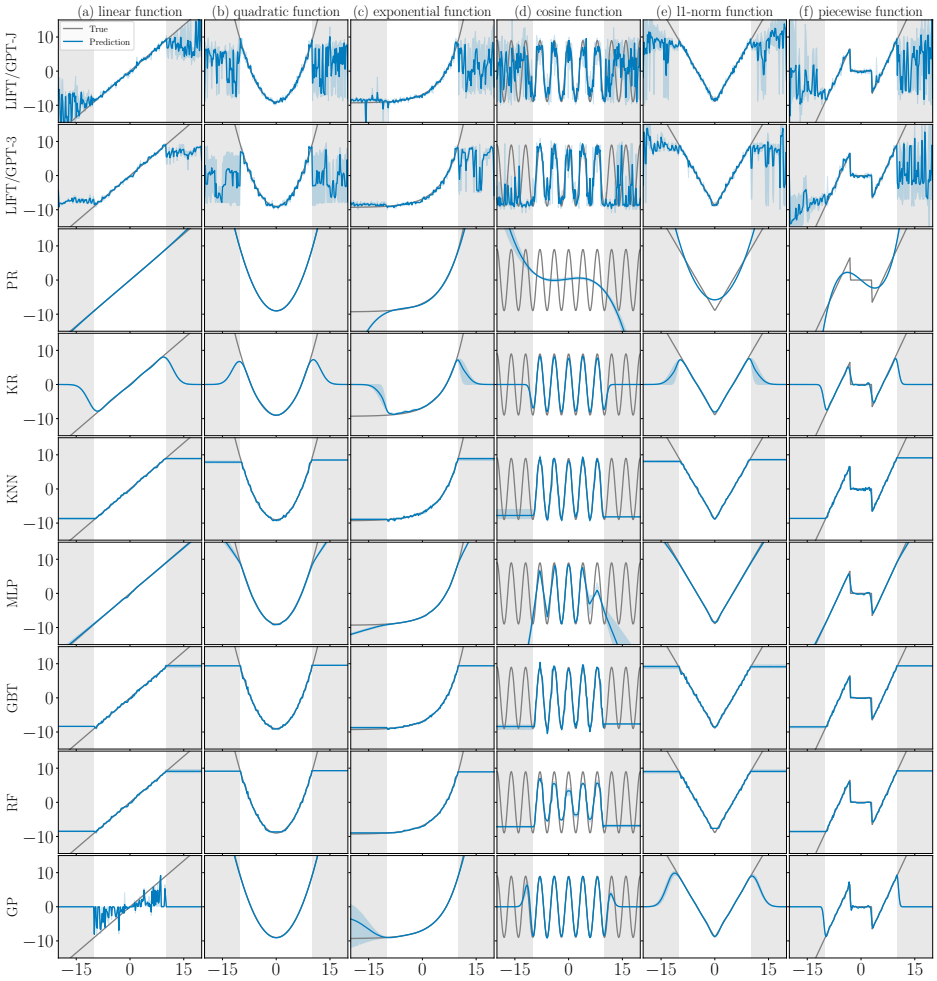



Figure 19: **Comparison of the extrapolation performance of LIFT and various baselines on synthetic regression tasks of approximating six functions $f$.** Each algorithm is trained by 1000 samples $(x, y)$ where the input $x$ is drawn from interval $[-10, 10]$ and the output is defined as $y = f(x)$. We test the how each algorithm perform regression for $x \notin [-10, 10]$.

| RAE | PR | KR | KNN | MLP | GBT | RF | GP | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|
| Linear | $0.95 \pm 0.0$ | $1.02 \pm 0.0$ | $0.88 \pm 0.0$ | $3.02 \pm 0.0$ | $0.99 \pm 0.0$ | $0.94 \pm 0.0$ | $0.64 \pm 0.0$ | $2.14 \pm 0.5$ |
| Quadratic | $1.02 \pm 0.0$ | $7.30 \pm 0.0$ | $1.29 \pm 0.0$ | $2.89 \pm 0.0$ | $1.01 \pm 0.0$ | $0.98 \pm 0.0$ | $0.89 \pm 0.0$ | $1.06 \pm 0.1$ |
| Exponential | $0.96 \pm 0.0$ | $14.12 \pm 0.0$ | $1.03 \pm 0.0$ | $4.14 \pm 0.0$ | $0.97 \pm 0.0$ | $0.93 \pm 0.0$ | $0.79 \pm 0.0$ | $1.03 \pm 0.0$ |
| Cosine | $1.02 \pm 0.0$ | $1.00 \pm 0.0$ | $1.09 \pm 0.0$ | $2.43 \pm 0.1$ | $1.04 \pm 0.0$ | $1.06 \pm 0.0$ | $1.00 \pm 0.0$ | $1.20 \pm 0.3$ |
| L1norm | $1.01 \pm 0.0$ | $1.01 \pm 0.0$ | $1.22 \pm 0.0$ | $2.80 \pm 0.1$ | $1.03 \pm 0.0$ | $1.01 \pm 0.0$ | $0.99 \pm 0.0$ | $1.27 \pm 0.2$ |
| Piecewise | $1.01 \pm 0.0$ | $1.00 \pm 0.0$ | $1.08 \pm 0.0$ | $4.20 \pm 0.1$ | $1.02 \pm 0.0$ | $1.01 \pm 0.0$ | $1.01 \pm 0.0$ | $1.11 \pm 0.0$ |

Table 13: **Comparison of various regression methods in approximating functions when $p = 120, \sigma = 0.1, (\mathbf{x}, \mathrm{y}) \in \mathbb{R}$.**

### C.1.7 Robustness to outliers

Fig. 23 and Fig. 24 visualize the full experiment results of robustness to outliers in Sec. 4.4.

### C.1.8 Robustness to label corruption

Table 15 and Table 16 extend the results reported in Fig. 22(b). These additional datasets follow a similar trend to what was discussed in Sec. 4.4. Namely, LIFT/GPT-3 performs somewhat in the middle of the baselines while LIFT/GPT-J underperforms.

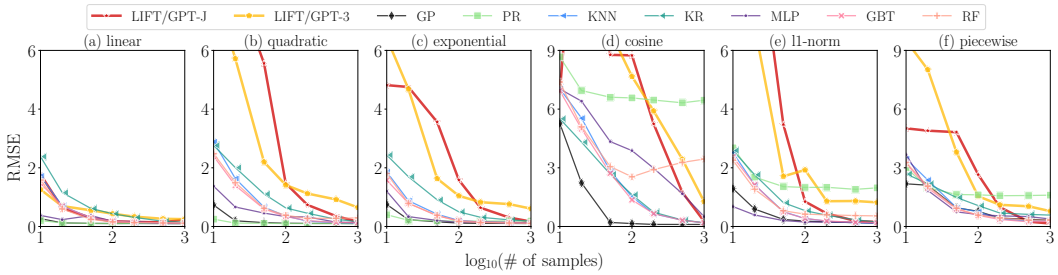



Figure 20: **Sample complexity of baselines and GPT (finetuned by LIFT) on regression problems of approximating various functions.** GPT needs slightly larger sample complexity to achieve similar performance. One can notice GPT achieves very competitive or even better performance when around 1000s of samples are given, especially for discontinuous function (piecewise).

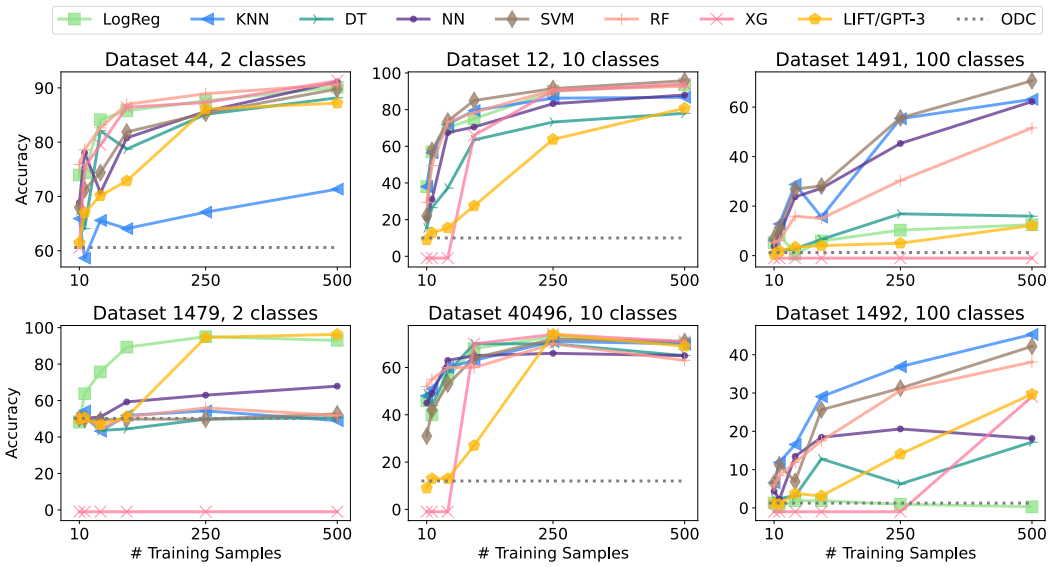



Figure 21: **Sample complexity on classification.** LIFT/GPT-3 achieves comparably good sample complexity among methods. When the number of classes increases, LIFT needs more samples to start adapting to the task.

| DID | | ODC | LogReg | KNN | DT | MLP | RBF-SVM | RF | XG | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1444 | Accuracy | **88.04** | 86.92±0.23 | 87.56±0.68 | 87.24±0.60 | 86.28±1.37 | **88.04±0.00** | **88.04±1.04** | **88.04±0.68** | 83.89±0.45 | 85.17±1.35 |
| | F1 | 0.00 | 10.77±2.74 | 16.84±5.26 | 9.01±12.74 | 11.77±3.43 | 0.00±0.00 | 15.79±6.45 | 35.50±3.68 | **35.83±3.61** | 24.52±1.78 |
| | Precision | 0.00 | 28.97±3.41 | 42.86±10.10 | 13.89±19.64 | 32.69±12.26 | 0.00±0.00 | 51.67±23.21 | **52.21±7.28** | 38.84±5.24 | 32.41±6.55 |
| | Recall | 0.00 | 6.67±1.89 | 10.67±3.77 | 6.67±9.43 | 8.00±3.27 | 0.00±0.00 | 9.33±3.77 | 28.00±5.66 | **33.33±2.36** | 20.00±0.00 |
| 1467 | Accuracy | **91.67** | 88.89±0.76 | 90.74±0.76 | 88.89±2.27 | **91.67±0.00** | 87.96±0.00 | 91.36±0.44 | 89.51±0.87 | 87.04±2.27 | **91.67±0.00** |
| | F1 | **95.65** | 94.00±0.43 | 95.13±0.40 | 94.04±1.27 | **95.65±0.00** | 93.47±0.00 | 95.48±0.24 | 94.37±0.48 | 94.51±1.08 | **95.65±1.00** |
| | Precision | 91.67 | **93.07±0.06** | 91.85±0.43 | 92.26±1.20 | 91.67±0.00 | 93.00±0.00 | 91.64±0.04 | 92.83±0.43 | 92.10±0.36 | 91.67±0.00 |
| | Recall | **100.00** | 94.95±0.82 | 98.65±0.48 | 95.96±2.86 | **100.00±0.00** | 93.94±0.00 | 99.66±0.48 | 95.96±0.82 | 97.08±2.57 | **100.00±0.00** |
| 1511 | Accuracy | 68.18 | 87.12±0.54 | **88.64±0.00** | 85.98±0.53 | 86.36±1.86 | 86.36±0.00 | 85.23±0.00 | 85.23±0.00 | 85.23±1.61 | 84.85±1.42 |
| | F1 | 0.00 | 79.76±0.89 | 80.51±0.36 | 78.60±1.00 | 77.80±2.79 | 78.82±0.35 | 76.64±0.39 | 76.91±0.39 | **84.43±1.43** | 75.28±2.60 |
| | Precision | 0.00 | 79.79±1.23 | **88.64±1.61** | 76.40±0.38 | 81.00±4.65 | 77.94±0.90 | 77.14±0.91 | 76.50±0.91 | 87.57±7.11 | 78.18±1.97 |
| | Recall | 0.00 | 79.76±1.68 | 73.81±1.68 | 80.95±1.68 | 75.00±2.91 | 79.76±1.68 | 76.19±1.68 | 77.38±1.68 | **82.61±7.10** | 72.62±3.37 |

Table 14: **Comparing classification Accuracy, F1, Precision, and Recall (%) on OpenML Imbalanced Datasets (1444, 1467, 1511).** All datasets are for binary classification and highly imbalanced. Here, higher values of the ODC accuracies show the higher imbalance of the data (50% shows the perfect balance). We observe that LIFT/GPT-J also perform well on these datasets, shown in high F1 scores which better represent the performance on these imbalanced data than the normal accuracy.

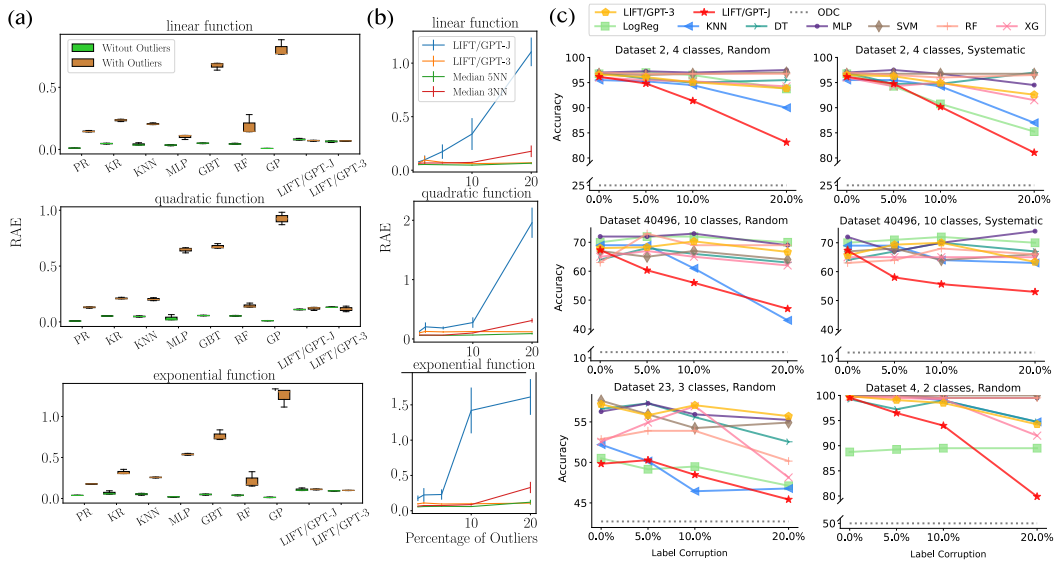



Figure 22: **Robustness of LIFT and baselines against outliers and label corruption.** (a) We inject 2% outliers in the dataset for three regression tasks. Both GPT-3 and GPT-J, finetuned by LIFT are highly robust against outliers, compared with baselines. (b) When the portion of outliers increases up to 20%, LIFT/GPT-3 is comparable to a strong baseline (median KNN), while LIFT/GPT-J fails. See Appendix for the comparison of all tested regression tasks. (c) Comparison of LIFT and baselines trained on corrupted synthetic data (ID 2 & 4) and real OpenML (ID 23 & 40496) datasets. We simulate *random errors* and *systematic errors*. While LIFT/GPT-3 shows robustness across the data, LIFT/GPT-J is more sensitive to the corrupted labels, similar to KNN.

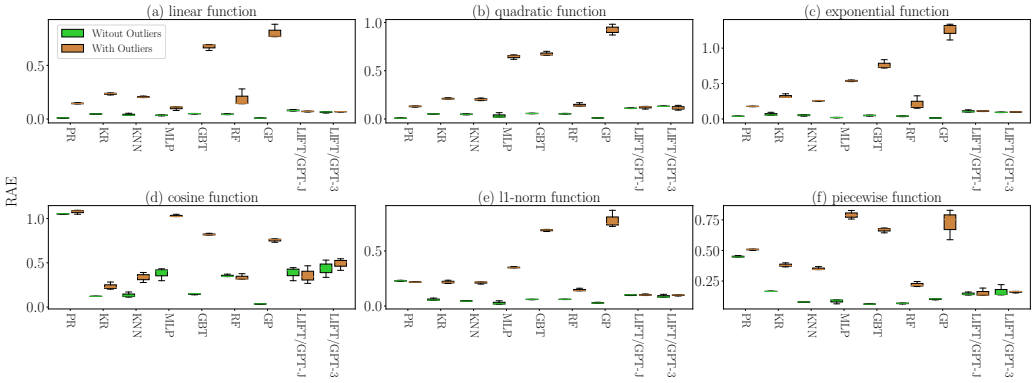



Figure 23: **Robustness against outliers.** We show how the regression error changes by injecting 2% outliers in the dataset for three regression tasks. Both GPT-3 and GPT-J, finetuned by LIFT are highly robust against outliers, compared with baselines.

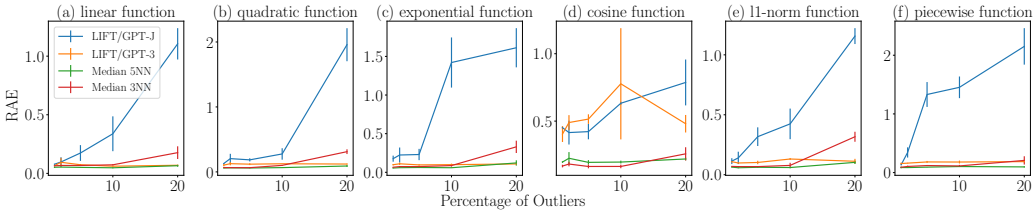



Figure 24: **Robustness of LIFT against outliers**. We show the regression error of LIFT when the portion of outliers increases up to 20%. LIFT/GPT-3 is comparable to a strong baseline (median NN), while LIFT/GPT-J fails when the portion increases.

| Dataset ID | Corruption | ODC | LogReg | KNN | DT | MLP | SVM | RF | XG | LIFT/GPT-3 | LIFT/GPT-J |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0% | 25.00 | 96.75 | 95.50 | 97.00 | 97.00 | 96.75 | 97.00 | 96.00 | 96.58 | 96.17 |
| | 5% | 25.00 | 97.00 | 95.25 | 95.75 | 97.25 | 96.50 | 96.75 | 95.50 | 96.08 | 94.83 |
| | 10% | 25.00 | 96.50 | 94.50 | 95.00 | 97.00 | 96.75 | 96.75 | 95.25 | 95.08 | 91.38 |
| | 20% | 25.00 | 93.75 | 90.00 | 95.50 | 97.50 | 97.00 | 96.75 | 94.25 | 93.83 | 83.12 |
| 4 | 0% | 50.00 | 88.75 | 100.00 | 99.25 | 99.75 | 100.00 | 99.75 | 99.75 | 99.83 | 99.58 |
| | 5% | 50.00 | 89.25 | 100.00 | 97.25 | 100.00 | 100.00 | 99.75 | 99.50 | 99.08 | 96.50 |
| | 10% | 50.00 | 89.50 | 99.00 | 99.00 | 99.50 | 100.00 | 99.50 | 99.25 | 98.50 | 94.00 |
| | 20% | 50.00 | 89.50 | 94.75 | 94.75 | 99.50 | 100.00 | 99.50 | 92.00 | 94.25 | 79.88 |
| 23 | 0% | 42.71 | 50.51 | 52.20 | 56.61 | 56.27 | 57.63 | 52.88 | 52.54 | 57.18 | 49.83 |
| | 5% | 42.71 | 49.15 | 50.17 | 57.29 | 57.29 | 55.93 | 53.90 | 54.92 | 55.82 | 50.28 |
| | 10% | 42.71 | 49.49 | 46.44 | 55.59 | 55.93 | 54.24 | 53.90 | 56.95 | 57.06 | 48.47 |
| | 20% | 42.71 | 47.12 | 46.78 | 52.54 | 55.25 | 54.92 | 50.17 | 48.14 | 55.71 | 45.42 |
| 48 | 0% | 35.48 | 51.61 | 61.29 | 67.74 | 58.06 | 61.29 | 77.42 | 64.52 | 50.54 | 61.29 |
| | 5% | 35.48 | 54.84 | 61.29 | 67.74 | 45.16 | 67.74 | 64.52 | 74.19 | 45.16 | 53.76 |
| | 10% | 35.48 | 41.94 | 45.16 | 54.84 | 32.26 | 32.26 | 51.61 | 54.84 | 52.69 | 46.24 |
| | 20% | 35.48 | 29.03 | 48.39 | 48.39 | 32.26 | 45.16 | 48.39 | 45.16 | 47.31 | 35.48 |
| 871 | 0% | 50.00 | 49.09 | 46.88 | 48.96 | 49.22 | 51.56 | 45.97 | 48.31 | 49.57 | 50.39 |
| | 5% | 50.00 | 51.43 | 48.18 | 49.22 | 50.26 | 49.74 | 48.44 | 46.62 | 50.65 | 48.61 |
| | 10% | 50.00 | 48.70 | 48.70 | 47.27 | 50.00 | 46.49 | 51.17 | 47.01 | 48.96 | 48.66 |
| | 20% | 50.00 | 50.39 | 49.22 | 50.52 | 47.01 | 50.52 | 47.14 | 49.61 | 50.74 | 50.82 |
| 1467 | 0% | 91.67 | 89.81 | 89.81 | 91.67 | 91.67 | 87.96 | 91.67 | 90.74 | 91.67 | 87.04 |
| | 5% | 91.67 | 89.81 | 91.67 | 87.04 | 91.67 | 91.67 | 90.74 | 87.96 | 91.36 | 85.49 |
| | 10% | 91.67 | 90.74 | 88.89 | 90.74 | 88.89 | 91.67 | 91.67 | 88.89 | 91.67 | 83.80 |
| | 20% | 91.67 | 90.74 | 81.48 | 83.33 | 88.89 | 91.67 | 89.81 | 87.04 | 91.67 | 76.39 |
| 40496 | 0% | 12.00 | 70.00 | 69.00 | 64.00 | 72.00 | 67.00 | 63.00 | 65.00 | 68.00 | 67.33 |
| | 5% | 12.00 | 72.00 | 69.00 | 68.00 | 72.00 | 65.00 | 73.00 | 67.00 | 68.33 | 60.33 |
| | 10% | 12.00 | 72.00 | 61.00 | 66.00 | 73.00 | 67.00 | 69.00 | 65.00 | 70.33 | 56.00 |
| | 20% | 12.00 | 70.00 | 43.00 | 63.00 | 69.00 | 64.00 | 69.00 | 62.00 | 66.67 | 47.00 |

Table 15: **Comparison of various methods fitted to *randomly* corrupted classification labels.**w label in the label space with equal probability.

| Dataset ID | Corruption | ODC | LogReg | KNN | DT | MLP | SVM | RF | XG | LIFT/GPT-3 | LIFT/GPT-J |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0% | 25.00 | 96.75 | 95.50 | 97.00 | 97.00 | 96.75 | 97.00 | 96.00 | 96.50 | 96.17 |
| | 5% | 25.00 | 94.25 | 95.50 | 94.75 | 97.50 | 96.75 | 96.50 | 94.25 | 96.25 | 94.75 |
| | 10% | 25.00 | 90.75 | 94.25 | 94.75 | 96.75 | 96.75 | 96.00 | 95.00 | 94.92 | 90.17 |
| | 20% | 25.00 | 85.25 | 87.00 | 97.00 | 94.50 | 96.75 | 96.50 | 91.50 | 92.58 | 81.07 |
| 40496 | 0% | 12.00 | 70.00 | 69.00 | 64.00 | 72.00 | 67.00 | 63.00 | 65.00 | 65.67 | 67.33 |
| | 5% | 12.00 | 71.00 | 69.00 | 67.00 | 67.00 | 68.00 | 64.00 | 65.00 | 69.33 | 58.00 |
| | 10% | 12.00 | 72.00 | 64.00 | 70.00 | 70.00 | 64.00 | 68.00 | 65.00 | 70.00 | 55.67 |
| | 20% | 12.00 | 70.00 | 63.00 | 67.00 | 74.00 | 66.00 | 66.00 | 65.00 | 63.33 | 53.00 |

Table 16: **Comparison of various methods fitted to *systematically* corrupted classification labels.** In the systematic error regime, we corrupt a label by assigning all corrupted labels of one class to a single other label.

### C.2 How do different language models affect LIFT?

Together with GPT-J and GPT-3 (Ada version), we now evaluate tasks on larger pretrained LMs to verify whether LIFT benefits more from larger LMs. We consider three bigger versions of GPT-3, namely Baggage, Curie, and Davinci (in the ascending order of the number of parameters). We compare all models on several regression tasks in Table 18 and classification tasks in Table **??**. We find that larger LMs bring some improvements when combined with LIFT. However, these improvements are not quite significant compared to smaller LMs. This may be because the downstream task is small with respect to all evaluated LMs, and finetuning is necessary for LMs to adapt to these new modalities. Furthermore, we observe the consistent improvements of in-context classification with larger models across classification tasks. We conjecture that larger LMs probably contain more embedded knowledge, useful for classification when training data is small.

| Function | GPT-J | Ada | Babbage | Curie | Davinci |
|---|---|---|---|---|---|
| Linear | 0.08±0.01 | 0.06±0.01 | 0.06±0.00 | 0.06±0.01 | 0.06±0.00 |
| Quadratic | 0.11±0.00 | 0.13±0.00 | 0.11±0.02 | 0.10±0.01 | 0.09±0.00 |
| Exponential | 0.11±0.02 | 0.09±0.00 | 0.09±0.01 | 0.08±0.00 | 0.08±0.00 |
| Cosine | 0.38±0.08 | 0.44±0.10 | 0.41±0.06 | 0.38±0.01 | 0.38±0.05 |
| L1-norm | 0.10±0.00 | 0.09±0.01 | 0.10±0.01 | 0.08±0.01 | 0.09±0.01 |
| Piecewise | 0.15±0.01 | 0.17±0.05 | 0.15±0.02 | 0.15±0.01 | 0.14±0.01 |

Table 18: **Comparison of LIFT on different LMs across regression tasks.** The scores are regression errors. In general, Davinci model performs the best, but the gaps to other models are not always significant.

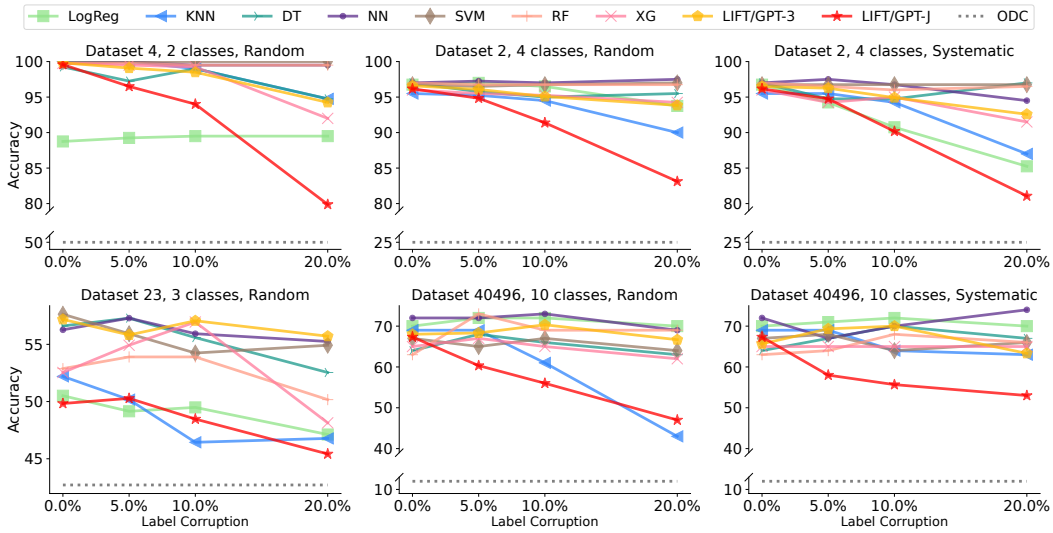



Figure 25: **Robustness against label corruption.** We train models on corrupted versions of synthetic data (ID 2 & 4) and real OpenML (ID 23 & 40496) datasets. We simulate *random errors* and *systematic errors*. While LIFT/GPT-3 shows robustness across the data, LIFT/GPT-J is more sensitive to the corruption, similar to KNN.

Table 17: **Comparisons of LIFT on basic classification tasks, classification tasks with feature names, and in-context classification (note that columns 48(50) etc. indicate DID(#Prompts))**. In general, larger LIFT/GPT-3 models (babbage, curie, davinci) perform better than LIFT/GPT-3 ada and LIFT/GPT-J. However, performance on different datasets is not always consistent with model size. Also observed that model size has large effects on in-context classification vs. other tasks. To see the complete version of comparisons, see Table **??** at appendix.

| Tasks | Basic Classification Accuracy | | Classfication w/ Feat. Names | | In-context Classifction | |
|---|---|---|---|---|---|---|
| DID(#Prompts) | 311 | 1493 | 48 | 54 | 40496(32) | 1511(28) |
| **LIFT/GPT-J** | 93.97±1.00 | 50.32±2.18 | 58.06±13.94 | 69.02±3.67 | 10.00±0.82 | 56.06±17.14 |
| **Ada** | 95.39±0.67 | 55.73±3.19 | 43.01±1.52 | 72.16±2.00 | 15.33±3.86 | 52.65±12.11 |
| **Babbage** | 96.81±0.07 | 62.19±1.80 | 47.31±8.05 | 69.22±1.11 | 11.00±0.00 | 53.79±12.07 |
| **Curie** | 95.21±0.06 | 62.50±0.97 | 49.46±6.08 | 70.66±2.28 | 20.67±4.78 | 67.80±0.53 |
| **Davinci** | 96.81±0.41 | 57.19±0.70 | 49.46±9.97 | 68.16±1.69 | 20.67±6.60 | 68.94±0.54 |

In addition, we perform comparison with baseline models with larger-capacities. In particular, we provide the results of deep neural network models based on architectures specifically designed for tabular data: TabNet and TabTransformer.

Table **??** presents the full version of Table 18 in the main paper in comparing the effects of different pretrained language models to LIFT via classification tasks. We observe that large models gain more effectiveness only in the in-context classification when no finetuning (LIFT) is involved. Otherwise, the gain is not significant to smaller models in the presence of LIFT.

## C.3 Results for LIFT-specific learning properties (Section 5)

### C.3.1 Does LIFT benefit from incorporating feature names?

Table 3 and Table 21 present the full experiment results of finetuning with feature names in Sec. 5.2.

## C.4 Results for improving techniques of LIFT (Section 6)

### C.4.1 Two-stage intermediate finetuning for LIFT

We provide an additional evaluation of the two-stage intermediate finetuning with LIFTGPT-J on four more datasets. For any given dataset, we first generate two pretext tasks with simple synthetic

Table 19: **Performance of LIFT with different LMs.** To better answer this question, we compare our results with two groups of methods: (G1) Apply LIFT on LMs using models having being trained on non-human-language data, including GPT-J being fine-tuned on gibberish data (Gibberish-GPT-J), and models being pretrained on source codes (CodeGen and CodeParrot), and (G2) Apply LIFT on a randomly initialized GPT-J (not pretrained).

| Dataset (DID) | ODC | LIFT/GPT-3 | LIFT/GPT-J | LIFT/Gibberish | LIFT/CodeGen | LIFT/CodeParrot |
|---|---|---|---|---|---|---|
| Blobs (2) | 25.00 | 96.67± 0.24 | 96.17± 0.59 | 96.42± 0.24 | 93.67± 0.72 | 93.39± 1.82 |
| Two Circles (6) | 50.00 | 81.42± 0.82 | 75.92± 1.65 | 68.67± 1.50 | 53.02± 0.66 | 50.08± 2.47 |
| Iris (61) | 33.33 | 97.0± 0.00 | 96.67± 0.00 | 94.44± 1.57 | 43.31± 6.67 | 60.00± 8.82 |
| Customers (1511) | 68.18 | 84.85± 1.42 | 85.23± 1.61 | 67.43± 1.42 | 45.96± 8.96 | 43.11± 3.34 |
| Wine (187) | 38.89 | 92.59± 1.31 | 93.52± 1.31 | 84.26± 3.46 | 77.78± 0.00 | 33.88± 3.87 |
| LED (40496) | 11.0 | 69.33± 2.05 | 65.33± 0.47 | 72.67± 1.25 | 11.00± 4.00 | 23.46± 13.85 |

| Dataset (DID) | ODC | LIFT/GPT-3 | LIFT/GPT-J | LIFT/Rand-GPTJ | TabNet | TabTransformer |
|---|---|---|---|---|---|---|
| Blobs (2) | 25.00 | 96.67± 0.24 | 96.17± 0.59 | 25.65± 1.58 | 96.75± 0.00 | 50.00± 0.00 |
| Two Circles (6) | 50.00 | 81.42± 0.82 | 75.92± 1.65 | 49.88± 5.01 | 74.25± 12.39 | 49.25± 1.29 |
| Iris (61) | 33.33 | 97.0± 0.00 | 96.67± 0.00 | 27.78± 20.79 | 97.78± 1.92 | 72.22± 5.09 |
| Customers (1511) | 68.18 | 84.85± 1.42 | 85.23± 1.61 | 52.47± 7.15 | 85.22± 3.93 | 87.12± 0.66 |
| Wine (187) | 38.89 | 92.59± 1.31 | 93.52± 1.31 | 22.22± 15.71 | 94.44± 5.56 | 90.74± 13.70 |
| LED (40496) | 11.0 | 69.33± 2.05 | 65.33± 0.47 | 11.68± 4.44 | 67.00± 2.46 | 41.00± 12.49 |

| Type | Dataset (ID) | $p/c$ | ODC | LogReg | KNN | DT | MLP | RBF-SVM | RF | XG | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Synthetic | circles (3) | 2 / 2 | 50.00 | 48.58±1.94 | 81.25±0.20 | 77.42±0.24 | 82.00±0.54 | **83.08±0.59** | 82.42±1.33 | 81.42±0.31 | 79.95±1.53 | 81.17±0.42 |
| | two circles (6) | 2 / 2 | 50.00 | 49.83±4.18 | **81.83±0.62** | 75.50±0.20 | 68.42±3.86 | 80.00±0.54 | 76.08±0.59 | 79.25±0.35 | 75.92±1.65 | 81.42±0.82 |
| | blobs (2) | 2 / 4 | 25.00 | 96.75±0.00 | 95.50±0.20 | 96.08±0.82 | 96.58±0.42 | 96.75±0.00 | **97.17±0.24** | 96.17±0.12 | 96.17±0.59 | 96.67±0.24 |
| | moons (4) | 2 / 4 | 50.00 | 88.58±0.12 | **100.00±0.00** | 99.25±0.41 | 98.75±1.08 | **100.00±0.00** | 99.75±0.00 | 99.83±0.12 | 99.58±0.42 | **100.00±0.00** |
| | 9Clusters (1) | 2 / 9 | 11.25 | 100.00±0.00 | 100.00±0.00 | 100.00±0.00 | 100.00±0.00 | 100.00±0.00 | **100.00±0.00** | **100.00±0.00** | 99.75±0.00 | **100.00±0.00** |
| Tabular (OpenML) | Customers (1511) | 9 / 2 | 68.18 | 87.12±0.54 | **88.64±0.00** | 85.98±0.53 | 86.36±1.86 | 86.36±0.00 | 85.23±0.00 | 85.23±0.00 | 85.23±1.61 | 84.85±1.42 |
| | Pollution (882) | 16 / 2 | 50.00 | 58.33±11.79 | 66.67±6.81 | **77.78±3.93** | 66.67±0.00 | 58.33±6.81 | **77.78±3.93** | 63.89±7.86 | 63.89±3.93 | 63.89±7.86 |
| | Spambase (44) | 58 / 2 | 60.59 | 93.27±0.00 | 90.77±0.00 | 90.7±0.14 | **94.35±0.00** | 93.70±0.00 | 95.01±0.00 | **95.87±0.00** | 94.03±0.54 | 94.90±0.36 |
| | Hill-Valley (1479) | 101 / 2 | 49.79 | 77.78±0.00 | 56.38±0.00 | 56.38±0.89 | 50.21±0.00 | 68.72±0.00 | 51.44±0.00 | 59.26±0.00 | **100.00±0.20** | **99.73±0.19** |
| | TAE (48) | 6 / 3 | 35.48 | 45.16±4.56 | 60.22±4.02 | 65.59±5.49 | 54.84±2.63 | 53.76±6.63 | **67.74±7.90** | 66.67±8.05 | 61.29±6.97 | 65.59±6.63 |
| | CMC (23) | 10 / 3 | 42.71 | 49.49±0.83 | 50.85±1.91 | 56.72±0.32 | 57.29±0.73 | 56.50±0.97 | 53.45±1.05 | 52.43±0.42 | 49.83±0.28 | **57.74±0.89** |
| | Wine (187) | 14 / 3 | 38.89 | **100.00±0.00** | 96.29±1.31 | 93.52±2.62 | 98.15±2.62 | **100.00±0.00** | **100.00±0.00** | 97.22±0.00 | 93.52±1.31 | 92.59±1.31 |
| | Vehicle (54) | 19 / 4 | 25.88 | 80.39±1.00 | 69.61±0.74 | 63.92±2.37 | 79.21±0.28 | **81.18±0.48** | 75.88±1.27 | 73.14±0.28 | 64.31±2.37 | 70.20±2.73 |
| | LED (40496) | 8 / 10 | 11.00 | 68.67±0.94 | 63.67±6.13 | 66.33±2.87 | **72.00±0.82** | 68.00±0.82 | 64.33±0.94 | 66.00±0.82 | 65.33±0.47 | 69.33±2.05 |
| | OPT (28) | 65 / 10 | 10.14 | 96.53±0.22 | 96.92±0.16 | 89.8±1.09 | 97.36±0.27 | 97.95±0.00 | 97.69±0.14 | 97.48±0.17 | 98.22±0.11 | **98.99±0.30** |
| | Mfeat (12) | 217 / 10 | 10.00 | 97.67±0.12 | 97.67±0.31 | 87.67±1.05 | 96.5±0.35 | **98.83±0.24** | 97.75±0.35 | 96.75±0.00 | 94.17±1.75 | 93.08±0.24 |
| | Margin (1491) | 65 / 100 | 0.94 | 81.35±0.15 | 77.60±0.97 | 43.86±1.21 | 77.71±1.91 | **81.98±0.30** | 77.71±1.98 | 70.21±0.29 | 50.23±1.33 | 59.37±0.92 |
| | Texture (1493) | 65 / 100 | 0.94 | 81.67±0.97 | 80.62±0.76 | 46.88±1.93 | 76.88±2.44 | **83.44±0.89** | 73.12±0.76 | 70.73±1.41 | 50.32±2.18 | 67.50±1.42 |
| Images | MNIST | 784 / 10 | 11.35 | 91.95±0.69 | 96.71±0.11 | 87.42±0.64 | 97.30±0.16 | 97.70±0.97 | 94.91±0.18 | 97.69±0.04 | 97.01±1.15 | **98.15±0.67** |
| | P-MNIST | | 11.35 | 92.58±0.04 | 96.74±0.08 | 87.87±0.69 | 97.39±0.14 | **98.06±0.31** | 94.59±0.18 | 97.62±0.09 | 95.80± 0.07 | 96.25±0.35 |
| | FMNIST | | 10.00 | 85.59±0.09 | 85.59±0.03 | 80.52±0.40 | 88.86±0.02 | **90.59±0.02** | 85.25±0.13 | 90.19±0.04 | 85.10 ± 0.19 | 90.18 ±0.12 |
| | P-FMNIST | | 10.00 | 84.95±0.84 | 85.15±0.61 | 79.91±0.93 | 88.86±0.61 | 88.04±1.69 | 84.93±0.59 | **89.93±0.14** | 82.25±0.27 | 88.92±0.71 |

Table 20: **Accuracies (↑) on various classification datasets.** We evaluate LIFT/GPTs on different classification datasets: 2D synthetic datasets, tabular datasets in OpenML [62] , and image datasets, varying number of features ($p$) and number of classes ($c$). Overall, LIFT/GPTs perform comparably well across all tasks. LIFT/GPTs can be adapted well to non-linear datasets (circles, two circles), beyond the capacity of logistic regression. On the OpenML data, LIFT/GPTs achieves competitive performances with the best methods, such as XGBoost or RBF-SVM. The performance of LIFT/GPTs degrades as the number of classes is large, *e.g.*, when the number of classes $c$=100. On the vision data, LIFT/GPTs perform relatively well, achieving highly competitive accuracies on both MNIST and Fashion MNIST. We note that the classes of MNIST are not fully balanced. Thus ODC gets 11.35% instead of 10% as ODC returns the optimal class learned from the training dataset.

Gaussian samples (discussed in A). We finetune the GPT with pretext tasks for a few (2–3) epochs, then finetune the newly finetuned GPT with the target (given) dataset. Here, due to the black-box API of GPT-3, we currently can neither keep the order of samples unchanged (pretext, target) during the finetuning stage nor finetune the model twice. Hence, we only use GPT-J in this experiment.

Fig. 26 presents results from eight datasets, including a regression task, three OpenML tasks, and other synthetic datasets. We can see that two-stage intermediate LIFT helps improve the original finetuning, especially when the number of training samples is small. The effect of two-stage finetuning is more clearly shown in synthetic classification datasets (Blobs, Circles, Moon, and Two Circles). We hypothesize that GPT requires a few samples to start adapting to the new task, in which we can replace the actual samples with the synthetic data. We also observe that besides the number of features and number of classes, the pretexts do not need to represent any other characteristics of the target dataset, such as the linear/non-linear correlation or the relevance of features. This makes it simpler to generate the pretexts.

| Dataset | Key Features | Frac. of Samples | GPT-3 Without Feat. Names | GPT-3 With Feat. Names | GPT-J Without Feat. Names | GPT-J With Feat. Names |
|---|---|---|---|---|---|---|
| CCPP | Air Temperature, Ambient Pressure, Relative Humidity, Exhaust Vacuum | 0.2 | 0.26 ±0.00 | **0.25 ±0.00** | **0.33 ±0.02** | 0.37 ±0.04 |
| | | 0.4 | **0.24 ±0.00** | 0.24 ±0.00 | **0.36 ±0.01** | 0.36 ±0.03 |
| | | 0.6 | 0.23 ±0.02 | **0.22 ±0.00** | **0.32 ±0.00** | 0.33 ±0.00 |
| | | 0.8 | 0.23 ±0.00 | **0.23 ±0.01** | **0.27 ±0.00** | 0.31 ±0.06 |
| | | 1.0 | **0.18 ±0.01** | 0.21 ±0.00 | **0.24 ±0.01** | 0.31 ±0.00 |
| Servo | Motor, Screw, Voltage, Power Gain | 0.2 | 0.81 ±0.01 | **0.81 ±0.00** | **1.05 ±0.17** | 1.45 ±0.14 |
| | | 0.4 | 0.61 ±0.16 | **0.53 ±0.02** | **1.17 ±0.20** | 1.71 ±0.73 |
| | | 0.6 | **0.31 ±0.03** | 0.51 ±0.05 | 1.22 ±0.16 | **1.03 ±0.22** |
| | | 0.8 | 0.69 ±0.18 | **0.44 ±0.09** | **1.14 ±0.13** | 1.30 ±0.13 |
| | | 1.0 | **0.38 ±0.02** | 0.44 ±0.14 | **1.17 ±0.16** | 1.27 ±0.25 |
| Insurance | Age, Gender, Smoke/Not Smoke, Body Mass Index | 0.2 | **0.89 ±0.03** | 0.89 ±0.03 | 1.13 ±0.12 | **0.66 ±0.09** |
| | | 0.4 | 0.44 ±0.15 | **0.24 ±0.02** | **0.72 ±0.14** | 1.06 ±0.07 |
| | | 0.6 | 0.38 ±0.10 | **0.27 ±0.07** | 0.79 ±0.15 | **0.45 ±0.05** |
| | | 0.8 | 0.34 ±0.05 | **0.16 ±0.01** | 0.33 ±0.01 | **0.32 ±0.10** |
| | | 1.0 | 0.20 ±0.05 | **0.16 ±0.01** | 0.59 ±0.11 | **0.27 ±0.05** |
| Student | Freetime, Study Time, Absenses | 0.2 | 0.37 ±0.06 | **0.31 ±0.01** | 0.43 ±0.07 | **0.41 ±0.05** |
| | | 0.4 | **0.32 ±0.02** | 0.35 ±0.01 | **0.36 ±0.02** | 0.36 ±0.01 |
| | | 0.6 | 0.31 ±0.01 | **0.31 ±0.01** | 0.42 ±0.06 | **0.40 ±0.01** |
| | | 0.8 | **0.28 ±0.01** | 0.29 ±0.01 | 0.44 ±0.07 | **0.39 ±0.04** |
| | | 1.0 | **0.27 ±0.02** | 0.30 ±0.03 | **0.37 ±0.02** | 0.39 ±0.02 |

Table 21: **Performance of LIFT with and without feature names in prompts on regression tasks.** Note that common sense may be helpful for tasks dealing with the Insurance and Student datasets, but not for the Servo and CCPP datasets.

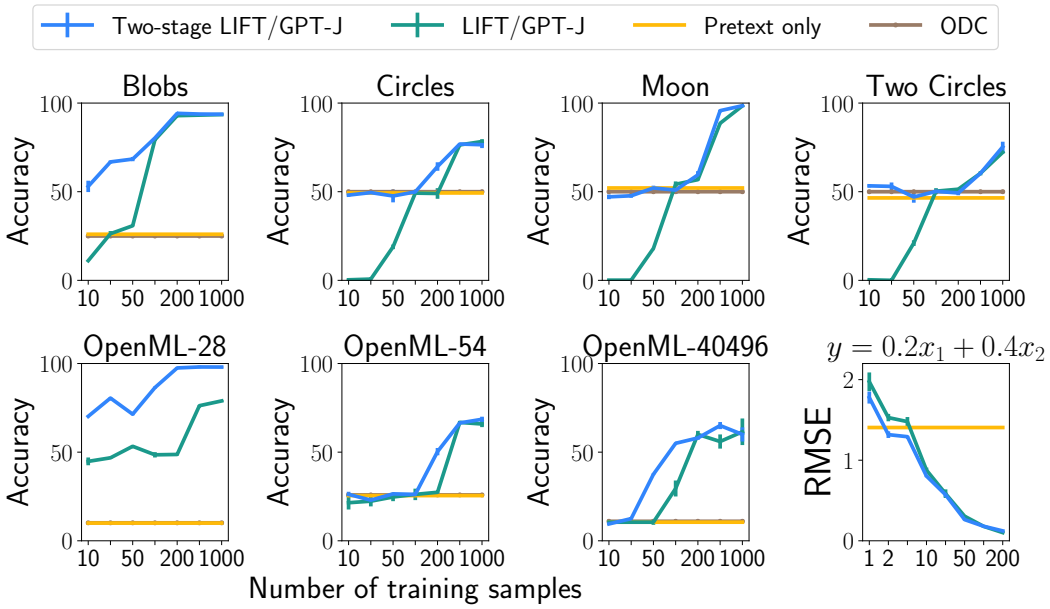



Figure 26: **Two-stage finetuning with LIFT/GPT-J**. We apply LIFT in two consecutive stages: first on synthetic pretext data, then on the target data. We evaluate on classification tasks (with accuracy) and a regression task (RMSE error). The inter-finetuning LIFT(blue) outperforms the original finetuning LIFT (green) when the number of training samples is small, across all tasks.

# D Additional Experiments and Findings (NOT Discussed in the Main Paper)

Here we provide results of additional experiments that have not been discussed in the main manuscript, including the investigation of the effect of numerical type (D.2), whether LIFT/GPT can perform ridge regression (D.3), and how contracting prompts affect LIFT/GPT (D.4).

## D.1 Robustness to Gaussian noises

Table 22 shows how much LIFT is robust against Gaussian noise in the input prompt for the MNIST classification problem. The input prompt is represented as a sequence of pixel values, where each

pixel is an integer in $\{0, 1, \cdots, 255\}$. For each pixel, we add Gaussian noise $z \sim \mathcal{N}(0, \sigma^2)$ and revise the input prompt accordingly. As shown in Table 22, LIFT/GPT-J can tolerate small noise, but when the standard deviation is $\sigma = 2.5$, the performance drastically degrades.

| Noise level ($\sigma$) | 0.0 | 0.5 | 2.5 | 5 |
|---|---|---|---|---|
| Accuracy (%) of LIFT/GPT-J | 96.88 | 95.29 | 68.79 | 53.82 |

Table 22: **Robustness of LIFT/GPT-J against additive Gaussian noise on the pixel values of MNIST**. Out of 255 values representing each black/white pixel, adding noise with standard deviation of $\sigma = 2.5$ significantly degrades the performance of LIFT/GPT-J.

### D.2 How do different numerical types affect LIFT/GPT?

| Function | Type | PR | KR | KNN | MLP | GBT | RF | GP | LIFT/GPT-J | LIFT/GPT-3 |
|---|---|---|---|---|---|---|---|---|---|---|
| Exponential | $\mathbb{R}$ | $0.04 \pm 0.0$ | $0.07 \pm 0.0$ | $0.05 \pm 0.0$ | $0.02 \pm 0.0$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.01 \pm 0.0$ | $0.11 \pm 0.0$ | $0.09 \pm 0.0$ |
| | $\mathbb{R}^+$ | $0.01 \pm 0.0$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.08 \pm 0.1$ | $0.05 \pm 0.0$ | $0.04 \pm 0.0$ | $0.01 \pm 0.0$ | $0.16 \pm 0.1$ | $0.10 \pm 0.0$ |
| | $\mathbb{Z}$ | $2.59 \pm 0.1$ | $0.46 \pm 0.1$ | $0.33 \pm 0.0$ | $0.57 \pm 0.0$ | $0.32 \pm 0.0$ | $0.24 \pm 0.1$ | $0.92 \pm 0.0$ | $1.43 \pm 0.1$ | $1.15 \pm 0.1$ |
| | $\mathbb{Z}^+$ | $2.69 \pm 0.0$ | $0.42 \pm 0.0$ | $0.35 \pm 0.1$ | $0.57 \pm 0.0$ | $0.28 \pm 0.0$ | $0.18 \pm 0.0$ | $0.94 \pm 0.0$ | $0.68 \pm 0.2$ | $2.05 \pm 0.8$ |
| Cosine | $\mathbb{R}$ | $1.05 \pm 0.0$ | $0.12 \pm 0.0$ | $0.14 \pm 0.0$ | $0.38 \pm 0.1$ | $0.15 \pm 0.0$ | $0.35 \pm 0.0$ | $0.04 \pm 0.0$ | $0.38 \pm 0.1$ | $0.44 \pm 0.1$ |
| | $\mathbb{R}^+$ | $0.91 \pm 0.0$ | $0.07 \pm 0.0$ | $0.07 \pm 0.0$ | $0.13 \pm 0.0$ | $0.10 \pm 0.0$ | $0.10 \pm 0.0$ | $0.02 \pm 0.0$ | $0.69 \pm 0.3$ | $0.40 \pm 0.2$ |
| | $\mathbb{Z}$ | $1.05 \pm 0.0$ | $0.79 \pm 0.0$ | $1.10 \pm 0.1$ | $1.03 \pm 0.0$ | $0.96 \pm 0.1$ | $1.05 \pm 0.0$ | $0.58 \pm 0.0$ | $0.65 \pm 0.6$ | $0.06 \pm 0.0$ |
| | $\mathbb{Z}^+$ | $1.05 \pm 0.0$ | $0.80 \pm 0.0$ | $1.08 \pm 0.1$ | $1.05 \pm 0.1$ | $0.91 \pm 0.1$ | $1.04 \pm 0.0$ | $0.57 \pm 0.0$ | $0.51 \pm 0.6$ | $0.07 \pm 0.0$ |

Table 23: **Comparing methods in approximating functions with different numerical type when $p = 1$.**

We also investigate the performance of GPT given different types of numerical values. First, how do different numerical types used in the input prompt (for representing the numerical feature values) change the performance of LIFT? Table 23 shows the regression error with different numerical types. No preference for numerical input types of LIFT is observed in Table 23.

### D.3 Can LIFT perform Ridge regression?

We investigate whether LIFT can perform Ridge regression here. Note that level encoding allows for ridge regression, whose objective is to minimize $\|y - Xw\|_2^2 + \lambda\|w\|_2^2$. We know that this is equivalent to $\|[y^T, 0]^T - [X^T, \sqrt{\lambda}I]^T w\|_2^2$. Therefore, we apply this level of encoding, and the results are reported in Table 24. We observe that LIFT fails to perform Ridge regression. Instead,

| $p$ | $\lambda$ | LIFT/GPT-J RAE LR | LIFT/GPT-J RAE Ridge | LIFT/GPT-3 RAE LR | LIFT/GPT-3 RAE Ridge |
|---|---|---|---|---|---|
| 1 | 0 | $0.000 \pm 0.000$ | $0.000 \pm 0.000$ | $0.915 \pm 0.000$ | $0.000 \pm 0.000$ |
| 1 | 10 | $0.000 \pm 0.000$ | $0.016 \pm 0.000$ | $0.915 \pm 0.000$ | $0.016 \pm 0.000$ |
| 1 | 50 | $0.000 \pm 0.000$ | $0.403 \pm 0.000$ | $0.915 \pm 0.000$ | $0.402 \pm 0.000$ |
| 1 | 100 | $0.000 \pm 0.000$ | $1.691 \pm 0.000$ | $0.915 \pm 0.000$ | $1.690 \pm 0.000$ |
| 1 | 1000 | $0.000 \pm 0.000$ | $170.406 \pm 0.000$ | $0.915 \pm 0.000$ | $170.612 \pm 0.000$ |
| 10 | 0 | $0.532 \pm 0.000$ | $0.532 \pm 0.000$ | $0.915 \pm 0.000$ | $0.521 \pm 0.000$ |
| 10 | 10 | $0.374 \pm 0.000$ | $0.369 \pm 0.000$ | $0.915 \pm 0.000$ | $0.504 \pm 0.000$ |
| 10 | 50 | $0.417 \pm 0.000$ | $0.523 \pm 0.000$ | $0.915 \pm 0.000$ | $0.563 \pm 0.000$ |
| 10 | 100 | $0.365 \pm 0.000$ | $1.307 \pm 0.000$ | $0.915 \pm 0.000$ | $1.539 \pm 0.000$ |
| 10 | 1000 | $0.414 \pm 0.000$ | $114.042 \pm 0.000$ | $0.915 \pm 0.000$ | $111.357 \pm 0.000$ |
| 50 | 0 | $0.688 \pm 0.000$ | $0.688 \pm 0.000$ | $0.915 \pm 0.000$ | $1.064 \pm 0.000$ |
| 50 | 10 | $0.628 \pm 0.000$ | $0.635 \pm 0.000$ | $0.915 \pm 0.000$ | $0.909 \pm 0.000$ |
| 50 | 50 | $0.553 \pm 0.000$ | $0.732 \pm 0.000$ | $0.915 \pm 0.000$ | $1.296 \pm 0.000$ |
| 50 | 100 | $0.774 \pm 0.000$ | $1.857 \pm 0.000$ | $0.915 \pm 0.000$ | $2.311 \pm 0.000$ |
| 50 | 1000 | $0.970 \pm 0.000$ | $118.241 \pm 0.000$ | $0.915 \pm 0.000$ | $133.122 \pm 0.000$ |

Table 24: **LIFT fails to perform Ridge regression.**

LIFT is shown to perform standard linear regression still. This is as expected, as we have already shown that LIFT is robust to outliers in Sec. 4.4.

### D.4 The effect of contradicting prompts on finetuning

Here we illustrate how we design "contradicting prompts" to study the impact of the prior knowledge LMs. First, we generate the synthetic dataset based on $y = x_1 + x_2$. That is, this is nothing but "addition". We compare the performance of the following three prompts and their sample complexities:

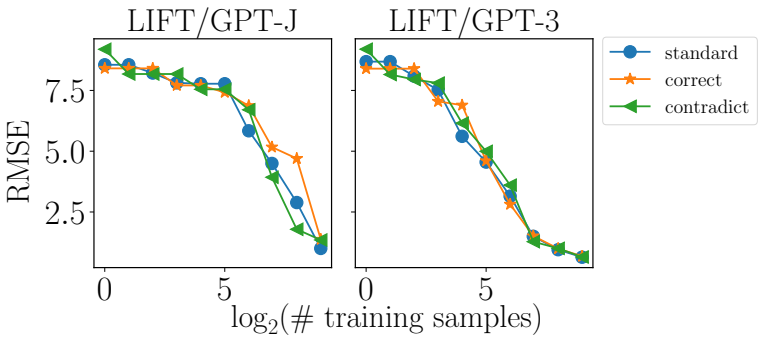



Figure 27: **Sample complexity of LIFT given different types of prompts.** In particular, standard prompts provide no prior knowledge, correct prompts provide correct prior knowledge, and contradict prompts provide contradicting prior knowledge.

1. "$x_1$ = `value1`, $x_2$ = `value2`, y ="
2. "$x_1$ = `value1`, $x_2$ = `value2`, $x_1 + x_2$ ="
3. "$x_1$ = `value1`, $x_2$ = `value2`, $x_1 - x_2$ =".

Note that the design of the second prompt reveals the proper relationship between $x_1, x_2$, and y, and the third prompt contradicts it. Intuitively, the second type of prompt may reduce the sample complexity of LIFT by making use of the prior knowledge of LMs, whereas the third type of prompt does the opposite. We generate the training set with 4, 8, 16, 32, 64, 125, 256, and 512 samples. We report the sample complexity of LIFT/GPT-J and LIFT/GPT-3 with respect to the three types of prompts in Fig. 27. We observe that contradicting prompts suffer from the highest RMSE, and correct prompts achieve the lowest RMSE when the number of training samples is only 4. However, when more training samples are provided, the benefit of correct prompts is not reflected in the results. We check whether the meaning of "-" has been modified after LIFT achieving low RMSE. However, it shows that the meaning of "-" is changed only when the same type of prompts are provided.

# E Discussion of Social Impacts

As LIFT greatly simplifies the machine learning pipeline, which only requires humans to provide the training dataset for specific tasks, it enables *no-code ML* for the masses. This means that someone who does not have prior knowledge on the implementation of ML frameworks can also use LIFT for their target non-language tasks: the only thing the customers need to do is design the input/output prompt properly in a language format. Therefore, LIFT can be applied to a wide range of areas such as credit loaning, disease diagnosis, and criminal sentencing. However, employing LIFT without careful justification or understanding can lead to undesired outcomes, such as discrimination, false information, etc.

Since most existing language models (LMs) are trained on a large amount of human-annotated data, LIFT could exhibit discrimination against different demographic groups (*e.g.*, gender, race, ethnicity) due to the bias existing in the training datasets. In other words, LIFT may prefer certain groups while making decisions in downstream tasks, especially when feature names and other demographic contexts are fed at training and inference time. This effect is exacerbated by the use of large pretrained language models (*i.e.*, GPT-J and GPT-3), which have been known to inherently contain bias [82]. The bias in the pretraining data for these large language models adds an opaque layer to regression and classification tasks beyond bias within the downstream finetuning data. Therefore, the adoption of LIFT in tasks that consider demographic information requires more consideration to avoid discrimination. To further remove the bias, users can also combine the existing fairness-aware reweighting mechanism [83, 84] and LIFT. Moreover, we raise a caveat about the risk of presenting false information of LIFT, especially in the health-critical area. It is very important to inform the users without a programming background about the significance of model evaluation to address this issue. Finally, we emphasize that more model evaluation steps are required when applying LIFT instead of using it as a panacea for all applications. With proper justifications, we believe that our work can significantly benefit society by providing a simple tool for handling various tasks.