In-Context Learning and Bayesian Inference

Madhur Panwar[*]
Microsoft Research India
t-mpanwar@microsoft.com

Kabir Ahuja[*]
University of Washington
kahuja@cs.washington.edu

Navin Goyal
Microsoft Research India
navingo@microsoft.com

Abstract

In-context learning (ICL) is one of the surprising and useful features of large language models and subject of intense research. Recently, stylized meta-learning-like ICL setups have been devised that train transformers on sequences of input-output pairs $(x, f(x))$ using the language modeling loss. The function f comes from a function class and generalization is checked by evaluation on sequences for unseen functions from the same class. One of the main discoveries in this line of research has been that for several function classes, such as linear regression, transformers successfully generalize to new functions in the class. However, it is unclear if transformers trained on multiple function classes (a setup closer to that of real-world LLMs) also exhibit this generalization. Moreover, the inductive biases of these models resulting in this generalization are not clearly understood. A model with unlimited training data and compute is a Bayesian predictor: it learns the pretraining distribution. In this paper, we empirically examine how far this Bayesian perspective can help us understand ICL. To this end, we generalize the previous meta-ICL setup to hierarchical meta-ICL setup which involves unions of multiple task families. We instantiate this setup on a diverse range of linear and nonlinear function families and find that transformers can do ICL in this setting as well. Where Bayesian inference is tractable, we find evidence that high-capacity transformers mimic the Bayesian predictor. Via the example of learning Fourier series, we also study the inductive bias for in-context learning. We find that in-context learning may or may not have simplicity bias depending on the pretraining data distribution. The Bayesian perspective provides insights into these inductive biases and how transformers perform a particular task when trained on multiple tasks.

1 Introduction

In-context learning (ICL) is one of the major ingredients behind the astounding performance of large language models (LLMs) Brown et al. [2020], Touvron et al. [2023]. Unlike traditional supervised learning, ICL is the ability to learn new functions f without weight updates from input-output examples $(x, f(x))$ provided as input at the test time; in other words, learning happens *in context*. For instance, given the prompt `up -> down, low -> high, small ->`, a pretrained LLM will likely produce output `big`: it apparently infers that the function in the two examples is the antonym of the input and applies it on the new input. This behavior often extends to more sophisticated and novel functions unlikely to have been seen during training and has been the subject of intense study, e.g., Min et al. [2022b], Webson and Pavlick [2022], Min et al. [2022a], Liu et al. [2023], Dong et al. [2023]. More broadly than its applications in NLP, ICL can also be viewed as providing a method for meta-learning Hospedales et al. [2022] where the model learns to learn a class of functions.

Theoretical understanding of ICL is an active area of research. Since the real-world datasets used for LLM training are difficult to model theoretically and are very large, ICL has also been studied in

[*]Equal contribution

stylized setups, e.g., Xie et al. [2022], Chan et al. [2022b], Garg et al. [2022], Wang et al. [2023], Hahn and Goyal [2023]. These setups study different facets of ICL. In this paper, we focus on the meta-learning-like framework of Garg et al. [2022]. Unlike in NLP where training is done on documents for the next-token prediction task, here the training and test data look similar in the sense that the training data consists of input of the form $((\boldsymbol{x}_1, f(\boldsymbol{x}_1)), \ldots, (\boldsymbol{x}_k, f(\boldsymbol{x}_k)), \boldsymbol{x}_{k+1})$ and output is $f(\boldsymbol{x}_{k+1})$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and are chosen i.i.d. from a distribution, and $f : \mathbb{R}^d \to \mathbb{R}$ is a function from a class of functions, for example, linear functions or shallow neural networks. We call this setup **MICL**. A striking discovery in Garg et al. [2022] was that for several function classes, transformer-based language models during pretraining learn to implicitly implement well-known algorithms for learning those functions in context. For example, when shown 20 examples of the form $(\boldsymbol{x}, \boldsymbol{w}^T\boldsymbol{x})$, where $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^{20}$, the model correctly outputs $\boldsymbol{w}_{\text{test}}^T \boldsymbol{x}_{\text{test}}$ on test input $\boldsymbol{x}_{\text{test}}$. Apart from linear regression, they show that for sparse linear regression and shallow neural networks the trained model appears to implement well-known algorithms; and for decision trees, the trained model does better than baselines. Two follow-up works Akyürek et al. [2022] and von Oswald et al. [2022] largely focused on the case of linear regression. Among other things, they showed that transformers with one attention layer learn to implement one step of gradient descent on the linear regression objective with further characterization of the higher number of layers. In our work, we question: *Can language models be trained to learn a variety of function classes and their mixtures? Can we explain the in-context behavior of language models and their inductive biases?*

Bayesian predictor. An ideal language model (LM) with unlimited training data and compute would learn the pretraining distribution as that results in the smallest loss. Such an LM produces the output by simply sampling from the pretraining distribution conditioned on the input prompt. Such an ideal model is often called *Bayesian predictor*. Many works make the assumption that trained LMs are Bayesian predictors, e.g. Saunshi et al. [2021], Xie et al. [2022], Wang et al. [2023]. Most relevant to the present paper, Akyürek et al. [2022] show that in the MICL setup for linear regression, in the underdetermined setting, namely when the number of examples is smaller than the dimension of the input, the model learns to output the least L_2-norm solution which is the Bayes-optimal prediction. In this paper we empirically examine how general this behavior is across choices of tasks.

Prior work has investigated related questions but we are not aware of any extensive empirical verification. E.g., Xie et al. [2022] study a synthetic setup where the pretraining distribution is given by a mixture of hidden Markov models and show that the prediction error of ICL approaches Bayes-optimality as the number of in-context examples approach infinity. In contrast, we test the Bayesian hypothesis for ICL over a wide class of function families and show evidence for equivalence with Bayesian predictor at all prompt lengths. Also closely related, Müller et al. [2022], Hollmann et al. [2023] train transformer models by sampling data from a prior distribution (Prior Fitted Networks), so it could approximate the posterior predictive distribution at inference time. While these works focus on training models to approximate posterior distributions for solving practical tasks (tabular data), our objective is to understand how in-context learning works in transformers and to what extent we can explain it as performing Bayesian Inference on the pre-training distribution.

Simplicity bias. Simplicity bias, the tendency of machine learning algorithms to prefer simpler hypotheses among those consistent with the data, has been suggested as the basis of the success of neural networks. There are many notions of simplicity [Mingard et al., 2023, Goldblum et al., 2023]. Does in-context learning also enjoy a simplicity bias like pretraining?

Our contributions. In brief, our contributions are

1. *A setup for studying ICL for multiple function families*: First, we extend the MICL setup from Garg et al. [2022] to include multiple families of functions. For example, the prompts could be generated from a mixture of tasks where the function f is chosen to be either a linear function or a decision tree with equal probability. We call this extended setup HMICL. We experimentally study HMICL and find that high-capacity transformer models can learn in context when given such task mixtures. (We use the term "high-capacity" informally; more precisely, it means that for the task at hand there is a sufficiently large model with the desired property.)

2. *High-capacity transformers perform Bayesian inference during ICL:* To understand how this ability arises we investigate in depth whether high-capacity transformers simulate the Bayesian predictor. This motivates us to choose a diverse set of linear and nonlinear function classes as well as prior distributions in HMICL and MICL setups. Function classes we consider were chosen because either they permit efficient and explicit Bayesian inference or have strong baselines. We provide direct and indirect evidence that indeed high-capacity transformers often mimic the Bayesian predictor. The ability to solve task mixtures arises naturally as a consequence of Bayesian prediction, in contrast to the algorithm selection and execution view from prior work (e.g., Bai et al. [2023]).

3. *Link between ICL inductive bias with the pretraining data distribution:* We also investigate the inductive bias in a simple setting for learning functions given by Fourier series. If ICL is biased towards fitting functions of lower maximum frequency, this would suggest that it has a bias for lower frequencies like the spectral bias for pretraining. We find that the model mimics the Bayesian predictor; the ICL inductive bias of the model is determined by the pretraining data distribution: if during pretraining all frequencies are equally represented, then during ICL the LM shows no preference for any frequency. On the other hand, if lower frequencies are predominantly present in the pretraining data distribution then during ICL the LM prefers lower frequencies. Chan et al. [2022a,b] studies the inductive biases of transformers for ICL and the effect of pretraining data distribution on them. However, the problem setting in these papers is very different from ours and they do not consider simplicity bias.

2 Background

We first discuss the in-context learning setup for learning function classes as introduced in Garg et al. [2022], which we call **Meta-ICL** or **MICL**. Let $\mathcal{D}_{\mathcal{X}}$ be a probability distribution on $\mathbb{R}^d$. Let $\mathcal{F}$ be a family of functions $f : \mathbb{R}^d \to \mathbb{R}$ and let $\mathcal{D}_{\mathcal{F}}$ be a distribution on $\mathcal{F}$. For simplicity, we often use $f \sim \mathcal{F}$ to mean $f \sim \mathcal{D}_{\mathcal{F}}$. We overload the term function class to encompass both function definition as well as priors on its parameters. Hence, linear regression with a standard gaussian prior and a sparse prior will be considered different function classes based on our notation.

To construct a prompt $P = \big(\boldsymbol{x}_1, f(\boldsymbol{x}_i), \cdots, \boldsymbol{x}_p, f(\boldsymbol{x}_p), \boldsymbol{x}_{p+1}\big)$ of length p, we sample inputs $\boldsymbol{x}_i \sim \mathcal{D}_{\mathcal{X}}$ i.i.d. for $i \in \{1, \cdots p\}$. A transformer-based language model M_θ is trained to predict $f(\boldsymbol{x}_{p+1})$ given P, using the objective: $\min_\theta \mathbb{E}_{f, \boldsymbol{x}_{1:p}} \left[\frac{1}{p+1} \sum_{i=0}^{p} \ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right)\right]$, where P^i denotes the sub-prompt containing the first i input-output examples as well as the $(i+1)$-th input, i.e. $\big(\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots, \boldsymbol{x}_i, f(\boldsymbol{x}_i), \boldsymbol{x}_{i+1}\big)$ and $\boldsymbol{x}_{1:p} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$. While other choices of the loss function $\ell\big(\cdot, \cdot\big)$ are possible, since we study regression problems we use the squared-error loss (i.e., $\ell(y, y') = (y - y')^2$) in accordance with Garg et al. [2022].

At test time, we present the model with few-shot prompts P_{test} that were unseen during training with high probability and compute the error when provided k in-context examples: $\texttt{loss@}k = \mathbb{E}_{f, P_{\text{test}}}\left[\ell\left(M_\theta(P^k), f(\boldsymbol{x}_{k+1})\right)\right]$, for $k \in \{1, \cdots, p\}$.

PME. We mentioned that an ideal model would learn the pretraining distribution. This happens when using the cross-entropy loss. Since we use the square loss in the objective definition, the predictions of the model can be computed using the posterior mean estimator (PME) from Bayesian statistics. For each prompt length i we can compute PME by taking the corresponding summand in objective definition above, which will be given by $M_\theta(P^i) = \mathbb{E}_f\left[f(\boldsymbol{x}_{i+1}) \,|\, P^i\right]$ for all $i \leq p$. This is the optimal solution for prompt P, which we refer to as PME. Please refer to §A.1 for technical details.

2.1 Hierarchical Meta-ICL

We generalize the MICL setup, where instead of training transformers from functions sampled from a single function class, we sample them from a mixture of function classes. Formally, we define a mixture of function classes using a set of m function classes $\boldsymbol{\mathcal{F}} = \{\mathcal{F}_1, \cdots, \mathcal{F}_m\}$ and sampling probabilities $\boldsymbol{\alpha} = [\alpha_1, \cdots \alpha_m]^T$ with $\sum_{i=1}^{m} \alpha_i = 1$. We use $\boldsymbol{\alpha}$ to sample a function class for constructing the training prompt P. We assume the input distribution $\mathcal{D}_{\mathcal{X}}$ to be same for each class $\mathcal{F}_{T_i}$. More concretely, the sampling process for P is defined as: i) $\mathcal{F}_i \sim \boldsymbol{\mathcal{F}}$ s.t. $\mathbb{P}(\mathcal{F} = \mathcal{F}_i) = \alpha_i$; ii) $f \sim \mathcal{F}_i$; iii) $\boldsymbol{x}_j \sim \mathcal{D}_{\mathcal{X}}, \forall j \in \{1, \cdots, p\}$; and finally, iv) $P = \big(\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots \boldsymbol{x}_p, f(\boldsymbol{x}_p), \boldsymbol{x}_{p+1}\big)$.

We call this setup **Hierarchical Meta-ICL** or **HMICL**, as there is an additional first step for sampling the function class in the sampling procedure. Note that the MICL setup can be viewed as a special case of HMICL where $m = 1$. The HMICL setting presents a more advanced scenario to validate the generality of in-context learning and whether it can be explained in transformers by Bayesian inference. Further, our HMICL setup is also arguably closer to the in-context learning in practical LLMs which can realize different classes of tasks (sentiment analysis, QA, summarization, etc.) depending on the few-shot in-context examples.

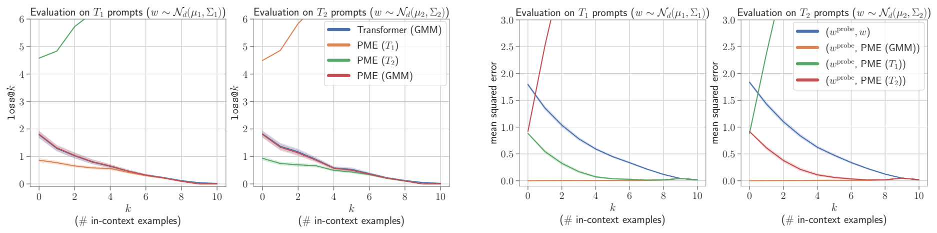


Figure 1: Transformers simulate PME when trained on dense regression task-mixture with weights having a mixture of Gaussian prior (GMM). *(left)*: Comparing the performance of the Transformer with PME of individual Gaussian components (PME (T_1) and PME (T_2)) and of the mixture PME (GMM). *(right)*: MSE between the probed weights of the Transformer and PMEs.

The PME for the hierarchical case is given by:

$$M_{\theta,\boldsymbol{\mathcal{F}}}(P) = \beta_1 M_{\theta,\mathcal{F}_1}(P) + \ldots + \beta_m M_{\theta,\mathcal{F}_m}(P), \tag{1}$$

where $\beta_i = \alpha_i p_i(P)/p_{\boldsymbol{\mathcal{F}}}(P)$ for $i \leq m$. Probability density $p_i(\cdot)$ is induced by the function class $\mathcal{F}_i$ on the prompts in a natural way, and $p_{\boldsymbol{\mathcal{F}}}(P) = \alpha_i p_i(P) + \cdots + \alpha_m p_m(P)$. Please refer to §A.1 in the Appendix for the derivation. The models are trained with the squared error loss mentioned above and at test time we evaluate `loss@`k for each task individually.

2.2 Model and training details

We use the decoder-only transformer architecture Vaswani et al. [2017] as used in the GPT models Radford et al. [2019]. Unless specified otherwise, we use 12 layers, 8 heads, and a hidden size (d_h) of 256 in the architecture for all of our experiments. We use a batch size of 64 and train the model for 500k steps. For encoding the inputs $\boldsymbol{x}_i$'s and $f(\boldsymbol{x}_i)$'s, we use the same scheme as Garg et al. [2022] which uses a linear map $\boldsymbol{E} \in \mathbb{R}^{d_h \times d}$ to embed the inputs $\boldsymbol{x}_i$'s as $\boldsymbol{E}\boldsymbol{x}_i$ and $f(\boldsymbol{x}_i)$'s as $\boldsymbol{E} f_{\text{pad}}(\boldsymbol{x}_i)$, where $f_{\text{pad}}(\boldsymbol{x}_i) = [f(\boldsymbol{x}_i), \boldsymbol{0}_{d-1}]^T \in \mathbb{R}^d$. In all of our experiments except the ones concerning the Fourier series, we choose $\mathcal{D}_{\mathcal{X}}$ as the standard normal distribution i.e. $\mathcal{N}(0, 1)$, unless specified otherwise. To accelerate training, we also use curriculum learning like Garg et al. [2022] for all our experiments where we start with simpler function distributions (lower values of d and p) at the beginning of training and increase the complexity as we train the model. The role of curriculum and other factors in models acquiring multi-task ICL is discussed in §C.4 in Appendix.

3 Transformers can in-context learn task mixtures

In this section, we provide evidence that transformers' ability to solve mixture of tasks arises naturally from the Bayesian perspective. We start with a Gaussian Mixture Models (GMMs) example where the exact Bayesian solution is tractable and later discuss results for more complex mixtures.

3.1 Gaussian Mixture Models (GMMs)

We define a mixture of dense-linear regression classes $\boldsymbol{\mathcal{F}}_{\text{GMM}} = \{\mathcal{F}_{\text{DR}_1}, \cdots, \mathcal{F}_{\text{DR}_m}\}$, where $\mathcal{F}_i = \left\{f : \boldsymbol{x} \mapsto \boldsymbol{w}_i^T \boldsymbol{x} \,|\, \boldsymbol{w}_i \in \mathbb{R}^d\right\}$ and $\boldsymbol{w}_i \sim \mathcal{N}_d(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. In other words, each function class in the mixture corresponds to dense regression with Gaussian prior on weights (but different means or covariance matrices). We report experiments with $m = 2$ here. The mean vectors are given by $\boldsymbol{\mu}_1 = (3, 0, .., 0)$ and $\boldsymbol{\mu}_2 = (-3, 0, ..., 0)$ for the two classes. The covariance matrices are equal ($\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \boldsymbol{\Sigma}^*$), where $\boldsymbol{\Sigma}^*$ is the identity matrix $\boldsymbol{I}_d$ with the top-left entry replaced by 0. Note that we can equivalently view this setup by considering the prior on weights as a mixture of Gaussians i.e. $p_M(\boldsymbol{w}) = \alpha_1 \mathcal{N}_d(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \alpha_2 \mathcal{N}_d(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$. We call the two function classes T_1 and T_2. We train the transformer on a uniform mixture ($\alpha_1 = \alpha_2 = \frac{1}{2}$), with $d = 10$ and the prompt length $p = 10$.

Recovering implied weights. To provide a stronger evidence for the Bayesian hypothesis, apart from the loss curves, we also extract the weights implied by transformers for solving the regression task in-context. Following Akyürek et al. [2022], we do this by generating model's predictions $\{y_i'\}$ on the test inputs $\{\boldsymbol{x}_i'\}_{i=1}^{2d} \sim \mathcal{D}_{\mathcal{X}}$ and then solving the system of equations to recover $\boldsymbol{w}^{\text{probe}}$. We then compare the implied weights $\boldsymbol{w}^{\text{probe}}$ with the ground truth weights $\boldsymbol{w}$ as well as the weights extracted from different baselines by computing the their MSE.

Results. In Figure 1 (*left*), we note that Transformer's errors almost exactly align with those of the PME of the mixture, PME (GMM), when prompts come from either T_1 or T_2. (For details on computation of PME, please refer to §C.1 in Appendix). For each plot, let T_{prompt} and T_{other}

denote the component from which prompts are provided and the other component respectively. When $d = 10$ examples from T_{prompt} have been provided, the Transformer, PME (T_{prompt}), and PME (GMM) all converge to the same minimum error of 0. This shows that Transformer is simulating PME (GMM), which converges to PME (T_{prompt}) at $k = d$. PME (T_{other})'s errors keep increasing as more examples are provided. These observations are in line with Eq. 3: As more examples from the prompt are observed, the weights of individual PMEs used to compute the PME (GMM) (i.e., the β's) evolve such that the contribution of T_{prompt} increases in the mixture with k (Fig. 15 in the Appendix). In Figure 1 (*right*), MSE between weights from different predictors are plotted. Transformer's implied weights are almost exactly identical to PME (GMM) for all k. Please refer to §C.1 for additional details and results.

More complex mixtures. We test the generality of the phenomenon discussed above for more complex mixtures, involving mixtures of two or three different linear inverse problems (e.g. dense regression, sparse regression, sign vector regression) as well as some mixtures involving non-linear function classes like neural networks and decision trees. In all of these cases we observe that transformers trained on the mixtures are able to generalize on the new functions from the mixture of function classes and match the the performance of single-function class transformer models depending upon the distribution of input prompt. Please refer to §C.2 for details.

Implications. Our GMM experiments challenge the existing explanations for the multi-task case, e.g. the models first recognizes the task and then solves it. When viewed through the Bayesian lens, transformers do not need to recognize the task separately and recognition and solution are intertwined as we show in Equation 1.

4 Simplicity bias in ICL?

In this section, we explore if transformers exhibhit simplicity bias in ICL. In other words, when given a prompt containing input output examples, do they prefer to fit simpler functions among those that fit the prompt? To study this behavior we consider the Fourier Series function class, where the output is a linear function of sine and cosine functions of different frequencies. By training transformers on this class, during ICL we can study if transformers prefer fitting lower-frequency functions to the prompt over higher frequencies, which can help us study the presence of a simplicity bias.

More formally, we can define Fourier series by the following expansion: $f(x) = a_0 + \sum_{n=1}^{N} a_n \cos(n\pi x/L) + \sum_{n=1}^{N} b_n \sin(n\pi x/L)$ where, $x \in [-L, L]$, and a_0, a_n's and b_n's are known as Fourier coefficients and $\cos n\pi/L$ and $\sin n\pi/L$ define the frequency n components.

MICL Setup. In the MICL setup we train transformer on a single function class defined as $\mathcal{F}_{\Phi_N}^{\text{fourier}} = \left\{f(\cdot;\Phi_N) | f(\boldsymbol{x};\Phi) = \boldsymbol{w}^T\Phi_N(\boldsymbol{x}), \boldsymbol{w} \in \mathbb{R}^d\right\}$ with standard gaussian prior on weights $\boldsymbol{w}$. Note that here Φ_N as the Fourier feature map i.e. $\Phi_N(x) = [1, \cos(\pi x/L), \cdots, \cos(N\pi x/L), \sin(\pi x/L), \cdots, \sin(N\pi x/L)]^T$. For training transformers to in-context-learn $\mathcal{F}_{\Phi_N}^{\text{fourier}}$, we fix a value of N and sample functions $f \in \mathcal{F}_{\Phi_N}^{\text{fourier}}$. We consider the inputs to be scalars, i.e. $x_i \in [-L, L]$ and we sample them i.i.d. from the uniform distribution on the domain: $x_i \sim \mathcal{U}(-L, L)$. In all of our experiments, we consider $N = 10$ and $L = 5$. At test time we evaluate on $\mathcal{F}_{\Phi_M}^{\text{fourier}}$ for $M \in [1, 10]$, i.e. during evaluation we also prompt the model with functions with different maximum frequency as seen during training.

HMICL Setup. We also consider a mixture of Fourier series function classes with different maximum frequencies, i.e. $\mathcal{F}_{\Phi_{1:N}}^{\text{fourier}} = \{\mathcal{F}_{\Phi_1}^{\text{fourier}}, \cdots, \mathcal{F}_{\Phi_N}^{\text{fourier}}\}$. We consider $N = 10$ in our experiments and train the models using a uniform mixture with normalization. During evaluation, we test individually on each $\mathcal{F}_{\Phi_M}^{\text{fourier}}$, where $M \in [1, N]$.

Measuring inductive biases. To study simplicity bias during ICL, we propose a method to recover implied frequency from the transformer model. We start by sampling in-context examples $(x_1, f(x_1), \cdots x_k, f(x_k))$, and given the context obtain the model's predictions on a set of m test inputs $\{x'_i\}_{i=1}^m$, i.e. $y'_i = M_\theta\left((x_1, f(x_1), \cdots x_k, f(x_k), x'_i)\right)$. We can then perform Discrete Fourier Transform (DFT) on $\{y'_1, \cdots, y'_m\}$ to obtain the Fourier coefficients of the function output by the model, which we can analyze to understand the dominant frequencies.

Results. In both MICL and HMICL setups discussed above we observe that transformers are able to in-context learn these function classes and match the performance of the Bayesian predictor or strong baselines. Since, in this section we are primarily interested in studying the simplicity bias, here we only report the plots for frequencies recovered from transformers at different prompt lengths

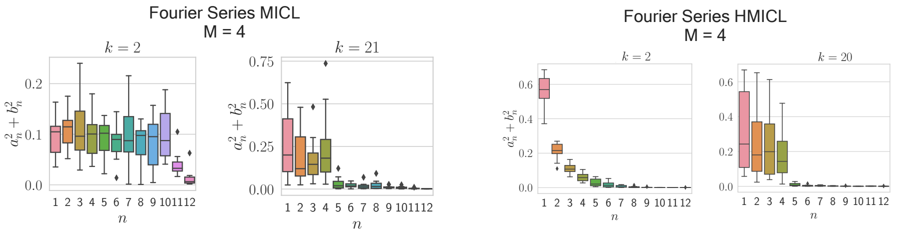


Figure 2: Measuring the frequencies of the simulated function during ICL by transformer.

in Figure 2 (more details in Figures 6 and 25 of Appendix). As can be seen in Figure 2 (*left*), in the single function class case, transformers exhibit no bias towards any particular frequency. For small prompt lengths ($k = 2$), all N frequencies receive similar absolute value of coefficients as implied by the transformer. As more examples are provided ($k = 21$), transformer is able to recognize the gold maximum frequency ($M = 4$) from the in-context examples, and hence coefficients are near zero for $n > M$, but as such there is no bias towards any particular frequencies. However, when we consider the mixture case in Figure 2 (*right*), the situation is different. We see a clear bias for lower frequencies at small prompt lengths; however, when given sufficiently many examples they are able to recover the gold frequencies. This simplicity bias can be traced to the training dataset for the mixture since lower frequencies are present in most of the functions of the mixture while higher frequencies will be more rare: Frequency 1 will be present in all the function classes whereas frequency N will be present only in $\mathcal{F}_{\Phi_N}^{\text{fourier}}$. We perform additional experiments biasing pre-training distribution to high frequencies and observe complexity bias during ICL (Appendix §C.3.1).

Implications. These results suggest that the simplicity bias (or lack thereof) during ICL can be attributed to the pre-training distribution which follows naturally from the Bayesian perspective i.e. the biases in the prior are reflected in the posterior. Transformers do not add any extra inductive bias of their own as they emulate the Bayesian predictor.

5 Summary of further results

In this section, we summarize further results for in-context learning and generality of Bayesian hypothesis that are provided in the Appendix. We test the Bayesian hypothesis on a variety of linear and non-linear inverse problems in both MICL and HMICL setups and find the transformers are able to in-context learn and generalize to unseen functions from these function classes. Where possible, we provide the Bayesian predictor for comparison and establish the agreement in behavior of transformer and the Bayesian predictor (PME). The cases where PME is intractable, we report comparisons with strong baselines that have been show to be near optimal in prior work. Among the class of linear problems, we test on Dense, Sparse, Sign Vector, Low Rank and Skewed Covariance Regression. For these problems, we show that the transformers match the Bayesian predictor (or the strong baselines), i.e., their squared errors as well as the weights (of the implied linear function) agree. For the non-linear case, we explore regression problems for Fourier Series, Degree-2 Monomials, Random Fourier Features and Haar Wavelets. Further, we also note that in the HMICL setup, generalization to functions from the mixture might depend on different factors. Normalizing the outputs from each function class turns out to be an important factor for HMICL to work. We provide the complete details for each of these function families and corresponding results in the Appendix §B and §C.

6 Conclusion

In this paper we provided empirical evidence that in-context learning in transformers works effectively in the HMICL setup and Bayesian perspective could serve as a unifying explanation for ICL. In particular, it can explain how the inductive bias of ICL comes from the pretraining distribution and how transformers solve mixtures of tasks. There are many interesting directions for future work. Much more remains to be done to determine how extensively transformers mimic the Bayesian predictor. Relation between the pretraining distribution and ICL inductive bias needs to be further fleshed out. What are the implications of this inductive bias to real-world LLMs? Can Bayesian inference explain the remarkable ability of in-context learning in transformers on natural language data? Finally, we treated transformers as black boxes: opening the box and uncovering the underlying mechanisms transformers use to do Bayesian prediction would be very interesting.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *CoRR*, abs/2211.15661, 2022. doi: 10.48550/arXiv.2211.15661. URL `https://doi.org/10.48550/arXiv.2211.15661`.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

E.J. Candes and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 2005. doi: 10.1109/TIT.2005.858979.

Stephanie Chan, Adam Santoro, Andrew Lampinen, Jane Wang, Aaditya Singh, Pierre Richemond, James McClelland, and Felix Hill. Data distributional properties drive emergent in-context learning in transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 18878–18891. Curran Associates, Inc., 2022a. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/77c6ccacfd9962e2307fc64680fc5ace-Paper-Conference.pdf`.

Stephanie C. Y. Chan, Ishita Dasgupta, Junkyung Kim, Dharshan Kumaran, Andrew K. Lampinen, and Felix Hill. Transformers generalize differently from information stored in context vs in weights. *CoRR*, abs/2210.05675, 2022b. doi: 10.48550/arXiv.2210.05675. URL `https://doi.org/10.48550/arXiv.2210.05675`.

Venkat Chandrasekaran, Benjamin Recht, Pablo A. Parrilo, and Alan S. Willsky. The convex geometry of linear inverse problems. *Foundations of Computational Mathematics*, 12(6):805–849, oct 2012. doi: 10.1007/s10208-012-9135-7. URL `https://doi.org/10.1007%2Fs10208-012-9135-7`.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. A survey on in-context learning, 2023.

D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. doi: 10.1109/TIT.2006.871582.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30583–30598. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/c529dba08a146ea8d6cf715ae8930cbe-Paper-Conference.pdf`.

Micah Goldblum, Marc Finzi, Keefer Rowan, and Andrew Gordon Wilson. The no free lunch theorem, kolmogorov complexity, and the role of inductive biases in machine learning. *CoRR*, abs/2304.05366, 2023. doi: 10.48550/arXiv.2304.05366. URL `https://doi.org/10.48550/arXiv.2304.05366`.

Michael Hahn and Navin Goyal. A theory of emergent in-context learning as implicit structure induction. *CoRR*, abs/2303.07971, 2023. doi: 10.48550/arXiv.2303.07971. URL `https://doi.org/10.48550/arXiv.2303.07971`.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. 2022.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=cp5PvcI6w8_`.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(09):5149–5169, sep 2022. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3079209.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9):195:1–195:35, 2023. doi: 10.1145/3560815. URL `https://doi.org/10.1145/3560815`.

O.L. Mangasarian and Benjamin Recht. Probability of unique integer solution to a system of linear equations. *European Journal of Operational Research*, 214(1):27–30, 2011. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.04.010. URL `https://www.sciencedirect.com/science/article/pii/S0377221711003511`.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. MetaICL: Learning to learn in context. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2791–2809, Seattle, United States, July 2022a. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.201. URL `https://aclanthology.org/2022.naacl-main.201`.

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? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates, December 2022b. Association for Computational Linguistics. URL `https://aclanthology.org/2022.emnlp-main.759`.

Chris Mingard, Henry Rees, Guillermo Valle Pérez, and Ard A. Louis. Do deep neural networks have an inbuilt occam's razor? *CoRR*, abs/2304.06670, 2023. doi: 10.48550/arXiv.2304.06670. URL `https://doi.org/10.48550/arXiv.2304.06670`.

Aaron Mueller and Tal Linzen. How to plant trees in language models: Data and architectural effects on the emergence of syntactic inductive biases. 2023.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=KSugKcbNf9`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf*, 1(8):9, 2019.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL `https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf`.

Yasaman Razeghi, Robert L. Logan IV au2, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot reasoning. 2022.

Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. A mathematical exploration of why language models help solve downstream tasks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=vVjIW3sEc1s`.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. doi: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x. URL `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x`.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. 2022.

Xinyi Wang, Wanrong Zhu, and William Yang Wang. Large language models are implicitly topic models: Explaining and finding good demonstrations for in-context learning. *CoRR*, abs/2301.11916, 2023. doi: 10.48550/arXiv.2301.11916. URL `https://doi.org/10.48550/arXiv.2301.11916`.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2300–2344, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.167. URL `https://aclanthology.org/2022.naacl-main.167`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://aclanthology.org/2020.emnlp-demos.6`.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL `https://openreview.net/forum?id=RdJVFCHjUMI`.

Contents

A Technical Details

A.1 PME Theoretical Details

We mentioned earlier that an ideal LM would learn the pretraining distribution. This happens when using the cross-entropy loss. Since we use the square loss in the ICL training objective, the predictions of the model can be computed using the posterior mean estimator (PME) from Bayesian statistics.

For each prompt length i we can compute PME by taking the corresponding summand in the ICL training objective

$$\begin{aligned}
\min_\theta \mathbb{E}_{f,\boldsymbol{x}_{1:i}}\, \ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right) &= \min_\theta \mathbb{E}_{f,P^i}\, \ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right) \\
&= \min_\theta \mathbb{E}_{P^i}\, \mathbb{E}_f \left[\ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right) \mid P^i\right] \\
&= \mathbb{E}_{P^i}\, \min_\theta \mathbb{E}_f \left[\ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right) \mid P^i\right].
\end{aligned}$$

The inner minimization is seen to be achieved by $M_\theta(P^i) = \mathbb{E}_f\left[f(\boldsymbol{x}_{i+1}) \mid P^i\right]$. This is the optimal solution for prompt P^i and what we refer to as PME.

PME for a task mixture. We describe the PME for a mixture of function classes. For simplicity we confine ourselves to mixtures of two function classes; extension to more function classes is analogous. Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be two function classes specified by probability distributions $\mathcal{D}_{\mathcal{F}_1}$ and $\mathcal{D}_{\mathcal{F}_2}$, resp. As in the single function class case, the inputs $\boldsymbol{x}$ are chosen i.i.d. from a common distribution $\mathcal{D}_\mathcal{X}$. For $\alpha_1, \alpha_2 \in [0,1]$ with $\alpha_1 + \alpha_2 = 1$, an (α_1, α_2)-mixture $\boldsymbol{\mathcal{F}}$ of $\mathcal{F}_1$ and $\mathcal{F}_2$ is the meta-task in which the prompt $P = \left(\boldsymbol{x}_1, f(\boldsymbol{x}_i), \cdots, \boldsymbol{x}_p, f(\boldsymbol{x}_p), \boldsymbol{x}_{p+1}\right)$ is constructed by first picking task $\mathcal{F}_i$ with probability α_i for $i \in \{1,2\}$ and then picking $f \sim \mathcal{D}_{\mathcal{F}_i}$. Thus $p_{\boldsymbol{\mathcal{F}}}(f) = \alpha_1 p_{\mathcal{F}_1}(f) + \alpha_2 p_{\mathcal{F}_2}(f)$, where $p_\mathcal{F}(\cdot)$ is the probability density under function class $\mathcal{F}$ which defines $\mathcal{D}_\mathcal{F}$. For conciseness in the following we use $p_1(\cdot)$ for $p_{\mathcal{F}_1}(\cdot)$ etc. Now recall that PME for function class $\mathcal{F}$ is given by

$$M_{\theta,\mathcal{F}}(P) = \mathbb{E}_{f\sim\mathcal{D}_\mathcal{F}}\left[f(\boldsymbol{x}_{p+1}) \mid P\right] = \int p_\mathcal{F}(f|P)\, f(x)\, \mathrm{d}f. \tag{2}$$

We would like to compute this in terms of PMEs for $\mathcal{F}_1$ and $\mathcal{F}_2$. To this end, we first compute

$$\begin{aligned}
p_{\boldsymbol{\mathcal{F}}}(f|P) = \frac{p_{\boldsymbol{\mathcal{F}}}(P|f)p_{\boldsymbol{\mathcal{F}}}(f)}{p_{\boldsymbol{\mathcal{F}}}(P)} = \frac{p(P|f)p_{\boldsymbol{\mathcal{F}}}(f)}{p_{\boldsymbol{\mathcal{F}}}(P)} &= \frac{p(P|f)}{p_{\boldsymbol{\mathcal{F}}}(P)}\left[\alpha_1 p_1(f) + \alpha_2 p_2(f)\right] \\
&= \frac{\alpha_1 p_1(P)}{p_{\boldsymbol{\mathcal{F}}}(P)}\frac{p(P|f)p_1(f)}{p_1(P)} + \frac{\alpha_2 p_2(P)}{p_{\boldsymbol{\mathcal{F}}}(P)}\frac{p(P|f)p_2(f)}{p_2(P)} \\
&= \frac{\alpha_1 p_1(P)}{p_{\boldsymbol{\mathcal{F}}}(P)} p_1(f|P) + \frac{\alpha_2 p_2(P)}{p_{\boldsymbol{\mathcal{F}}}(P)} p_2(f|P) \\
&= \beta_1\, p_1(f|P) + \beta_2\, p_2(f|P),
\end{aligned}$$

where $\beta_1 = \frac{\alpha_1 p_1(P)}{p_{\boldsymbol{\mathcal{F}}}(P)}$ and $\beta_2 = \frac{\alpha_2 p_2(P)}{p_{\boldsymbol{\mathcal{F}}}(P)}$. Plugging this in equation 2 we get

$$M_{\theta,\boldsymbol{\mathcal{F}}}(P) = \beta_1 \int p_1(f|P)\, f(x)\, \mathrm{d}f + \beta_2 \int p_2(f|P)\, f(x)\, \mathrm{d}f = \beta_1 M_{\theta,\mathcal{F}_1}(P) + \beta_2 M_{\theta,\mathcal{F}_2}(P). \tag{3}$$

A.2 Experimental Setup

We use Adam optimizer Kingma and Ba [2015] to train our models. We train all of our models with curriculum and observe that curriculum helps in faster convergence, i.e., the same optima can also be achieved by training the model for more training steps as also noted by Garg et al. [2022]. Table 1 states the curriculum used for each experiment, where the syntax followed for each column specifying curriculum is `[start, end, increment, interval]`. The value of the said attribute goes from `start` to `end`, increasing by `increment` every `interval` train steps. Our experiments were conducted on a system comprising 32 NVIDIA V100 16GB GPUs. The cumulative training time of all models for this project was $\sim$ 30,000 GPU hours. While reporting the results, the error is averaged over 1280 prompts and shaded regions denote a 90% confidence interval over 1000 bootstrap trials.

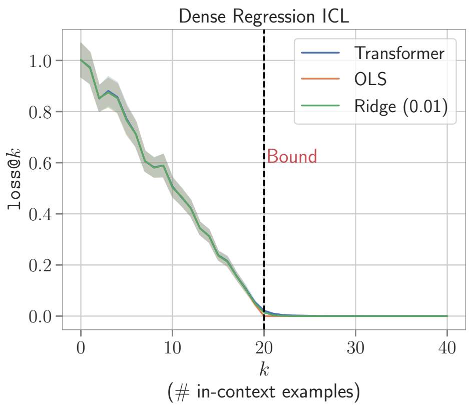
We adapt Garg et al. [2022] code-base for our experiments. We use PytorchPaszke et al. [2019] and Huggingface TransformersWolf et al. [2020] libraries to implement the model architecture and training procedure. For the baselines against which we compare transformers, we use `scikit-learn`'s [2] implementation of OLS, Ridge and Lasso, and for L_∞ and L_* norm minimization given the linear constraints we use CVXPY[3].

[2]https://scikit-learn.org/stable/index.html

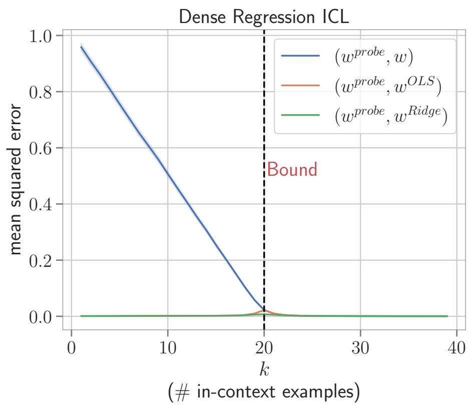
[3]`https://www.cvxpy.org/`

Table 1: The values of curriculum attributes used for each experiment. C_d, C_p and C_{freq} denote the curriculum on number of input dimensions (d), number of points (p) and maximum frequency N (for Fourier Series).

Experiment	**Section**	C_d	C_p	$C_{\textbf{freq}}$
Dense, Sparse and Sign-Vector Regression	§B.1.1	$[5, 20, 1, 2000]$	$[10, 40, 2, 2000]$	n/a
Low-Rank Regression	§B.1.1	Fixed ($d = 100$)	Fixed ($p = 114$)	n/a
Fourier Series	§B.2.1	Fixed ($d = 1$)	$[7, 43, 4, 2000]$	$[1, 10, 1, 2000]$
Fourier Series Mixture	§4	Fixed ($d = 1$)	Fixed (p = 40)	Fixed ($N = 10$)
GMM Regression ($d = 10, p = 10$)	§3.1, §C.1	[5, 10, 1, 2000]	[5, 10, 1, 2000]	n/a
GMM Regression ($d = 10, p = 20$)	§3.1, §C.1	[5, 10, 1, 2000]	[10, 20, 2, 2000]	n/a
Degree-2 Monomial Basis Regression	§B.2.3	Fixed ($d = 20$)	Fixed ($p = 290$)	n/a
Haar Wavelet Basis Regression	§B.2.4	Fixed ($d = 1$)	Fixed ($p = 32$)	n/a



(a)



(b)

Figure 3: Results on the Dense Regression tasks mentioned in section §B.1.1.

B Linear and Non-linear inverse problems

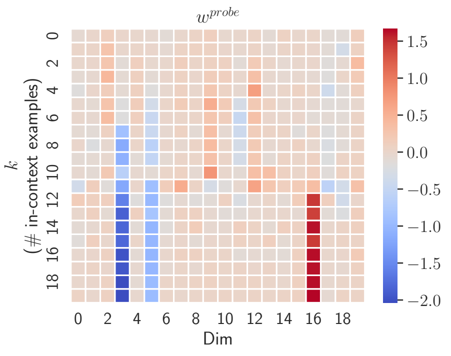
Here, we discuss the results mentioned in §5. Figure 3 shows the results on the Dense Regression task and our experiments corroborate the findings of Akyürek et al. [2022], where transformers not only obtain errors close to OLS and Ridge regression for the dense regression task (Figure 3a) but the extracted weights also very closely align with weights obtained by the two algorithms (Figure 3b). This does indicate that the model is able to simulate the PME behavior for the dense regression class.

For sparse and sign-vector regression, we also visualize the weights recovered from the transformer for one of the functions for each family. As can be observed in Figure 4, for sparse regression at sufficiently high prompt lengths ($k > 10$), the model is able to recognize the sparse structure of the problem and detect the non-zero elements of the weight vector. Similarly, the recovered weights for sign-vector regression beyond $k > 10$, start exhibiting the sign-vector nature of the weights (i.e. each component either being +1 or -1).

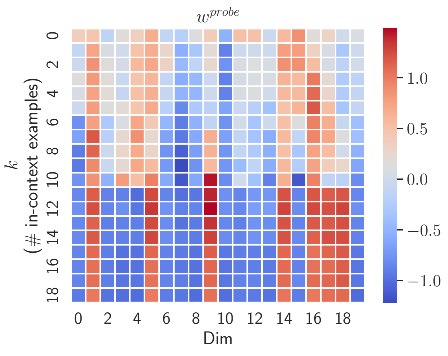
We evaluate transformers on a family of linear and non-linear regression tasks. On the tasks where it is possible to compute the Bayesian predictor, we study how close the solutions obtained by the transformer and Bayesian predictor are. In this section, we focus only on the MICL setting, while the mixture of tasks, i.e., HMICL, is discussed §C.

B.1 Linear inverse problems

In this section, the class of functions is fixed to the class of linear functions across all problems, i.e. $\mathcal{F} = \left\{f : \boldsymbol{x} \mapsto \boldsymbol{w}^T\boldsymbol{x} \,|\, \boldsymbol{w} \in \mathbb{R}^d\right\}$; what varies across the problems is the distribution of $\boldsymbol{w}$. Problems in this section are instances of linear inverse problems. Linear inverse problems are classic problems arising in diverse applications in engineering, science, and medicine. In these problems, one wants to estimate model parameters from a few linear measurements. Often these measurements are expensive and can be fewer in number than the number of parameters ($p < d$). Such seemingly



(a)



(b)

Figure 4: Visualizing recovered weights for sparse and sign vector regression for one of the examples in the test set.

ill-posed problems can still be solved if there are structural constraints satisfied by the parameters. These constraints can take many forms from being sparse to having a low-rank structure. The sparse case was addressed by a famous convex programming approach Candes and Tao [2005], Donoho [2006] also known as compressed sensing. This was greatly generalized in later work to apply to many more types of inverse problems; see Chandrasekaran et al. [2012]. In this section, we will show that transformers can solve many inverse problems in context—in fact all problems that we tried. The problem-specific structural constraints are encoded in the prior for $\boldsymbol{w}$.

B.1.1 Function classes and baselines

Dense Regression ($\mathcal{F}_{\text{DR}}$). This represents the simplest case of linear regression as studied in Garg et al. [2022], Akyürek et al. [2022], von Oswald et al. [2022], where the prior on $\boldsymbol{w}$ is the standard Gaussian i.e. $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I})$. We are particularly interested in the underdetermined region i.e. $k < d$. Gaussian prior enables explicit PME computation: both PME and maximum a posteriori (MAP) solution agree and are equal to the minimum L_2-norm solution of the equations forming the training examples, i.e. $\min_{\boldsymbol{w}} \|\boldsymbol{w}\|_2$ s.t. $\boldsymbol{w}^T \boldsymbol{x}_i = f(\boldsymbol{x}_i), \forall i \leq k$. Standard Ordinary Least Squares (OLS) solvers return the minimum L_2-norm solution, and thus PME and MAP too, in the underdetermined region, i.e. $k < d$.

Skewed-Covariance Regression ($\mathcal{F}_{\text{Skew-DR}}$). This setup is similar to dense-regression, except that we assume the following prior on weight vector: $\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$ is the covariance matrix with eigenvalues proportional to $1/i^2$, where $i \in [1, d]$. For this prior on $\boldsymbol{w}$, we can use the same (but more general) argument for dense regression above to obtain the PME and MAP which will be equal and can be obtained by minimizing $\boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w}$ w.r.t to the constraints $\boldsymbol{w}^T \boldsymbol{x}_i = f(\boldsymbol{x}_i)$. This setup was motivated by Garg et al. [2022], where it was used to sample $\boldsymbol{x}_i$ values for out-of-distribution (OOD) evaluation, but not as a prior on $\boldsymbol{w}$.

Sparse Regression ($\mathcal{F}_{\text{SR}}$). In sparse regression, we assume $\boldsymbol{w}$ to be an s-sparse vector in $\mathbb{R}^d$ i.e. out of its d components only s are non-zero. Following Garg et al. [2022], to sample $\boldsymbol{w}$ for constructing prompts P, we first sample $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I})$ and then randomly set its $d - s$ components as 0. We consider $s = 3$ throughout our experiments. While computing the PME appears to be intractable here, the MAP solution can be estimated using Lasso by assuming a Laplacian prior on $\boldsymbol{w}$ Tibshirani [1996].

Sign-Vector Regression ($\mathcal{F}_{\text{SVR}}$). Here, we assume $\boldsymbol{w}$ to be a sign vector in $\{-1, +1\}^d$. For constructing prompts P, we sample d independent Bernoulli random variables b_j with a mean of 0.5 and obtain $\boldsymbol{w} = [2b_1 - 1, \cdots, 2b_d - 1]^T$. While computing the exact PME remains intractable in this case as well, the optimal solution for $k > d/2$ can be obtained by minimizing the L_∞-norm $\|\boldsymbol{w}\|_\infty$ w.r.t. the constraints specified by the input-output examples ($\boldsymbol{w}^T \boldsymbol{x}_i = f(\boldsymbol{x}_i)$) Mangasarian and Recht [2011].

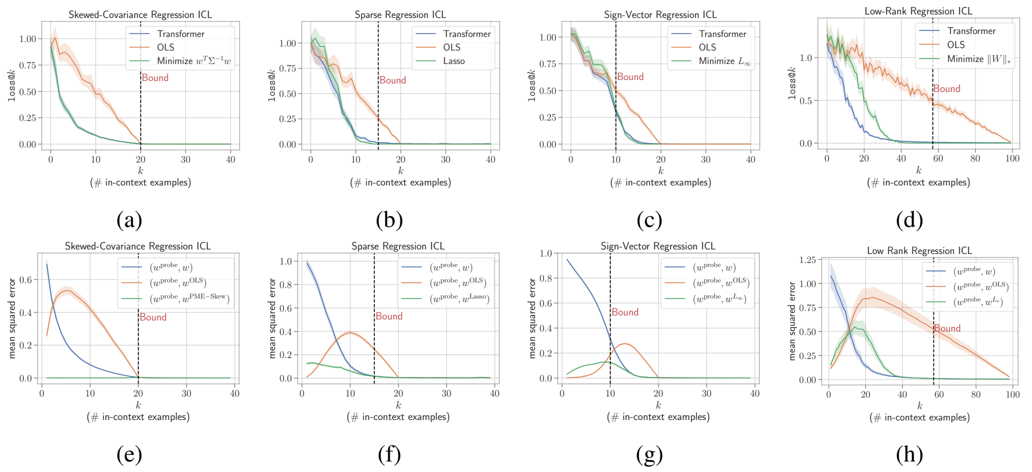


Figure 5: Comparing ICL in transformers for different linear functions with the relevant baselines. **Top**: loss@k values for transformers and baselines on skewed covariance, sparse, sign-vector, and low-rank regression tasks. **Bottom**: Comparing the errors between the implicit weights recovered from transformers w^{probe} with the ground truth weights w and weights computed by different baselines. $\boldsymbol{w}^{\text{PME-Skew}}$ denotes the weights obtained by minimizing $\boldsymbol{w}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{w}$ for the skewed covariance regression task.

Low-Rank Regression ($\mathcal{F}_{\textbf{LowRank-DR}}$). In this case, $\boldsymbol{w}$ is assumed to be a flattened version of a matrix $\boldsymbol{W} \in \mathbb{R}^{q\times q}$ ($d = q^2$) with a rank r, where $r \ll q$. A strong baseline, in this case, is to minimize the nuclear norm L_* of $\boldsymbol{W}$, i.e. $\|\boldsymbol{W}\|_*$ subject to constraints $\boldsymbol{w}^T\boldsymbol{x}_i = f(\boldsymbol{x}_i)$. To sample the rank-$r$ matrix $\boldsymbol{W}$, we sample $\boldsymbol{A} \sim \mathcal{N}(0, 1)$, s.t. $\boldsymbol{A} \in \mathbb{R}^{q\times r}$ and independently a matrix $\boldsymbol{B}$ of the same shape and distribution, and set $\boldsymbol{W} = \boldsymbol{A}\boldsymbol{B}^T$.

Recovery bounds. For each function class above, there is a bound on the minimum number of in-context examples needed for the exact recovery of the solution vector $\boldsymbol{w}$. The bounds for sparse, sign-vector and low-rank regression are $2s\log(d/s) + 5s/4$, $d/2$, and $3r(2q - r)$ respectively Chandrasekaran et al. [2012].

B.1.2 Results

We train transformer-based models on the five tasks following §2.2. Each model is trained with $d = 20$ and $p = 40$, excluding Low-Rank Regression where we train with $d = 100$, $p = 114$, and $r = 1$. Figures 5b-5d compare the loss@k values on these tasks with different baselines. Additionally, we also extract the implied weights $\boldsymbol{w}^{\text{probe}}$ from the trained models when given a prompt P following Akyürek et al. [2022] by generating model's predictions $\{y'_i\}$ on the test inputs $\{\boldsymbol{x}'_i\}_{i=1}^{2d} \sim \mathcal{D}_{\mathcal{X}}$ and then solving the system of equations to recover $\boldsymbol{w}^{\text{probe}}$. We then compare the implied weights $\boldsymbol{w}^{\text{probe}}$ with the ground truth weights $\boldsymbol{w}$ as well as the weights extracted from different baselines to better understand the inductive biases exhibited by these models during in-context learning (Figures 5f-5h).

Since results for dense regression have been already covered in Akyürek et al. [2022], we do not repeat them here, but for completeness provide them in Figure 3. For skewed-covariance regression, we observe that the transformer follows the PME solution very closely both in terms of the loss@k values (Figure 5a) as well as the recovered weights for which the error between $\boldsymbol{w}^{\text{probe}}$ and $\boldsymbol{w}^{\text{PME}-\text{Skew}}$ (weights obtained by minimizing $\boldsymbol{w}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{w}$) is close to zero at all prompt lengths (Figure 5e). On all the remaining tasks as well, the models perform better than OLS and are able to solve the problem with $< d$ samples i.e. underdetermined region meaning that they are able to understand the structure of the problem. The error curves of transformers for the tasks align closely with the errors of Lasso (Figure 5b), L_∞ minimization (Figure 5c), and L_* minimization (Figure 5d) baselines for the respective tasks. Interestingly for low-rank regression transformer actually performs better. Though, due to the larger problem dimension, ($d = 100$) in this, it requires a bigger model: 24 layers, 16 heads, and 512 hidden size. In Figures 5f, 5g, and 5h, we observe that at small prompt lengths w^{probe} and w^{OLS} are close. We conjecture that this might be attributed to both w^{probe} and w^{OLS} being close to 0 for small prompt lengths (Figure 4). Prior distributions for all three tasks are centrally-symmetric, hence, at small prompt lengths when the posterior is likely to be close to the prior, the PME is close to the mean of the prior which is 0. At larger prompt lengths transformers start to agree with $\boldsymbol{w}^{\text{Lasso}}$, $\boldsymbol{w}^{L_\infty}$, and $\boldsymbol{w}^{L_*}$. This is consistent with the transformer following PME, assuming $\boldsymbol{w}^{\text{Lasso}}$, $\boldsymbol{w}^{L_\infty}$, and $\boldsymbol{w}^{L_*}$ are close to PME—we leave it to future work to determine whether this is true (note that for sparse regression Lasso approximates the MAP estimate which should approach the PME solution as more data is observed). The recovered weights $\boldsymbol{w}^{\text{probe}}$ also agree with $\boldsymbol{w}^{\text{Lasso}}$, $\boldsymbol{w}^{L_\infty}$, and $\boldsymbol{w}^{L_*}$ for their respective tasks after sufficient in-context examples are provided.

B.2 Non-linear problems

Moving beyond linear functions, we now study how well transformers can in-context learn function classes with more complex relationships between the input and output, and if their behavior resembles the ideal learner i.e. the PME. Particularly, we consider the function classes of the form $\mathcal{F}_\Phi = \{f(\cdot;\Phi)|f(\boldsymbol{x};\Phi) = \boldsymbol{w}^T\Phi(\boldsymbol{x}), \boldsymbol{w} \in \mathbb{R}^\Delta\}$, where $\Phi : \mathbb{R}^d \to \mathbb{R}^\Delta$ maps the input vector $\boldsymbol{x}$ to an alternate feature representation. This corresponds to learning the mapping $\Phi(\boldsymbol{x})$ and then performing linear regression on top of it. Under the assumption of a standard Gaussian prior on $\boldsymbol{w}$, the PME for the dense regression can be easily extended for $\mathcal{F}_\Phi$: $\min_{\boldsymbol{w}} \|\boldsymbol{w}\|_2$, s.t. $\boldsymbol{w}^T\Phi(\boldsymbol{x}_i) = f(\boldsymbol{x}_i)$ for $i \in \{1, \cdots, p\}$.

B.2.1 Fourier Series

A Fourier series is an expansion of a periodic function into a sum of trigonometric functions. One can represent the Fourier series using the sine-cosine form given by:

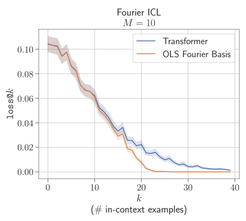
$$f(x) = a_0 + \sum_{n=1}^{N} a_n \cos(n\pi x/L) + \sum_{n=1}^{N} b_n \sin(n\pi x/L)$$

where, $x \in [-L, L]$, and a_0, a_n's and b_n's are known as Fourier coefficients and $\cos n\pi/L$ and $\sin n\pi/L$ define the frequency n components. We can define the function class $\mathcal{F}_{\Phi_N}^{\text{fourier}}$ by considering Φ as the Fourier feature map i.e. $\Phi_N(x) = [1, \cos(\pi x/L), \cdots, \cos(N\pi x/L), \sin(\pi x/L), \cdots, \sin(N\pi x/L)]^T$, and $\boldsymbol{w}$ as Fourier coefficients: $\boldsymbol{w} = [a_0, a_1, \cdots, a_N, b_1, \cdots, b_N]$. Hence, $\Phi_N(x) \in \mathbb{R}^d$ and $\boldsymbol{w} \in \mathbb{R}^d$, where $d = 2N + 1$.

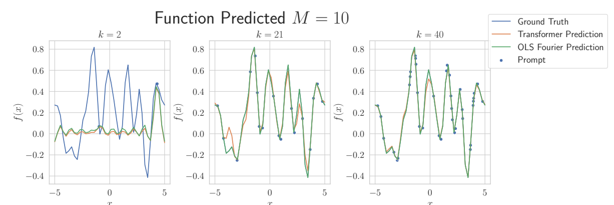
For training transformers to in-context-learn $\mathcal{F}_{\Phi_N}^{\text{fourier}}$, we fix a value of N and sample functions $f \in \mathcal{F}_{\Phi_N}^{\text{fourier}}$ by sampling the Fourier coefficients from the standard normal distribution i.e. $\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}_d, \boldsymbol{I})$. We consider the inputs to be scalars, i.e. $x_i \in [-L, L]$ and we sample them i.i.d. from the uniform distribution on the domain: $x_i \sim \mathcal{U}(-L, L)$. In all of our experiments, we consider $N = 10$ and $L = 5$. At test time we evaluate on $\mathcal{F}_{\Phi_M}^{\text{fourier}}$ for $M \in [1, 10]$, i.e. during evaluation we also prompt the model with functions with different maximum frequency as seen during training. As a baseline, we use OLS on the Fourier features (denoted as OLS Fourier Basis) which will be equivalent to the PME.

Measuring inductive biases. Once we train a transformer-based model to in-context learn $\mathcal{F}_{\Phi_N}^{\text{fourier}}$, how can we investigate the inductive biases that the model learns to solve the problem? We would like to answer questions such as, when prompted with k input-output examples what are the prominent frequencies in the function simulated by the model, or, how do these exhibited frequencies change as we change the value of k? We start by sampling in-context examples $(x_1, f(x_1), \cdots x_k, f(x_k))$, and given the context obtain the model's predictions on a set of m test inputs $\{x'_i\}_{i=1}^m$, i.e. $y'_i = M_\theta\left((x_1, f(x_1), \cdots x_k, f(x_k), x'_i)\right)$. We can then perform Discrete Fourier Transform (DFT) on $\{y'_1, \cdots, y'_m\}$ to obtain the Fourier coefficients of the function output by M, which we can analyze to understand the dominant frequencies.

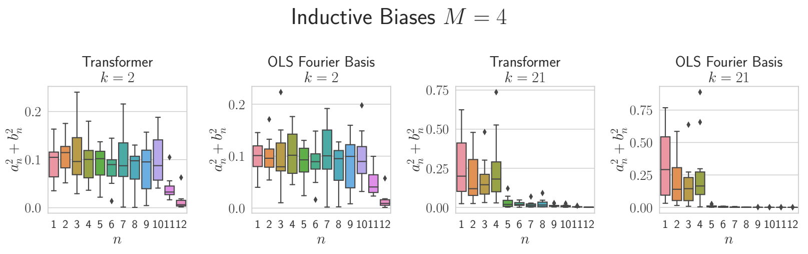
Results. The results of our experiments concerning the Fourier series are provided in Figure 6. Transformers obtain `loss@`k values close to the OLS Fourier Basis baseline (Figure 6a) indicating at least for the smaller prompt lengths the model is able to simulate the behavior of the ideal predictor (PME). These plots use 12-layer transformers to obtain results, but we also investigate if bigger models help. Figure 7 plots bigger models with 18 and 21 layers where the agreement with PME is much better. Since the inputs x_i, in this case, are scalars, we can visualize the functions learned in context by transformers. We show one such example for a randomly selected function $f \sim \mathcal{F}_{\Phi_M}^{\text{fourier}}$



(a)



(b)



(c)

Figure 6: Effectiveness of ICL in transformers for Fourier series family of functions. **Top left**: `loss@`k values for transformer and OLS Fourier Basis baseline. **Top Right**: Visualizing the functions simulated by the transformer and the OLS Fourier Basis. **Bottom**: Measuring the frequencies of the simulated function by the transformer and the baseline.

for prompting the model in Figure 6b. As can be observed, the functions predicted by both the transformer and baseline have a close alignment, and both approach the ground truth function f as more examples are provided. Finally, we visualize the distribution of the frequencies for the predicted functions in Figure 6c. For a value of M, we sample 10 different functions and provide k in-context examples to the model to extract the frequencies of the predicted functions using the DFT method. As can be observed, when provided with fewer in-context examples ($k = 2$) both Transformer and the baseline predict functions with all the 10 frequencies (indicated by the values of $a_n^2 + b_n^2$ in a similar range for $n \in [1, 10]$), but as more examples are provided they begin to recognize the gold maximum frequency (i.e. $M = 4$). The function visualizations for the transformer and Fourier OLS baseline for different combinations of M and k are provided in Figure 9. We have observations consistent with Figure 6b, where the function outputs of the transformer and the baseline align closely. Similarly, in Figure 8, we present the distribution of frequencies in the predicted functions for the two methods and again observe consistent findings. This suggests that the transformers are following the Bayesian predictor and are not biased towards smaller frequencies.

B.2.2 Random Fourier Features

Mapping input data to random low-dimensional features has been shown to be effective to approximate large-scale kernels Rahimi and Recht [2007]. In this section, we are particularly interested in Random Fourier Features (RFF) which can be shown to approximate the Radial Basis Function kernel and are given as:

$$\Phi_D(\boldsymbol{x}) = \sqrt{\frac{2}{D}}[\cos(\boldsymbol{\omega}_1^T\boldsymbol{x} + \delta_1), \cdots, \cos(\boldsymbol{\omega}_D^T\boldsymbol{x} + \delta_D)]^T$$

where $\boldsymbol{\omega}_i \in \mathbb{R}^d$ and $\delta_i \in \mathbb{R}\ \forall i \in [1, D]$, such that $\Phi_D : \mathbb{R}^d \to \mathbb{R}^D$. Both $\boldsymbol{\omega}_i$ and δ are sampled randomly, such that $\boldsymbol{\omega}_i \in \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$ and $\delta_i \in (0, 2\pi)$. We can then define the function family $\mathcal{F}_{\Phi_D}^{\text{RFF}}$ as linear functions over the random fourier features i.e. $f = \boldsymbol{w}^T\Phi_D(\boldsymbol{x})$ such that $f \sim \mathcal{F}_{\Phi_D}^{\text{RFF}}$. While training the transformer on this function class, we sample $\boldsymbol{\omega}_i$'s and δ_i's once and keep them fixed throughout the training. As a baseline, we use OLS over $(\Phi_D(x), y)$ pairs which will give the PME for the problem (denote this as RFF-OLS).

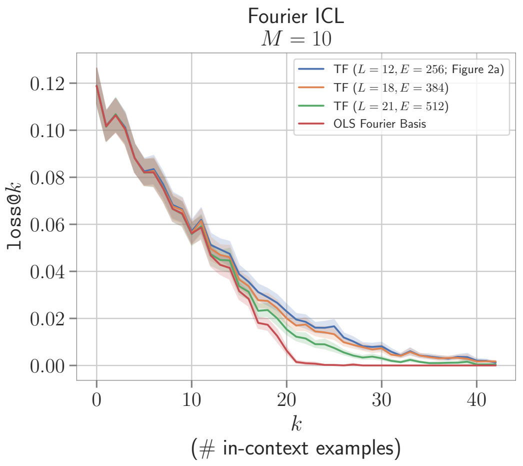


Figure 7: **Bigger models achieve better results on the Fourier Series task.** Plotting the squared error (averaged over 1280 prompts) for bigger transformer (TF) models trained for 500k steps on the Fourier Series task. Training setup is the same as used for the model plotted in Figure 2a (Section 3.2.1), which is also plotted here for comparison. L and E denote the number of layers and embedding size for TF models respectively.

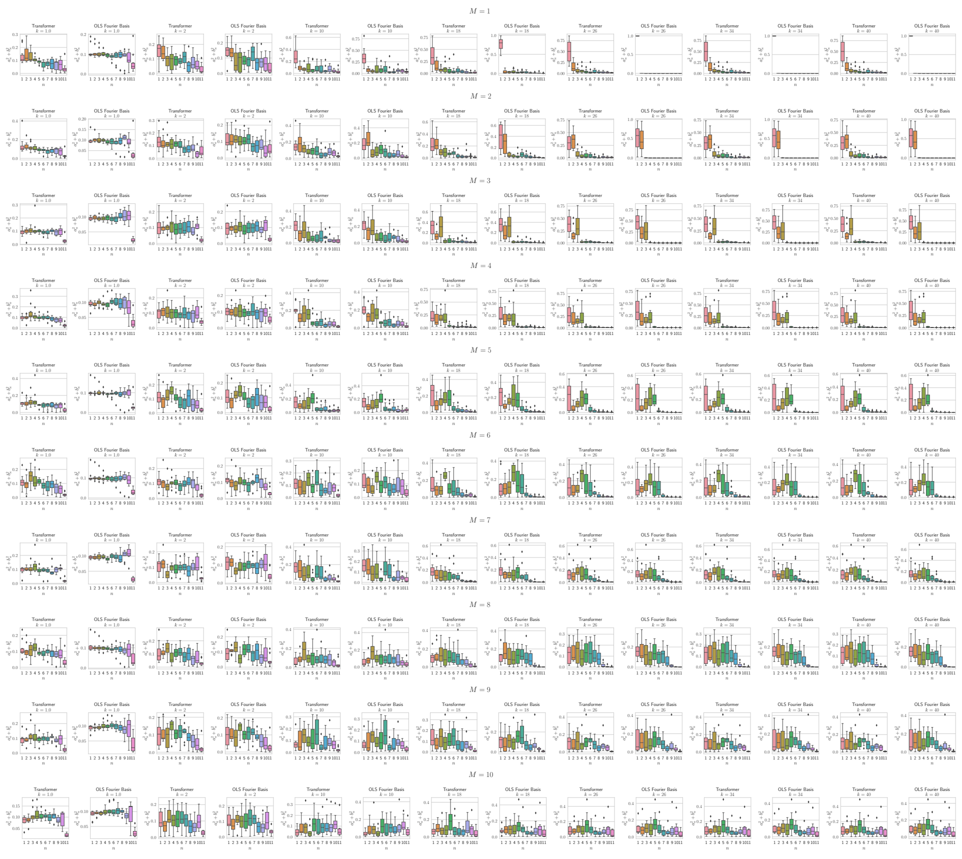


Figure 8: Measuring the frequencies of the simulated function by the transformer and the baseline for different values of M (maximum frequency) and k (number of in-context examples)

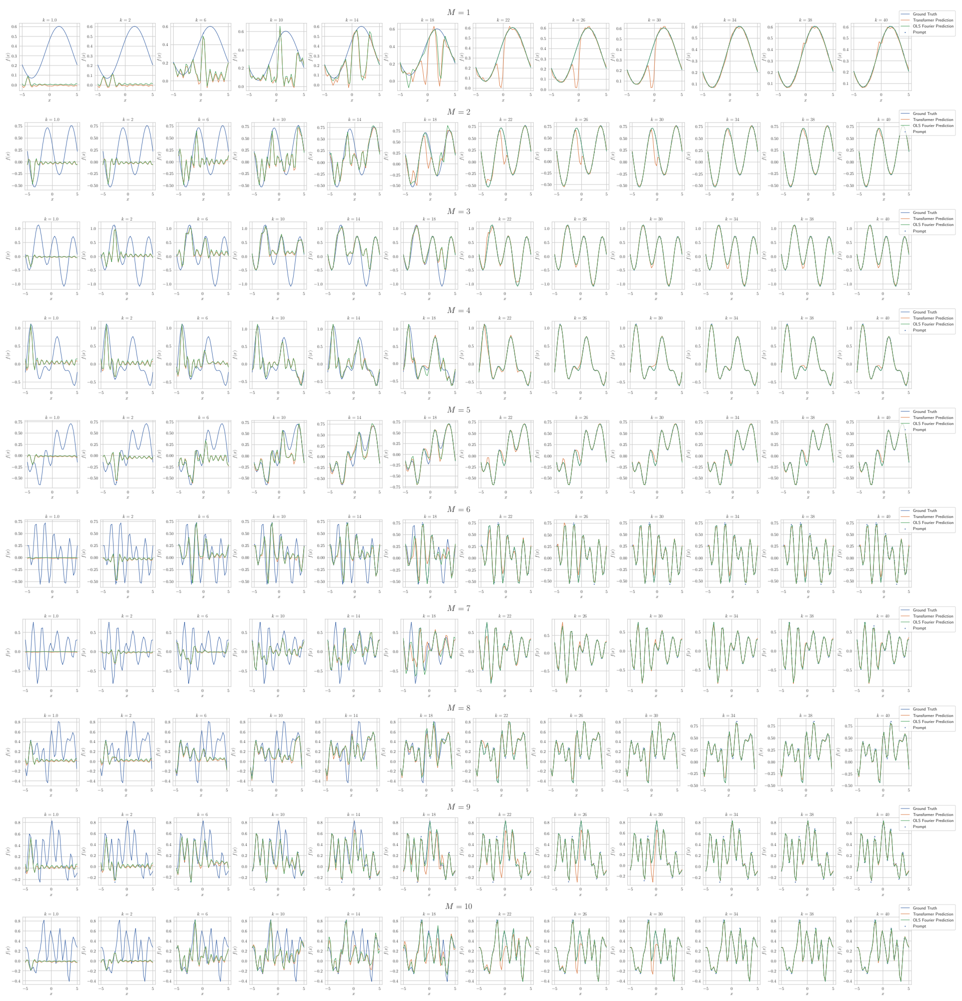


Figure 9: Visualizing the functions simulated by the transformer and the OLS Fourier Basis, for different values of M (maximum frequency) and k (number of in-context examples)

Results. For this particular family, we observed mixed results for transformers, i.e. they fail to generalize to functions of the family when the complexity of the problem is high. The complexity of this function class is dictated by the length of the $\boldsymbol{\omega}_i$ vectors (and the inputs $\boldsymbol{x}$) i.e. d and the number of random features D. We plot the `loss@`k values for transformer models trained on $\mathcal{F}_{\Phi_D}^{\text{RFF}}$ for different values of d and D in Figure 10. As can be observed, the complexity of the problem for the transformers is primarily governed by d, where they are able to solve the tasks for even large values of D, however, while they perform well for smaller values of d ($d = 1$ and $d = 4$), for $d = 10$, they perform much worse compared to the RFF-OLS baseline and the `loss@`k doesn't improve much once ~ 15 in-context examples are provided.

B.2.3 Degree-2 Monomial Basis Regression

Defined in §B.2.1, the Fourier Series function class can be viewed as linear regression over the Fourier basis consisting of sinusoidal functions. Similarly, we define a function class $\mathcal{F}_{\Phi_M}^{\text{mon(2)}}$ with the basis formed by degree-2 monomials for any d-dimensional input vector $\boldsymbol{x}$.

Using the notation introduced in B.1.1 the basis for $\mathcal{F}_{\Phi_M}^{\text{mon(2)}}$ is defined as $\Phi_M(\boldsymbol{x}) = \{x_i x_j \mid 1 \leq i, j \leq d\}$. Each function $f \in \mathcal{F}_{\Phi_M}^{\text{mon(2)}}$ is a linear combination of basis and $\boldsymbol{w}$ i.e. $f(\boldsymbol{x}) = \boldsymbol{w}^T \Phi_M(\boldsymbol{x})$, where $\boldsymbol{w}$ is a $|\Phi_M|$-dimensional vector sampled from standard normal distribution.

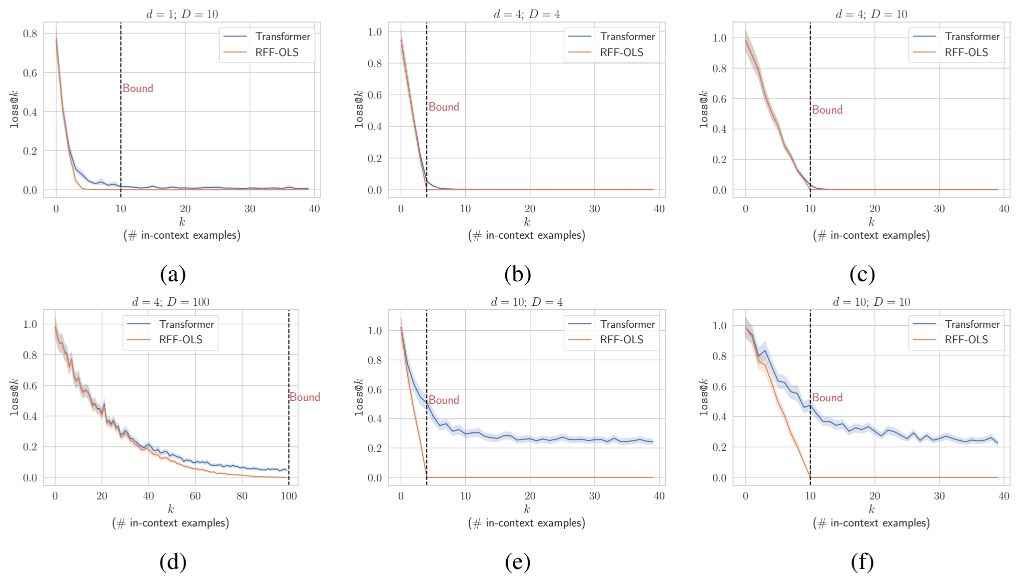


Figure 10: Comparing transformers performance on RFF function family ($\mathcal{F}_{\Phi_D}^{\text{RFF}}$) with the RFF-OLS baseline for different values of d and D.

For experimentation, we define a sub-family $\mathcal{F}_{\mathcal{S}}^{\text{mon}(2)}$ under $\mathcal{F}_{\Phi_M}^{\text{mon}(2)}$ by choosing a proper subset $\mathcal{S} \subset \Phi_M$ and linearly combining the terms in $\mathcal{S}$ to form f. This is equivalent to explicitly setting coefficients w_i of terms in $\Phi_M - \mathcal{S}$ to 0. We experiment with $d = 20$, with the prompt length $p = 290$ and $|\mathcal{S}| = 20$. We do not use curriculum ($d, p, |\mathcal{S}|$ are fixed for the entire duration of the training run).

Baselines. We use OLS fitted to the following bases as baselines: $\mathcal{S}$ basis (OLS$_{\mathcal{S}}$), all degree-2 monomials i.e., Φ_M basis (OLS$_{\Phi_M}$), and to a basis of all polynomial features up to degree-2 (OLS$_{\text{poly.(2)}}$). We also compare Lasso ($\alpha = 0.01$) fitted to all degree-2 monomials i.e., Φ_M basis (Lasso$_{\Phi_M}$) as a baseline.

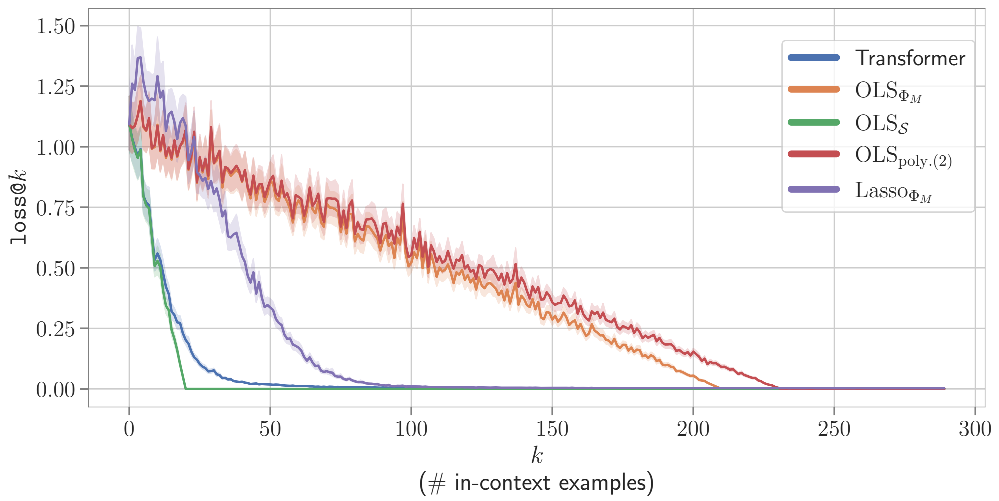


Figure 11: **In-Distribution evaluation results on $\mathcal{F}_{\mathcal{S}}^{\text{mon(2)}}$ sub-family of degree-2 monomial basis regression.** Evaluation of transformer on prompts generated using the same $\mathcal{S}$ used during training.

Results. In Figure 11, we show the In-Distribution (ID) evaluation results for the $\mathcal{F}_{\mathcal{S}}^{\text{mon}(2)}$ experiments. Here, the test prompts contain functions formed by $\mathcal{S}$ (the same basis used during training). We observe that Transformers closely follow OLS$_{\mathcal{S}}$. The increasing order of performance (decreasing `loss@`k for $k \geq |\mathcal{S}|$) of different solvers is: OLS$_{\text{poly.(2)}}$ $\leq$ OLS$_{\Phi_M}$ $<$ Lasso$_{\Phi_M}$ $<$ Transformers $<$

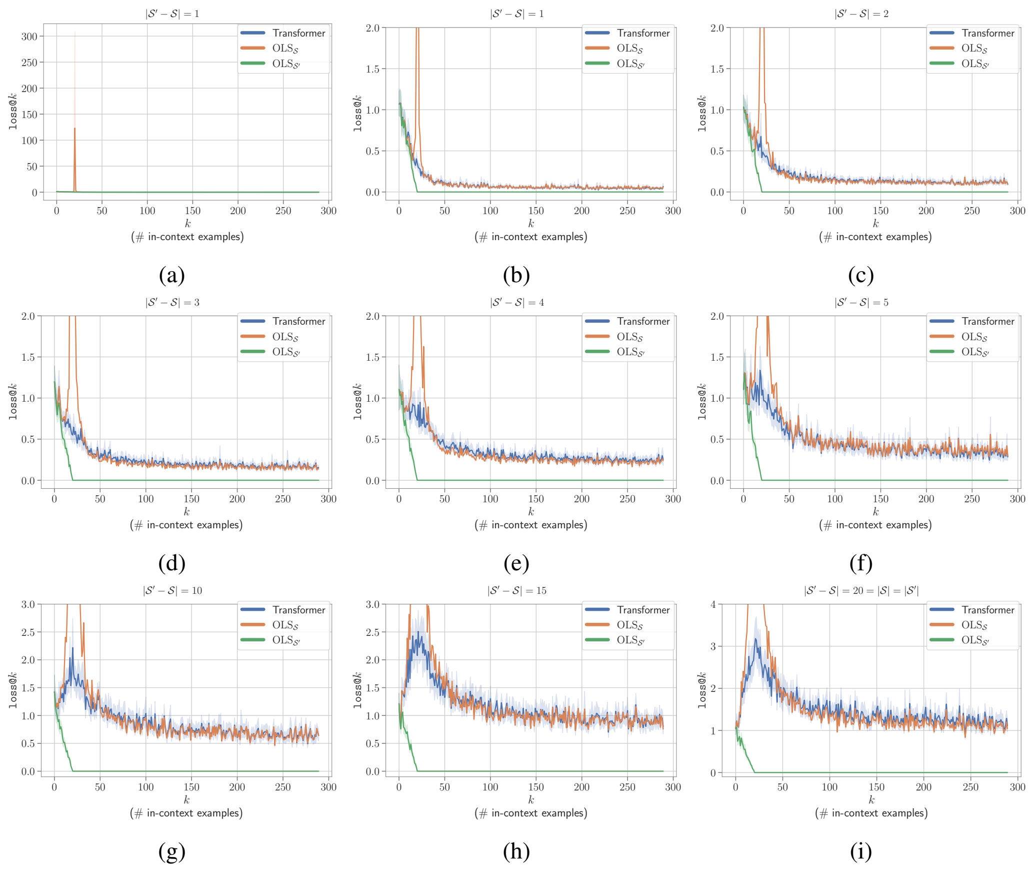


Figure 12: **Out-of-Distribution evaluation results on $\mathcal{F}_{\mathcal{S}}^{\text{mon(2)}}$ sub-family of degree-2 monomial basis regression.** Evaluation of transformer trained on prompts generated using $\mathcal{S}'$, where $\mathcal{S}'$ contains n degree-2 monomials not present in $\mathcal{S}$ that was used during training. We show results for different values of n.

$\text{OLS}_{\mathcal{S}}$. Transformer's squared error takes a little longer than $\text{OLS}_{\mathcal{S}}$ to converge. Lasso_{Φ_M} is able to take the advantage of sparsity of the problem and is hence better than both OLS_{Φ_M} and $\text{OLS}_{\text{poly.(2)}}$, which respectively converge at $k = 210$ and $k = 231$[4]. We also conduct an Out-of-Distribution (OOD) evaluation for $\mathcal{F}_{\mathcal{S}}^{\text{mon(2)}}$, whose results are shown in Figure 12. Here, we generate prompts from a basis $\mathcal{S}' \subset \Phi_M$ of the same size as $\mathcal{S}$ but differing from $\mathcal{S}$ in n degree-2 terms, i.e. $|\mathcal{S}' - \mathcal{S}| = n$. We show the results for different values of n. Figure 12a shows the $\text{OLS}_{\mathcal{S}}$ undergoes a steep rise in errors momentarily at $k = |\mathcal{S}|$ (double descent). Figure 12b zooms into the lower error region of Figure 12a where we notice that Transformer mimics $\text{OLS}_{\mathcal{S}}$, while $\text{OLS}_{\mathcal{S}'}$ is the best-performing baseline (since it fits to the $\mathcal{S}'$ basis used to construct the prompts). Transformer does not undergo double descent (for $n = 1$) and is hence momentarily better than $\text{OLS}_{\mathcal{S}}$ at $k = |\mathcal{S}|$. Similar plots are shown for $n \in \{2, 3, 4, 5, 10, 15, 20\}$. As n increases, the height of $\text{OLS}_{\mathcal{S}}$ peak increases and the Transformer also starts to have a rise in errors at $k = |\mathcal{S}|$. For $n = 20$, $\mathcal{S}'$ and $\mathcal{S}$ have nothing in common, and Transformer still follows $\text{OLS}_{\mathcal{S}}$ (OLS fitted to the training basis $\mathcal{S}$). As mentioned under §B.2, when the prior on weights $\boldsymbol{w}$ is Gaussian, the PME is the minimum L_2-norm solution. For $\mathcal{F}_{\mathcal{S}}^{\text{mon(2)}}$, that solution is given by $\text{OLS}_{\mathcal{S}}$. Therefore, the results suggest that the transformer is computing PME. In summary, transformers closely follow $\text{OLS}_{\mathcal{S}}$ in this set-up, and more so on the OOD data, where they even surpass $\text{OLS}_{\mathcal{S}}$'s performance when it experiences double descent.

[4]210 and 231 are the sizes of the bases to which OLS_{Φ_M} and $\text{OLS}_{\text{poly.(2)}}$ are fitted. Hence, they converge right when the problem becomes determined in their respective bases.

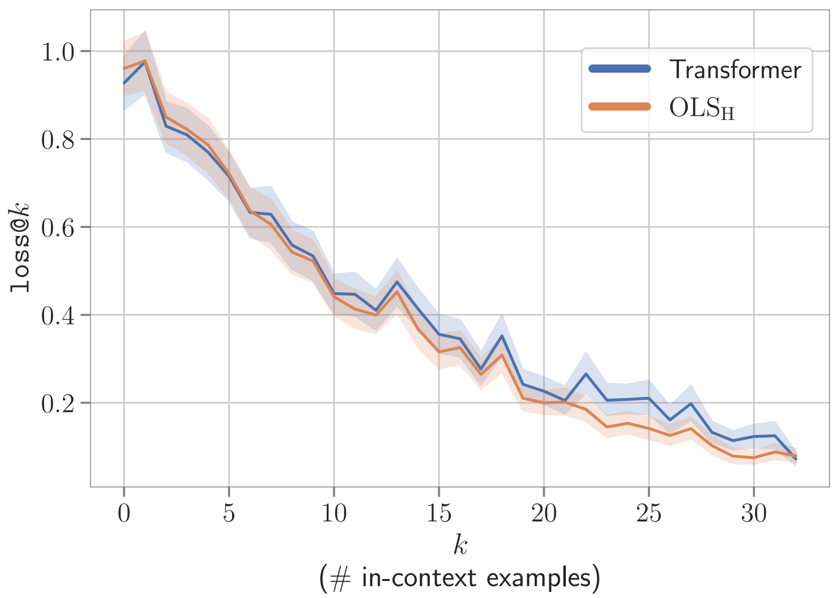


Figure 13: Evaluating Transformer trained on Haar Wavelet Basis Regression task ($\mathcal{F}_{\Phi_H}^{\text{Haar}}$).

B.2.4 Haar Wavelet Basis Regression

Similar to Fourier Series and Degree-2 Monomial Basis Regression, we also define another non-linear regression function family ($\mathcal{F}_{\Phi_H}^{\text{Haar}}$) using a different basis, Φ_H, called the Haar wavelet basis. Φ_H is defined on the interval $[0, 1]$ and is given by:

$$\Phi_H(x) = \{x \in [0,1] \mapsto \psi_{n,k}(x) : n \in \mathbb{N} \cup \{0\}, 0 \leq k < 2^n\} \cup \{\mathbf{1}\},$$
$$\psi_{n,k}(x) = 2^{n/2}\psi(2^n x - k), x \in [0,1],$$
$$\psi(x) = \begin{cases} 1 & 0 \leq x < \frac{1}{2}, \\ -1 & \frac{1}{2} \leq x < 1, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbf{1}$ is the constant function which is 1 everywhere on $[0, 1]$. To define f, we sample $\boldsymbol{w}$ from $\mathcal{N}(0, 1)$ and compute its dot product with the basis, i.e. $\boldsymbol{w}^T \Phi_H(\cdot)$. We construct the prompt P by evaluating f at different values of $\boldsymbol{x} \sim \mathcal{U}(0, 1)$. The Transformer model is then trained on these prompts P.

We use $d = 1$ and $p = 32$, both of which are fixed throughout the training run, i.e. we do not use curriculum. We only consider the basis terms corresponding to $n \in \{0, 1, 2, 3\}$. The baseline used is OLS on Haar Wavelet Basis features (OLS_H). Note that for the model used throughout the paper (§2.2), at $k = 32$ the `loss@`k value is 0.18, while for a bigger model and OLS_H it is 0.07. Therefore, for this task we report the results for the bigger model which has 24 layers, 16 heads and 512 hidden size.

Results. In Figure 13, we observe that Transformer very closely mimics the errors of OLS_H (i.e. OLS fitted to the Haar Wavelet Basis) and converged to OLS_H at $k = 32$. Since the prior on the weights $\boldsymbol{w}$ is Gaussian, OLS_H is the PME. Hence, Transformer's performance on this task also suggests that it is simulating PME.

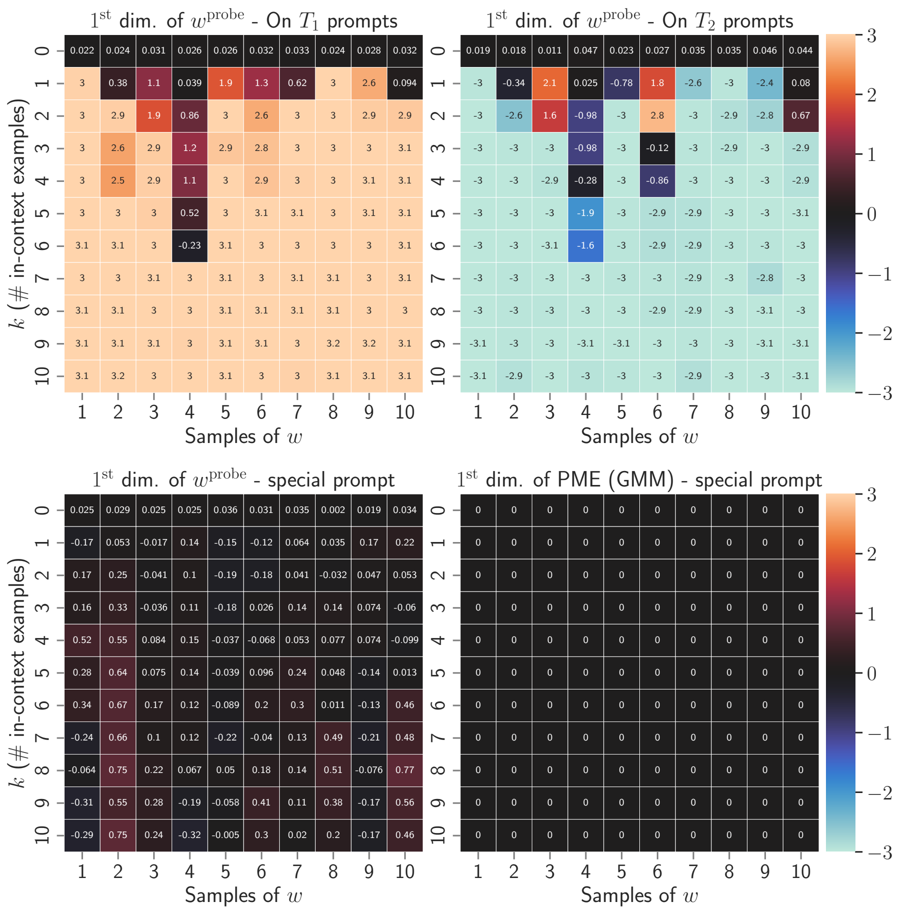


Figure 14: **Transformers simulate PME when trained on dense regression task-mixture ($d = 10, p = 10, \alpha_1 = \alpha_2 = \frac{1}{2}$) with weights having a mixture of Gaussian prior (GMM).** *(top)*: 1^{st} dimension of Transformer's probed weights across the prompt length. *(bottom)*: 1^{st} dimension of Transformer's probed weights and PME (GMM) across the prompt length for a specially constructed prompt.

C Detailed Experiments for HMICL setup

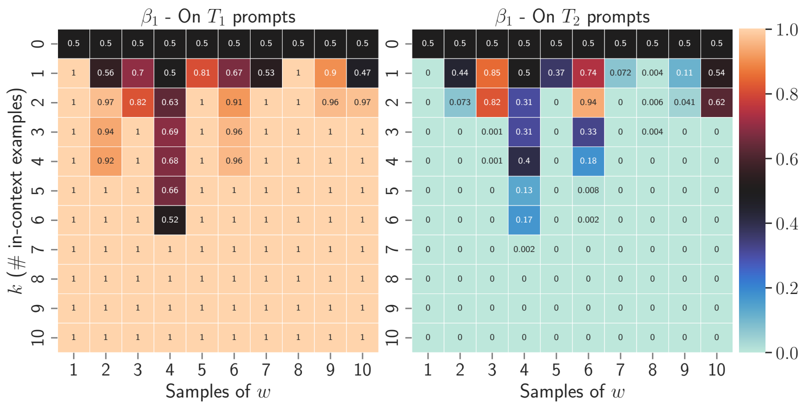
C.1 Gaussian Mixture Models (GMMs)

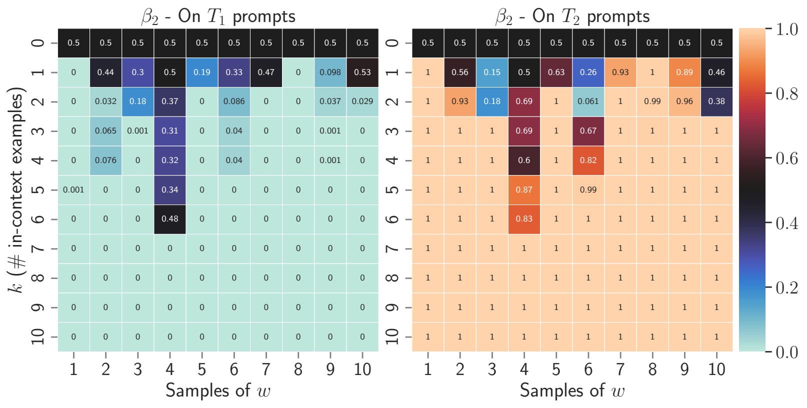
Here we discuss some details regarding §3.1 and more results on GMMs. We start with a description of how we calculate PMEs for this setup.

Computation of PMEs. As mentioned in §A.1 and §B.2, we can compute the individual PMEs for components T_1 and T_2 by minimizing the L_2 distance between the hyperplane induced by the prompt constraints and the mean of the Gaussian distribution. In particular, to compute PME for each Gaussian component of the prior, we solve a system of linear equations defined by the prompt constraints ($\boldsymbol{w}_i^T x_i = y_i, \forall i \in \{1, 2, .., p\}$) in conjunction with an additional constraint for the first coordinate, i.e. $(\boldsymbol{w})_1 = +3$ (for $\mathcal{N}_d(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ or $\boldsymbol{w}_1 = -3$ (for $\mathcal{N}_d(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$). Given these individual PMEs, we calculate the PME of the mixture using Eq. 3.

Now we discuss more results for GMMs. First, we see the evolution of β's (from Eq. 3), PME (GMM), and Transformer's probed weights across the prompt length (Figures 15 and 16). Next, we see the results for the Transformer models trained on the mixture with unequal weights, i.e. $\alpha_1 \neq \alpha_2$ (Figure 17) and for the $p = 20$ model (Figure 18).

Agreement of weights between Transformer and PME(GMM). Figure 14 (top) shows the evolution of the first dimension of the Transformer weights, i.e. $(\boldsymbol{w}^{\text{probe}})_1$, with prompt length k. We


(a)


(b)

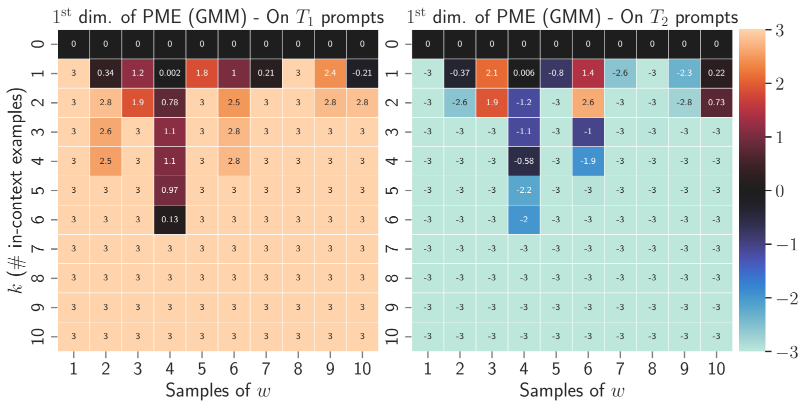
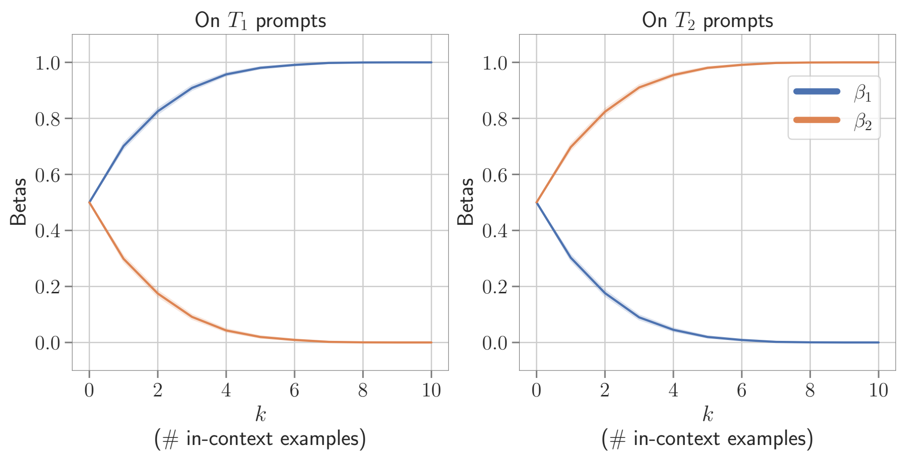
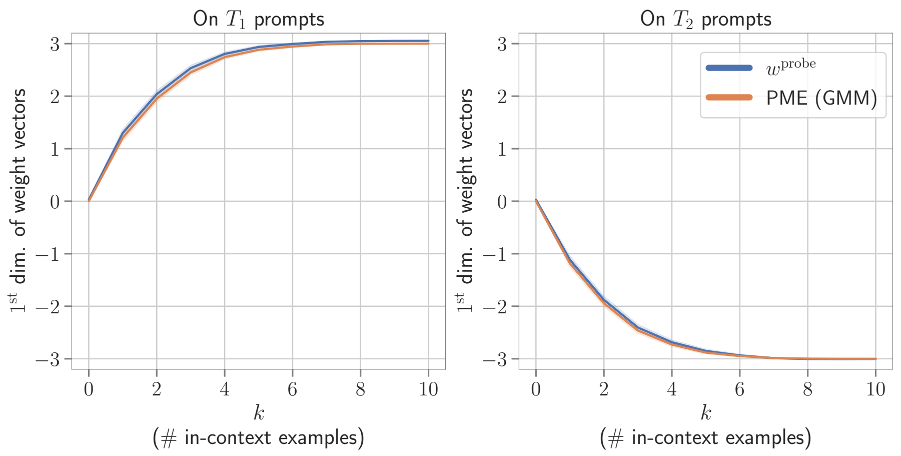

(c)

Figure 15: Evolution (as heatmaps) with prompt length (k) of β's and PME (GMM) appearing in Eq. 3 for the model trained with $d = 10, p = 10, \alpha_1 = \alpha_2 = \frac{1}{2}$. We show 10 different samples of $\boldsymbol{w}$ for each plot.



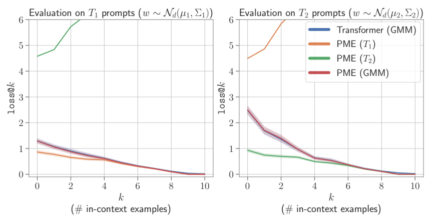
(a)



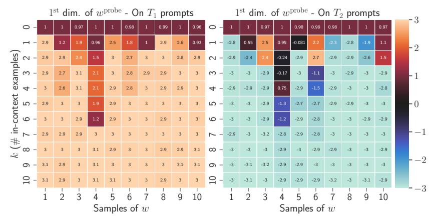
(b)

Figure 16: Evolution (as line plots) with prompt length (k) of β's, PME (GMM), and $\boldsymbol{w}^{\text{probe}}$ for the model trained with $d = 10, p = 10, \alpha_1 = \alpha_2 = \frac{1}{2}$. We show the values averaged over 1280 samples.

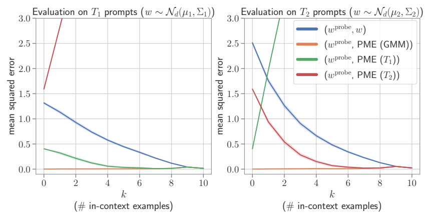
see that Transformer is simulating PME (GMM), which approaches PME (T_{prompt}) with increasing prompt length (k). Note that regardless of k, the first dimension of PME (T_i) is $(\boldsymbol{\mu}_i)_1$, the first dimension of the mean of the prior distribution T_i since the Gaussian has a fixed value in the first dimension. Note that PME (GMM) approaches PME (T_{prompt}) with increasing k (Eq. 3). Also note that in our setting, regardless of k the first dimension of PME (T_i) is $(\boldsymbol{\mu}_i)_1$, the first dimension of the mean of the prior distribution T_i, since T_i has a fixed value (i.e. zero variance) in the first dimension. Hence, if Transformer is simulating PME (GMM), the first dimension of Transformer's weights $(\boldsymbol{w}^{\text{probe}})_1$ must approach $(\boldsymbol{\mu}_1)_1$ (when $T_{\text{prompt}} = T_1$) and $(\boldsymbol{\mu}_2)_1$ (when $T_{\text{prompt}} = T_2$). This is exactly what we observe as $(\boldsymbol{w}^{\text{probe}})_1$ approaches $+3$ and -3 on T_1 and T_2 prompts respectively. At prompt length 0, in the absence of any information about the prompt, $(\boldsymbol{w}^{\text{probe}})_1 \approx 0$. This agrees with Eq. 3 since $0 = (\boldsymbol{\mu}_1)_1.\beta_1 + (\boldsymbol{\mu}_2)_1.\beta_2$, where $(\boldsymbol{\mu}_1)_1 = +3, (\boldsymbol{\mu}_2)_1 = -3, \beta_1 = \alpha_1 = 0.5$ and $\beta_2 = \alpha_2 = 0.5$ when prompt P is empty. The figure shows that with the increasing evidence from the prompt, the transformer shifts its weights to T_{prompt}'s weights as evidenced by the first coordinate changing from 0 to $+3$ or -3 based on the prompt. In **Figure 14 (bottom)**, we check the behavior of Transformer and PME (GMM) on specially constructed prompts P where $(\boldsymbol{x}_i)_1 = 0$ and $(\boldsymbol{x}_i)_{2:d} \sim \mathcal{N}(0, 1), \forall i \in \{1, \cdots, p\}$. For our setup, choosing such $\boldsymbol{x}_i$'s guarantees that no information about the distribution of $\boldsymbol{w}$ becomes known by observing P (since the only distinguishing dimension between T_1 and T_2 is the 1st dimension and that does not influence the prompt in this case as $(\boldsymbol{x}_i)_1 = 0$). We note that Transformer's weights are all ≈ 0 regardless of the prompt length, agreeing with the PME (GMM). Observing more examples from the prompt does not reveal any



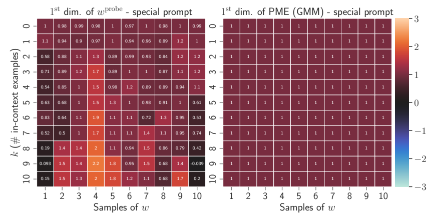
(a)



(c)

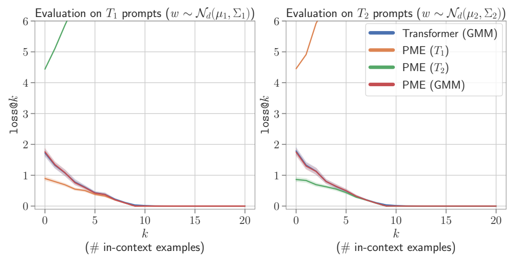


(b)

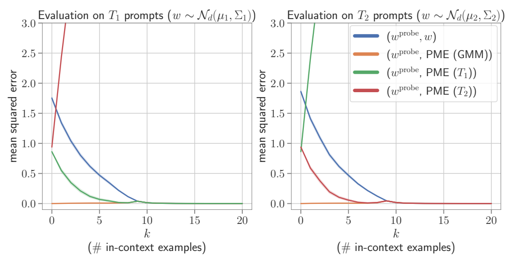


(d)

Figure 17: **Transformers simulate PME when trained on dense regression task-mixture ($d = 10, p = 10, \alpha_1 = \frac{2}{3}, \alpha_2 = \frac{1}{3}$) with weights having a mixture of Gaussian prior (GMM). (a)**: Comparing the performance of the Transformer with Posterior Mean Estimator (PME) of individual Gaussian components (PME (T_1) and PME (T_2)) and of the mixture PME (GMM). **(b)**: MSE between the probed weights of the Transformer and PMEs. **(c)**: 1st dimension of Transformer's probed weights across the prompt length. **(d)**: 1st dimension of Transformer's probed weights and PME (GMM) across the prompt length for a specially constructed prompt.



(a)



(b)

Figure 18: **Transformers simulate PME when trained on dense regression task-mixture ($d = 10, p = 20, \alpha_1 = \alpha 2 = \frac{1}{2}$) with weights having a mixture of Gaussian prior (GMM). Left:** Comparing the performance of the Transformer with Posterior Mean Estimator (PME) of individual Gaussian components (PME (T_1) and PME (T_2)) and of the mixture PME (GMM). **Right:** MSE between the probed weights of the Transformer and PMEs.

information about the underlying distribution of $\boldsymbol{w}$ in this case. All of this evidence strongly supports our hypothesis that Transformer behaves like the ideal learner and computes the Posterior Mean Estimate (PME).

Evolution of β's, PME (GMM), and $\boldsymbol{w}^{\text{probe}}$. Figure 15 plots the evolution of β's and 1st dimension of PME (GMM) for 10 different $\boldsymbol{w}$'s. The β's (Figures 15a and 15b) are 0.5 (equal to α's) at $k = 0$ (when no information is observed from the prompt). Gradually, as more examples are observed from the prompt, $\beta_{T_{\text{prompt}}}$ approaches 1, while $\beta_{T_{\text{other}}}$ approaches 0. This is responsible for PME (GMM) converging to PME (T_{prompt}) as seen in §3.1. The 1st dimension of PME (GMM) (Figure 15c) starts at 0 and converges to $+3$ or -3 depending on whether T_{prompt} is T_1 or T_2. Figure 16 shows the same evolution in the form of line plots where we see the average across 1280 samples of $\boldsymbol{w}$. In Figure 16a, $\beta_{T_{\text{prompt}}}$ approaches 1, while $\beta_{T_{\text{other}}}$ approaches 0 as noted earlier. Consequently, in Figure 16b, 1st dimension of PME (GMM) approaches $+3$ or -3 based on the prompt. The 1st dimension of Transformer's probed weights, i.e. $(\boldsymbol{w}^{\text{probe}})_1$ almost exactly mimics PME (GMM).

Unequal weight mixture with $\alpha_1 = \frac{2}{3}$ & $\alpha_2 = \frac{1}{3}$. Figure 17 shows the results for another model where $\alpha's$ are unequal ($d = 10, p = 10, \alpha_1 = \frac{2}{3}, \alpha_2 = \frac{1}{3}$). The observations made for Figure 1 in §3.1 still hold true, with some notable aspects: **(1)** The difference between prediction errors, i.e. `loss@`k (17a), of PME (GMM) and PME (T_1) is smaller than that of the uniform mixture ($\alpha_1 = \alpha_2 = \frac{1}{2}$) case, while the difference between prediction errors and weights of PME (GMM) and PME (T_2) is larger. This is because, at prompt length = 0, PME (GMM) is a weighted combination of component PMEs with α's as coefficients (Eq. 3). Since $\alpha_1 > \alpha_2$, PME (GMM) starts out as being closer to T_1 than T_2. Also, since the Transformer follows PME (GMM) throughout, its prediction errors also have similar differences (as PME (GMM)'s) with PMEs of both components T_1 and T_2. **(2)** Transformer's probed weights ($\boldsymbol{w}^{\text{probe}}$), which used to have the same MSE with PME (T_1) and PME (T_2) at $k = 0$, now give smaller MSE with PME (T_1) than PME (T_2) on prompts from both T_1 and T_2 (Figure 17b). This is a consequence of PME (GMM) starting out as being closer to T_1 than T_2 due to unequal mixture weights as discussed above. Since Transformer is simulating PME (GMM), $\boldsymbol{w}^{\text{probe}}$ is also closer to PME (T_1) than PME (T_2) at $k = 0$ regardless of which component (T_1 or T_2) the prompts come from. Due to $\boldsymbol{w}^{\text{probe}}$ mimicking T_1 more than T_2 we also observe in Figure 17b that $\boldsymbol{w}^{\text{probe}}$ gives smaller MSE with $\boldsymbol{w}$ (ground truth) when T_{prompt} = T_1 compared to when T_{prompt} = T_2. **(3)** The 1st dimension of Transformer's weights ($(\boldsymbol{w}^{\text{probe}})_1$) and PME (GMM) is 1 instead of 0 when the prompt is either empty (17c) or lacks information regarding the distribution of $\boldsymbol{w}$ (17d). It happens because $(\boldsymbol{w}^{\text{probe}})_1 \approx$ 1st dimension of PME (GMM) $= (\boldsymbol{\mu}_1)_1.\beta_1 + (\boldsymbol{\mu}_2)_1.\beta_2 = (+3)(\frac{2}{3}) + (-3)(\frac{1}{3}) = 1$. Note that $\beta_1 = \alpha_1 = \frac{2}{3}$ and $\beta_2 = \alpha_2 = \frac{1}{3}$ when prompt P is empty at $k = 0$ (Eq. 3). When P is inconclusive of $\boldsymbol{w}$, $\beta_1 = \alpha_1$ and $\beta_2 = \alpha_2$ $\forall k \in \{1, 2, \cdots, p\}$.

Transformer model trained with longer prompt length ($p = 20$). Figure 18 depicts similar evidence as Figure 1 of Transformer simulating PME (GMM) for a model trained with $d = 10, p = 20, \alpha_1 = \alpha_2 = \frac{1}{2}$. We see that all the observations discussed in §3.1 also hold true for this model. Transformer converges to PME (GMM) and PME (T_{prompt}) w.r.t. both `loss@`k (Figure 18a) and weights (Figure 18b) at $k = 10$ and keeps following them for larger k as well.

In summary, all the evidence strongly suggests that Transformer performs Bayesian Inference and computes PME corresponding to the task at hand. If the task is a mixture, Transformer simulates the PME of the task mixture as given by 3.

C.2 More complex mixtures

We start by training transformer models on the mixture of dense linear regression ($\mathcal{F}_{\text{DR}}$) and sparse linear regression ($\mathcal{F}_{\text{SR}}$) function classes. The function definition remains the same for both these classes i.e. $f : \boldsymbol{x} \mapsto \boldsymbol{w}_i^T \boldsymbol{x}$, but for $\mathcal{F}_{\text{DR}}$ we consider a standard gaussian prior on w and a sparse prior for $\mathcal{F}_{\text{SR}}$. We use the sparse prior from Garg et al. [2022], where we first sample $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}_d, \boldsymbol{I})$ and then randomly set its $d - s$ components as 0. We consider $s = 3$ throughout our experiments. Unless specified we consider the mixtures to be uniform i.e. $\alpha_i = 0.5$ and use these values to sample batches during training.

During the evaluation, we test the mixture model (denoted as Transformer $\mathcal{F}_{\{\text{DR, SR}\}}$) on the prompts sampled from each of the function classes in the mixture. We consider the model to have in-context learned the mixture of tasks if it obtains similar performance as the single-task models specific to these function classes. For example, a transformer model trained on the dense and sparse regression

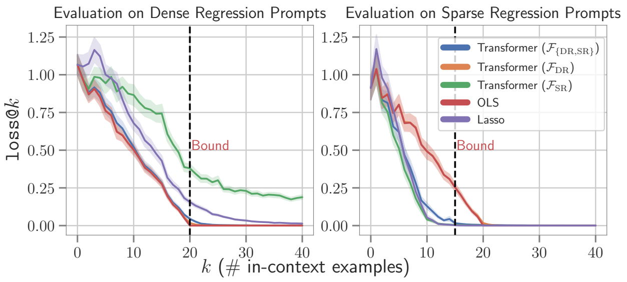
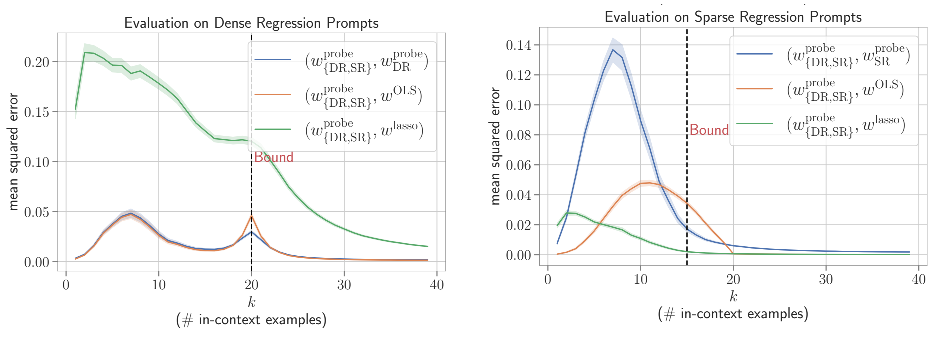


Figure 19: Comparing the performance of a Transformer model trained on dense and sparse regression mixture $\mathcal{F}_{\{\text{DR, SR}\}}$ with baselines, as well as single task models, trained on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SR}}$ individually.



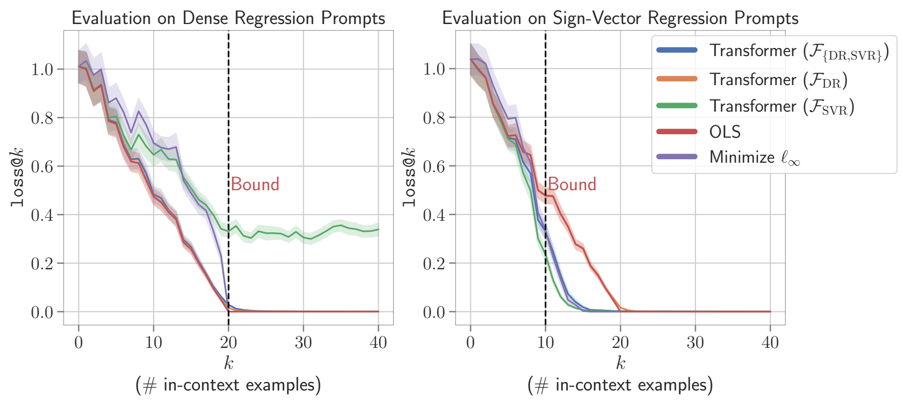
(a) (b)

Figure 20: Comparing the errors between the weights recovered from the mixture model trained on $\mathcal{F}_{\{\text{DR, SR}\}}$ mixture and different single task models and baselines while evaluating on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SR}}$ prompts

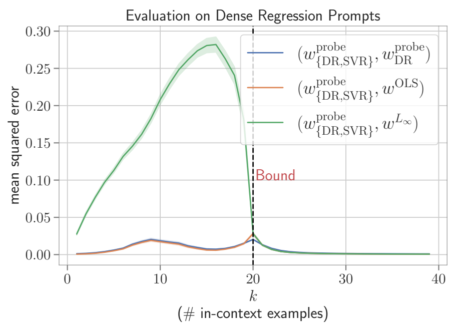
mixture (Transformer $\mathcal{F}_{\{\text{DR, SR}\}}$) should obtain performance similar to the single-task model trained on dense regression function class (Transformer $\mathcal{F}_{\text{DR}}$), when prompted with a function $f \sim \mathcal{F}_{\text{DR}}$ and vice-versa.

Results. The results for the binary mixtures of linear functions are given in Figure 19. As can be observed, the transformer model trained on $\mathcal{F}_{\{\text{DR, SR}\}}$ obtains performance close to the OLS baseline as well as the transformer model specifically trained on the dense regression function class $\mathcal{F}_{\text{DR}}$ when evaluated with dense regression prompts. On the other hand, when evaluated with sparse regression prompts the same model follows Lasso and single-task sparse regression model (Transformer ($\mathcal{F}_{\text{SR}}$)) closely. As a check, note that the single-task models when prompted with functions from a family different from what they were trained on, observe much higher errors, confirming that the transformers learn to solve individual tasks based on the in-context examples provided. Similar to GMMs in §3.1, here also we compare the implied weights from multi-task models under prompts for both $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SR}}$ and show that here again they agree with the weights recovered from single-task models as well as the strong baselines in this case (OLS and Lasso). We provide the plots for the weight agreement in this case in Figure 20.

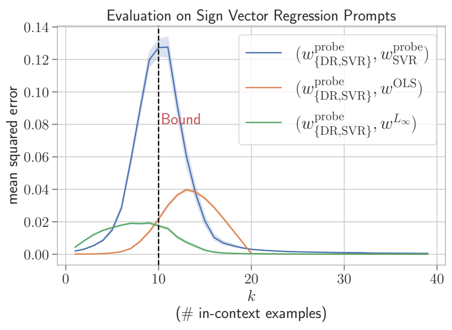
Next, we describe the results for other homogeneous mixtures $\mathcal{F}_{\{\text{DR, SVR}\}}$, $\mathcal{F}_{\{\text{DR, Skew-DR}\}}$ and $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$, as well as heterogeneous mixtures $\mathcal{F}_{\{\text{DR, DT}\}}$ and $\mathcal{F}_{\{\text{DT, NN}\}}$. As can be seen in Figure 21, the transformer model trained on $\mathcal{F}_{\{\text{DR, SVR}\}}$ mixture, behaves close to OLS when prompted with $f \in \mathcal{F}_{\text{DR}}$ and close to the L_∞ minimization baseline when provided sign-vector regression prompts ($f \in \mathcal{F}_{\text{SVR}}$). We also have similar observations for the $\mathcal{F}_{\{\text{DR, Skew-DR}\}}$ mixture case in Figure 22, where the multi-task ICL model follows the PME of both tasks when sufficient examples are provided from the respective task. Similarly, for the model trained on the tertiary mixture $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$ (as can be seen in Figure 23), the multi-task model can simulate the behavior of the three single-task models depending on the distribution of in-context examples. On $\mathcal{F}_{\text{SR}}$ and $\mathcal{F}_{\text{SVR}}$ prompts the multi-task model performs slightly worse compared to the single-task models trained on $\mathcal{F}_{\text{SR}}$ and $\mathcal{F}_{\text{SVR}}$ respectively, however once sufficient examples are provided (still < 20), they do obtain close errors.



(a)



(b)



(c)

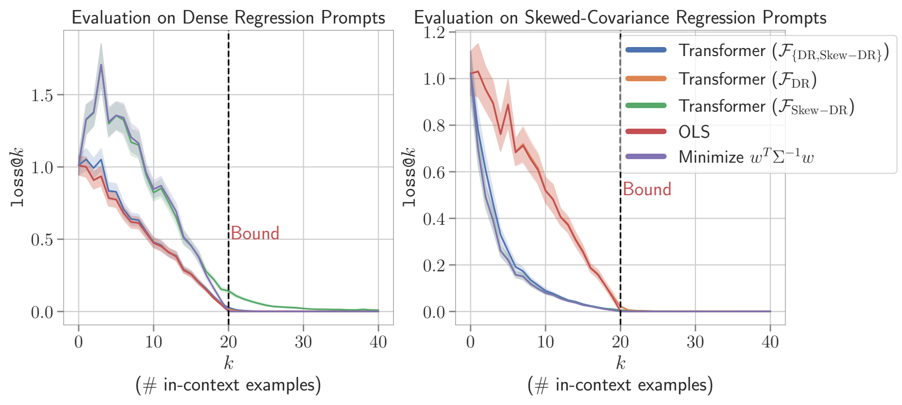
Figure 21: Comparing the performance of a Transformer model trained on dense and sign-vector regression mixture $\mathcal{F}_{\{\text{DR, SVR}\}}$ with baselines, as well as single task models, trained on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SVR}}$ individually. **Top**: Comparing `loss@`k values of the mixture model with single-task models with different prompt distributions. **Bottom**: Comparing the errors between the weights recovered from the mixture model and different single task models and baselines while evaluating on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SVR}}$ prompts.

This observation is consistent with the PME hypothesis i.e. once more evidence is observed the β values PME of the mixture should converge to the PME of the task from which prompt P is sampled. The results on heterogeneous mixtures we discuss in detail below:

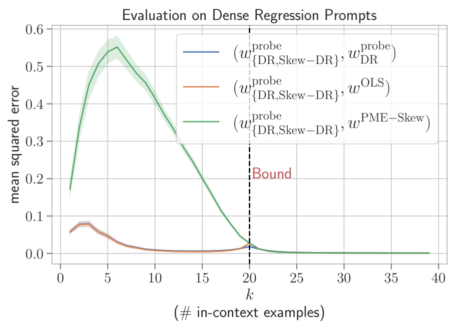
Heterogeneous Mixtures: Up until now, our experiments for the multi-task case have been focused on task mixtures where all function families have the same parameterized form i.e $w^T x$ for linear mixtures and $w^T \Phi(x)$ for Fourier mixtures. We now move to more complex mixtures where this no longer holds true. In particular, we consider dense regression and decision tree mixture $\mathcal{F}_{\{\text{DR, DT}\}}$ and decision tree and neural network mixture $\mathcal{F}_{\{\text{DT, NN}\}}$.

We follow Garg et al. [2022]'s setup for decision trees and neural networks. We consider decision trees of depth 4 and 20-dimensional input vectors $\boldsymbol{x}$. A decision tree is sampled by choosing the split node randomly from the features at each depth, and the output of the function is given by the values stored in the leaf nodes which are sampled from $\mathcal{N}(0, 1)$. For neural networks, we consider 2-layer (1 hidden + 1 output) multi-layer perceptrons (MLP) with ReLU non-linearity i.e. $f(x) = \sum_{i=1}^{r} \alpha_i \texttt{ReLU}(w_i^T \boldsymbol{x})$, where $\alpha \in \mathbb{R}$ and $\boldsymbol{w}_i \in \mathbb{R}^d$. The network parameters a_is and $\boldsymbol{w}_i$s are sampled from $\mathcal{N}(0, 2/r)$ and $\mathcal{N}(0, 1)$ respectively. The input vectors $\boldsymbol{x}_i$s are sampled from $\mathcal{N}(0, 1)$ for both tasks. We consider greedy tree learning and stochastic gradient descent [5] over a 2-layer MLP as our baselines for decision trees and neural networks respectively. The values of hyperparameters for baselines such as the number of gradient descent steps, initial learning rate for Adam, etc. are the same as Garg et al. [2022].

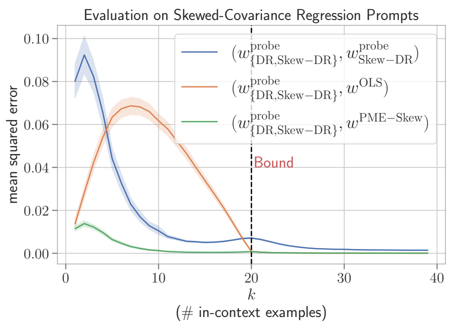
[5]In practice, we use Adam just like Garg et al. [2022]



(a)



(b)



(c)

Figure 22: Comparing the performance of a Transformer model trained on dense and skewed-covariance regression mixture $\mathcal{F}_{\{\text{DR, Skew-DR}\}}$ with baselines, as well as single task models, trained on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{Skew-DR}}$ individually. **Top**: Comparing `loss@`k values of the mixture model with single-task models with different prompt distributions. Red (OLS) and orange (Transformer ($\mathcal{F}_{\text{DR}}$)) curves overlap very closely, so are a bit hard to distinguish in the plots. Similarly in the top right plot, purple (Minimize $\boldsymbol{w}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{w}$) and green (Transformer $\mathcal{F}_{\text{Skew-DR}}$) curves overlap. **Bottom**: Comparing the errors between the weights recovered from the mixture model and different single task models and baselines while evaluating on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{Skew-DR}}$ prompts.

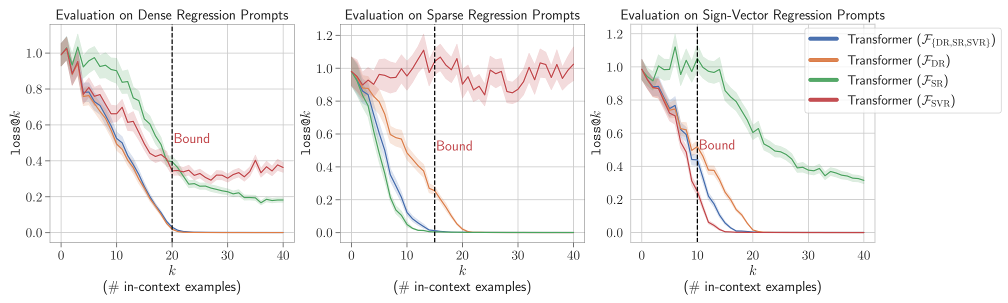
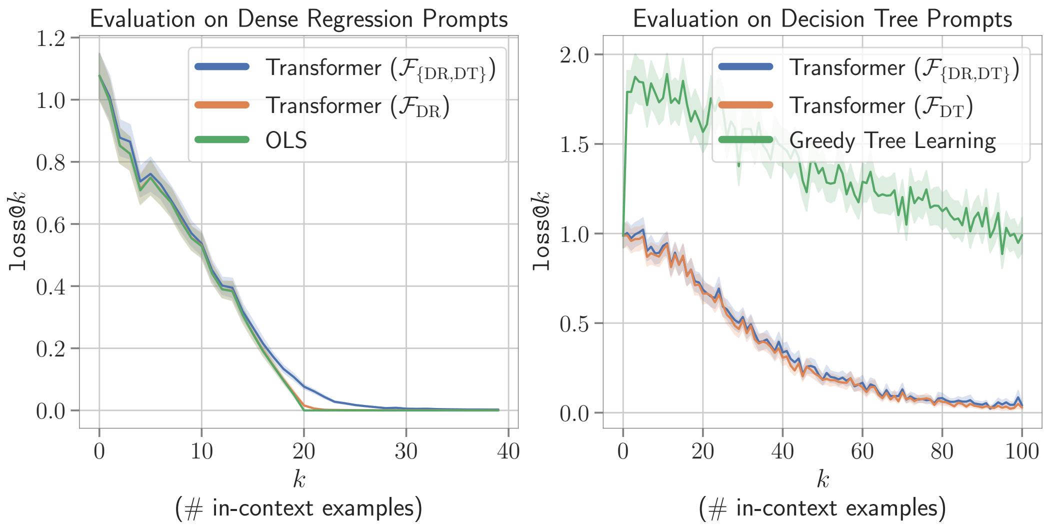
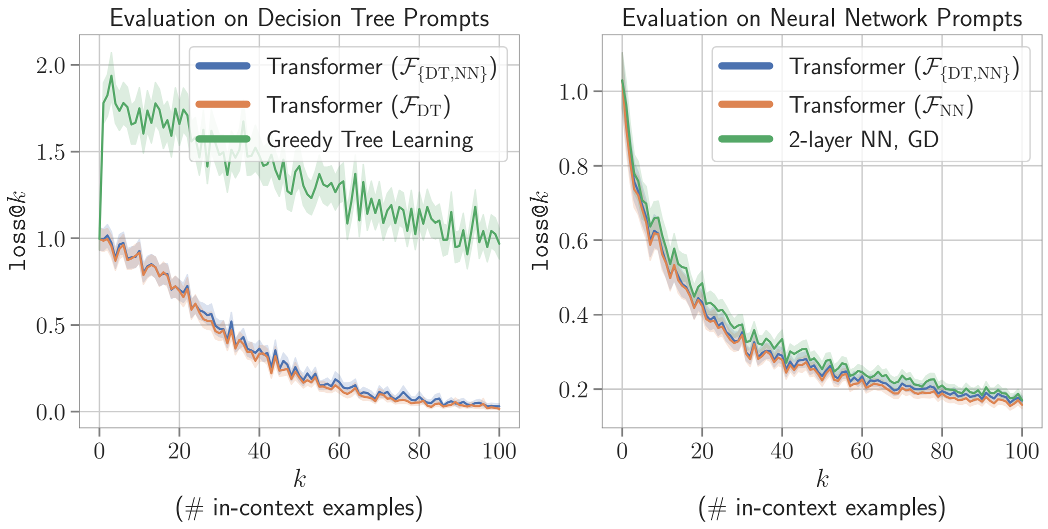


Figure 23: Comparing the performance of transformer model trained to in-context learn $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$ mixture family with the corresponding single task models.



(a)



(b)

Figure 24: Multi-task in-context learning for heterogeneous mixtures: Results for (a) $\mathcal{F}_{\{\mathrm{DR,\,DT}\}}$, and (b) $\mathcal{F}_{\{\mathrm{DT,\,NN}\}}$.

The results for the two mixtures are provided in Figure 24. The mixture model Transformer ($\mathcal{F}_{\{\text{DR, DT}\}}$) follows the single task model Transformer ($\mathcal{F}_{\text{DR}}$) when provided in-context examples from $f \sim \mathcal{F}_{\text{DR}}$ and agrees with Transformer ($\mathcal{F}_{\text{DT}}$) when prompted with $f \sim \mathcal{F}_{\text{DT}}$ (Figure 24a. Similarly, we have consistent findings for $\mathcal{F}_{\{\text{DT, NN}\}}$ mixture as well, where the model learns to solve both tasks depending upon the input prompt (Figure 24b).

C.3 Fourier series mixture detailed results

We consider a mixture of Fourier series function classes with different maximum frequencies, i.e. $\mathcal{F}^{\text{fourier}}_{\Phi_{1:N}} = \{\mathcal{F}^{\text{fourier}}_{\Phi_1}, \cdots, \mathcal{F}^{\text{fourier}}_{\Phi_N}\}$. We consider $N = 10$ in our experiments and train the models using a uniform mixture with normalization. During evaluation, we test individually on each $\mathcal{F}^{\text{fourier}}_{\Phi_M}$, where $M \in [1, N]$. We compare against consider two baselines: i) OLS Fourier Basis $\mathcal{F}^{\text{fourier}}_{\Phi_M}$ i.e. performing OLS on the basis corresponding to the number of frequencies M in the ground truth function, and ii) $\mathcal{F}^{\text{fourier}}_{\Phi_N}$ which performs OLS on the basis corresponding to the maximum number of frequencies in the mixture i.e. N.

Figure 25a plots the `loss@`k metric aggregated over all the $M \in [1, N]$ for the model and the baselines. The performance of the transformer lies somewhere in between the gold-frequency baseline (OLS Fourier Basis $\mathcal{F}^{\text{fourier}}_{\Phi_M}$) and the maximum frequency baseline ($\mathcal{F}^{\text{fourier}}_{\Phi_N}$), with the model performing much better compared to the latter for short prompt lengths ($k < 20$) while the former baseline performs better. We also measure the frequencies exhibited by the functions predicted by the transformer in Figure 25b. We observe that transformers have a bias towards lower frequencies when prompted with a few examples; however, when given sufficiently many examples they are able to recover the gold frequencies. This simplicity bias can be traced to the training dataset for the mixture since lower frequencies are present in most of the functions of the mixture while higher frequencies will be more rare: Frequency 1 will be present in all the function classes whereas frequency N will be present only in $\mathcal{F}^{\text{fourier}}_{\Phi_N}$. Our results indicate that the simplicity bias in these models during in-context learning arises from the training data distribution. We confirm the above observations by detailing results for different combinations of M and k in Figure 26.

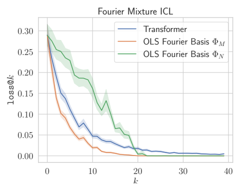
C.3.1 Complexity Biased Pre-training

To further verify this observation, we also consider the case where the training data is biased towards high frequencies and check if transformers trained with such data exhibit bias towards high frequencies (complexity bias). To motivate such a mixture, we first define an alternate fourier basis: $\Phi_{n_0,N}(x) = [\cos(n_0\pi/L), \sin(n_0\pi/L), \cos((n_0+1)\pi/L), \sin((n_0+1)\pi/L), \cdots, \cos(N\pi/L), \sin(N\pi/L)]$, where $n_0 \geq 0$ is the minimum frequency in the basis. $\Phi_{n_0,N}$ defines the function family $\mathcal{F}^{\text{fourier}}_{\Phi_{n_0,N}}$ and equivalently we can define the mixture of such function classes as $\mathcal{F}_{\Phi^{\text{fourier}}_{1:N,N}} = \{\mathcal{F}^{\text{fourier}}_{\Phi_{1,N}}, \cdots, \mathcal{F}^{\text{fourier}}_{\Phi_{N,N}}\}$. One can see such a mixture will be biased towards high frequency; frequency N is present in each function class of the mixture, while frequency 1 is only present in $\mathcal{F}^{\text{fourier}}_{\Phi_{1,N}}$. We train a transformer model on such a mixture for $N = 5$ and at test time, we evaluate the model on functions $f \sim \mathcal{F}^{\text{fourier}}_{\Phi_{m_0,M}}$ Figure 25c shows the inductive biases measure from this trained model and we can clearly observe a case of complexity bias, where at small prompt lengths, the model exhibited a strong bias towards the higher end of the frequencies that it was trained on i.e. close to 5.

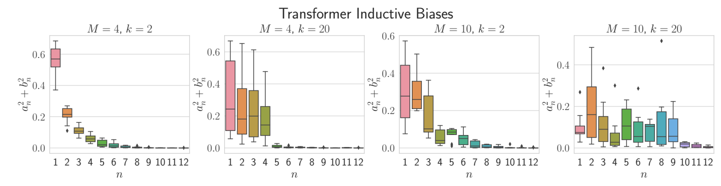
We also trained models for higher values of the maximum frequency i.e. $N = 10$ for the high-frequency bias case, but interestingly observed the model failed to learn this task mixture. Even for $N = 5$, we noticed that the convergence was much slower compared to training on the simplicity bias mixture $\mathcal{F}^{\text{fourier}}_{\Phi_{1:N}}$. This indicates, while in this case, the origin of simplicity bias comes from the training data, it is harder for the model to learn to capture more complex training distributions, and simplicity bias in the pre-training data distribution might lead to more efficient training Mueller and Linzen [2023].

C.4 Conditions necessary for multi-task ICL

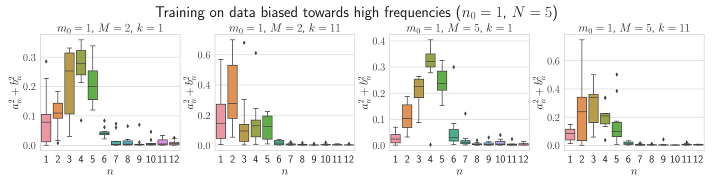
We observed that the training setup can also influence the ability of transformers to simulate the Bayesian predictor during ICL. Particularly, in our initial experiments with $\mathcal{F}_{\{\text{DR, SR}\}}$ mixture (§C.2), transformers failed to learn to solve the individual tasks of the mixture and were following OLS for both $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SR}}$ prompts. To probe this, we first noted that the variance of the function



(a)



(b)



(c)

Figure 25: In-context learning on the Fourier series mixture class. **Top Left**: Comparing transformers with the baselines. Errors are computed on batches of 128 for $M \in [1, 10]$ and aggregated in the plot. **Top Right**: Visualizing the frequencies of the simulated function by transformers. **Bottom**: Training transformer on high-frequency biased Fourier mixture $\mathcal{F}_{\Phi^{fourier}_{1:N,N}}$ and visualizing the simulated frequencies of the trained model.

outputs varied greatly for the two tasks, where for dense regression it equals d and equals the sparsity parameter s for sparse regression. We hypothesized that the model learning to solve just dense regression might be attributed to the disproportionately high signal from dense regression compared to sparse. To resolve this, we experimented with increasing the sampling rate for the $\mathcal{F}_{\text{SR}}$ task family during training. Particularly on training the model with $\alpha_{\text{SR}} = 0.87$, we observed that the resulting model did learn to solve both tasks. Alternatively, normalizing the outputs of the two tasks such that they have the same variance and using a uniform mixture ($\alpha_{\text{SR}} = 0.5$) also resulted in multi-task in-context learning capabilities (also the setting of our experiments in Figure 19). Hence, the training distribution can have a significant role to play in the model acquiring abilities to solve different tasks as has been also observed in other works on in-context learning in LLMs Razeghi et al. [2022], Chan et al. [2022a].

We also studied if the curriculum had any role to play in the models acquiring multi-task in-context learning capabilities. In our initial experiments without normalization and non-uniform mixtures, we observed that the model only learned to solve both tasks when the curriculum was enabled. However, training the model without curriculum for a longer duration ($\approx$ more training data), we did observe it to eventually learn to solve both of the tasks indicated by a sharp dip in the evaluation loss for the sparse regression task during training. This is also in line with recent works Hoffmann et al. [2022], Touvron et al. [2023], which show that the capabilities of LLMs can be drastically improved by scaling up the number of tokens the models are trained on. Detailed results concerning these findings are in Figure 27.

Figure 27 compares transformer models trained on $\mathcal{F}_{\{\text{DR, SR}\}}$ mixture with different setups i.e. training without task-normalization and uniform mixture weights α_i's (Figure 27a), training without task-normalization and non-uniform mixture weights (Figure 27b), and training with task normalization and uniform mixture weights (Figure 27c). As described above, we perform task normalization by ensuring that the outputs $f(x)$ for all the tasks have the same variance, which results in all the tasks providing a similar training signal to the model. To perform normalization, we simply divide the weights w sampled for the tasks by a normalization constant, which is decided according to the nature of the task. With this, we make sure that the output $y = w^T x$ has a unit variance. The normalization constants for different tasks are provided in Table 2.

All the experiments discussed above (like most others in the main paper) were performed using curriculum learning. As discussed above, we investigated if the curriculum has any effect on multi-task ICL capabilities. The results for the same are provided in Figure 28.

We also explore the effect of normalization on multi-task ICL in Figure 29 for $\mathcal{F}_{\{\text{DR, SVR}\}}$ task. As can be seen in Figure 29a, for this particular mixture even while training the model without

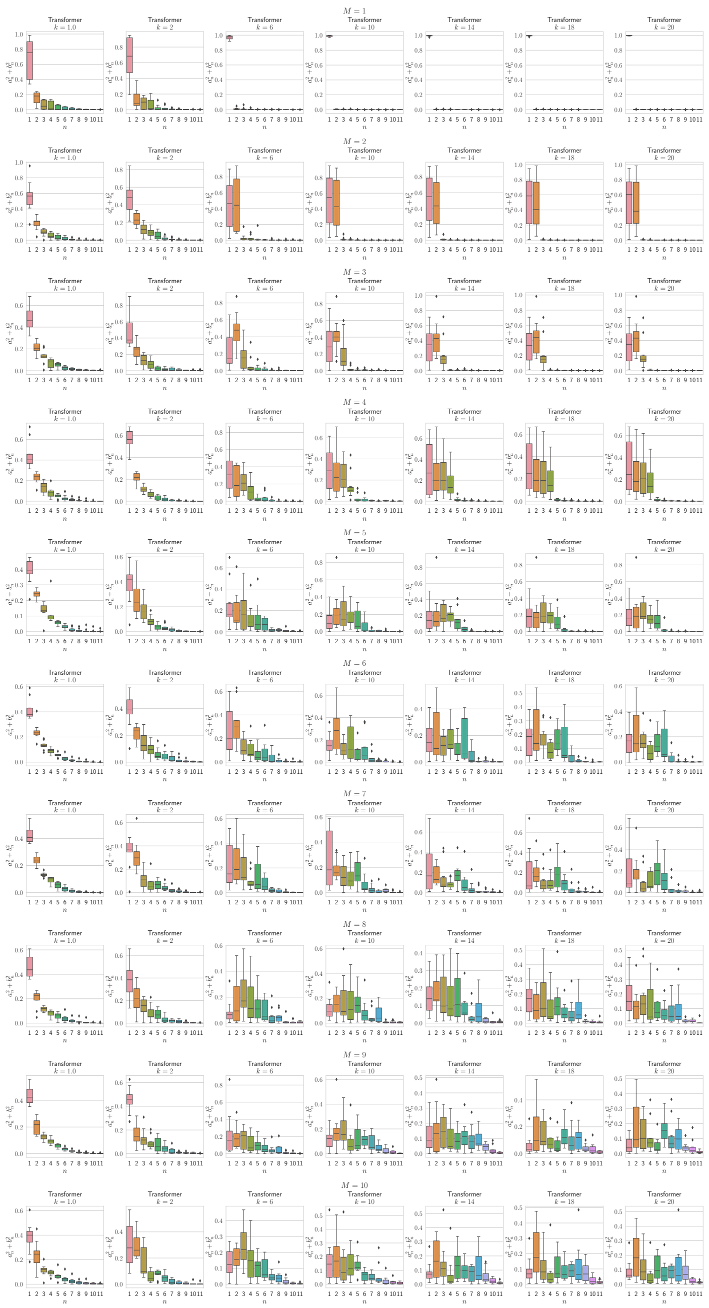
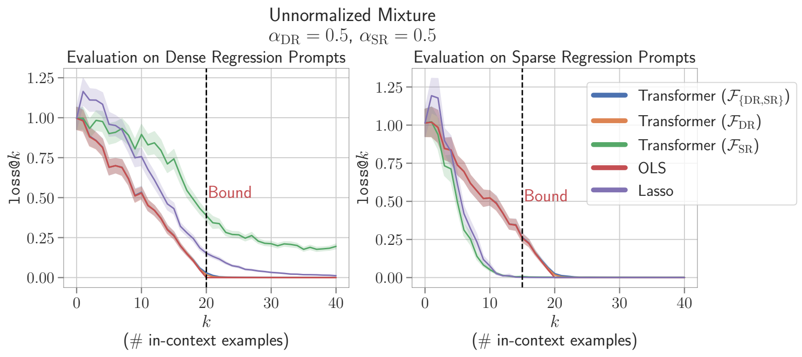
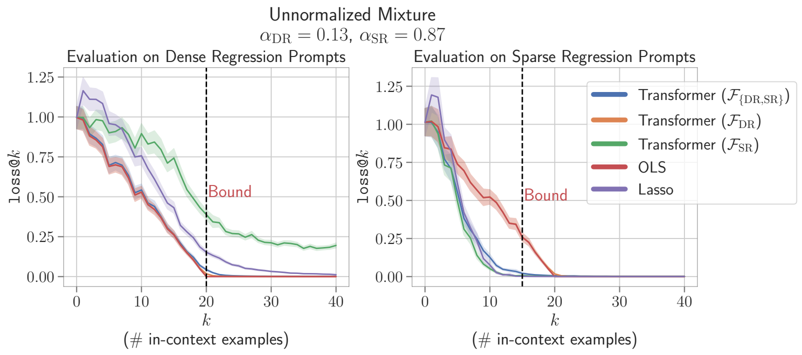


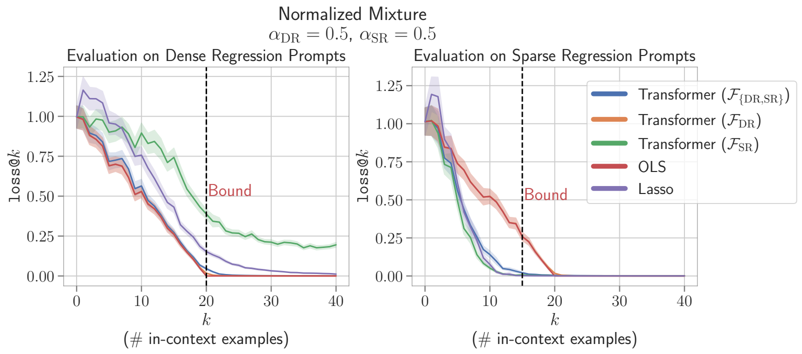
Figure 26: In-context learning of Fourier series mixture class. Measuring the frequencies of the simulated function by the transformer for different values of M (maximum frequency) and k (number of in-context examples). Showcases the simplicity bias behavior exhibited by the model at low frequencies.



(a)

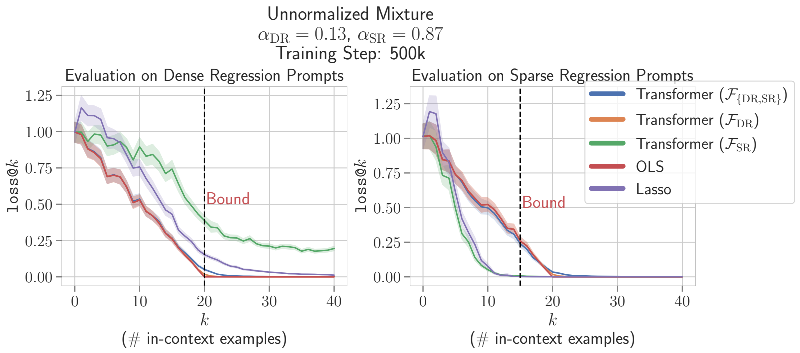


(b)

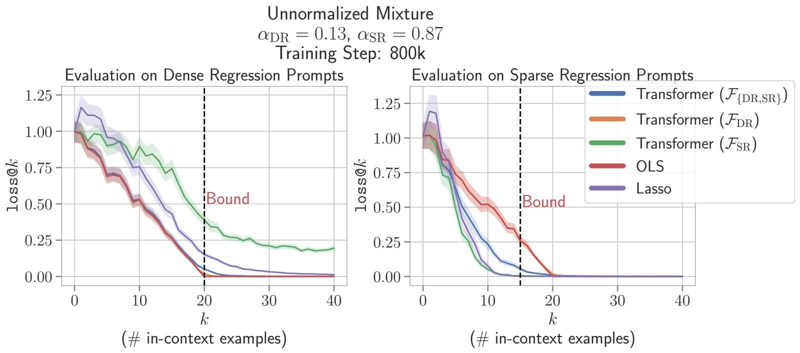


(c)

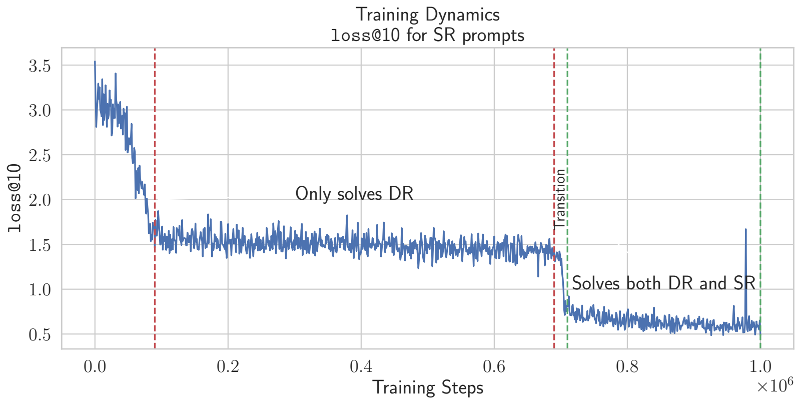
Figure 27: Conditions affecting multi-task ICL in transformers. **Top**: Evaluating `loss@`k for transformer model trained on $\mathcal{F}_{\{\text{DR, SR}\}}$ task family without normalization and considering uniform mixtures (i.e. $\alpha_{\text{DR}} = \alpha_{\text{SR}} = 0.5$), and comparing with single-task models and baselines. While the blue curve (Transformer $\mathcal{F}_{\{\text{DR, SR}\}}$) is hard to see here, it is because it overlaps almost perfectly with the red curve corresponding to OLS in both cases.**Center**: Similar plots as above but for the model trained on the mixture $\mathcal{F}_{\{\text{DR, SR}\}}$ with non-uniform weights i.e. $\alpha_{\text{DR}} = 0.13, \alpha_{\text{SR}} = 0.87$. **Bottom**: Training the model with the normalized (and uniform) mixture such that outputs for the two tasks have the same variance. All the models are **trained with the curriculum**. The discussion continues in Figure 28 for the models trained without curriculum.



(a)



(b)

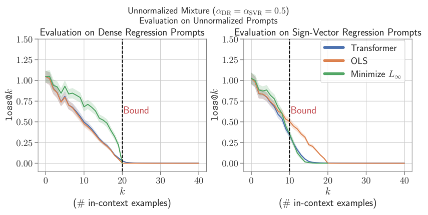


(c)

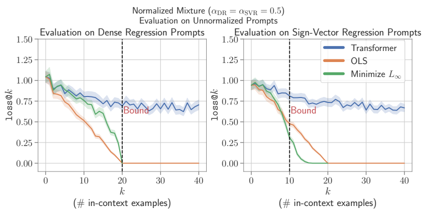
Figure 28: Evaluating transformer model trained **without curriculum** on $\mathcal{F}_{\{\mathrm{DR,\,SR}\}}$ task family without normalization and non-uniform weights i.e. $\alpha_{\mathrm{DR}} = 0.13, \alpha_{\mathrm{SR}} = 0.87$ (similar to Figure 27b). **Top**: Evaluating the checkpoint corresponding to the 500k training step of the aforementioned model. Again, the blue curve (Transformer $\mathcal{F}_{\{\mathrm{DR,\,SR}\}}$) is hard to see here, but it is because it overlaps almost perfectly with the red curve corresponding to OLS in both cases.**Center**: Evaluating the same model but a much later checkpoint i.e. at 800k training step. **Bottom**: Evolution of `loss@10` on $\mathcal{F}_{\mathrm{SR}}$ prompts while training the above model.

Table 2: Normalization constants used for different tasks to define normalized mixtures for multi-task ICL experiments. Here d denotes the size of the weight vectors used in linear-inverse problems as well as the last layer of the neural network. s refers to the sparsity of sparse regression problems, r is the hidden size of the neural network and N refers to the maximum frequency for Fourier series.

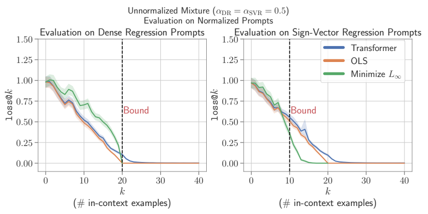
Function Family	**Normalization Constant**
Dense Regression	$\sqrt{d}$
Sparse Regression	$\sqrt{s}$
Sign-Vector Regression	$\sqrt{d}$
Fourier-Series	N
Degree-2 Monomial Basis Regression	$\sqrt{\lvert\mathcal{S}\rvert}$
Decision Trees	1
Neural Networks	$\sqrt{\frac{dr}{2}}$



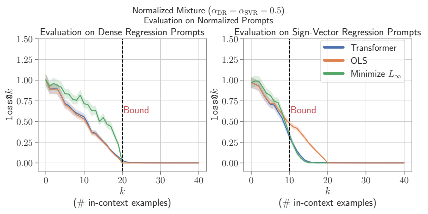
(a)



(c)



(b)



(d)

Figure 29: **Effect of output normalization on multi-task ICL in transformers. Top Left (a)**: A transformer model is trained on a uniform mixture of $\mathcal{F}_{\{\mathrm{DR, SVR}\}}$ task family (i.e. $\alpha_{\mathrm{DR}} = \alpha_{\mathrm{SVR}} = 0.5$) without normalization. Evaluating `loss@`k for this model on unnormalized prompts (where outputs $f(x)$ are not normalized to have unit variance i.e. same as training). Note that for the $\mathcal{F}_{\{\mathrm{DR, SVR}\}}$ task family even without normalization the outputs $f(x)$ have the same mean and variance ($\mu = 0, \sigma^2 = 20$) for both the tasks. **Bottom Left (b)**: Evaluating `loss@`k for the model in (a) on normalized prompts (where outputs $f(x)$ for both tasks are normalized to have unit variance). **Top Right (c)**: A transformer model is trained on a uniform mixture of $\mathcal{F}_{\{\mathrm{DR, SVR}\}}$ task family (i.e. $\alpha_{\mathrm{DR}} = \alpha_{\mathrm{SVR}} = 0.5$) with normalization. Evaluating `loss@`k for this model on unnormalized prompts. **Bottom Right (d)** Evaluating `loss@`k for the model in (c) on normalized prompts. All the models are trained with the curriculum.

normalization, the model exhibited multi-task ICL, which can be explained by both tasks having the same output variance (i.e. d). Interestingly, when we evaluate this model (i.e. the one trained on unnormalized mixture) on in-context examples which have the outputs $f(x_i)$'s normalized, the model fails to solve $\mathcal{F}_{\mathrm{SVR}}$ and follows OLS baseline for both the tasks. We hypothesize that since this situation represents Out of Distribution (OOD) evaluation and the model might not be robust towards performing multi-task ICL on prompts that come from a different distribution than those seen during training. Exploring OOD generalization in the multi-task case is a compelling direction that we leave for future work.