Transformers Can Learn To Solve Linear-Inverse Problems In-Context

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

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

Navin Goyal
Microsoft Research India
navingo@microsoft.com

Abstract

In-context learning is one of the surprising and useful features of large language models. How it works is an active area of research. Recently, stylized meta-learning-like setups have been devised that train these models on a sequence of input-output pairs $(x, f(x))$ from a function class using the language modeling loss and observe generalization to unseen functions from the same class. One of the main discoveries in this line of research has been that for several problems such as linear regression, trained transformers learn algorithms for learning functions in context. We extend this setup to different types of linear-inverse problems and show that transformers are able to in-context learn these problems as well. Additionally, we show that transformers are able to recover the solutions in fewer-measurements than the number of unknowns, leveraging the structure of these problems and are in accordance with the recovery bounds given by Chandrasekaran et al. [4]. Finally, we also discuss the multi-task setup, where the transfromer is pre-trained on multiple types of linear-inverse problems at once and show that at inference time, given the measurements, they are able to identify the correct problem structure and solve the inverse problem efficiently.

1 Introduction

In-context learning (ICL) is one of the ingredients behind the astounding performance of large language models (LLMs) [2, 21]. Unlike traditional learning, ICL is the ability to learn new functions f without weight updates during inference from input-output examples $(x, f(x))$; in other words, learning happens *in context*. For instance, given prompt `up -> down, low -> high, small ->` a pretrained LLM will likely produce output `big`. It infers that the function in the two examples is the antonym of the input and applies it on the new input. This behaviour extends to more sophisticated and novel functions unlikely to have been seen during training e.g. [13, 24, 12, 10, 5]. Apart from its applications in NLP, more broadly ICL can also be viewed as providing a method for meta-learning [16, 18, 8] 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 stylized setups [25, 3, 6, 23, 7]. In this paper we focus on the framework of Garg et al. [6] which is closely related to meta-learning. Unlike in NLP where training is done on documents for next-token prediction task, here 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 Meta-ICL (or **MICL**). A striking discovery in Garg et al. [6] was that for several function classes, transfromer-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

[*]Equal contribution

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

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 $\boldsymbol{x}_{\text{test}}$. A follow-up work Akyürek et al. [1] shows that for linear regression, high-capacity transfromers match the minimum Bayes risk solution during ICL.

In our work, we extend the above setup to different linear inverse problems. Inverse problems (IPs) are ubiquitous in different areas of science and engineering and involve inferring the state of a system from a (finite) set of measurements. In linear inverse problems (LIPs), the state of the system can be mapped to the measurements using a linear operator, i.e., $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}$, where $\boldsymbol{y} \in \mathbb{R}^k$ corresponds to the k measurements, $\boldsymbol{w} \in \mathbb{R}^d$ is the state of the system and $\boldsymbol{X} \in \mathbb{R}^{k\times d}$ is the linear map (observed). We are particularly interested in *ill-posed* LIPs in our work, where the number of measurements is fewer than the problem size i.e. $k < d$. *Prior* information about $\boldsymbol{w}$ can be used to impose well-formedness to the LIP by utilizing penalty functions [4] or regularized objectives [20], or by performing Bayesian inference [17]. In our work, we ask the question: *When pre-trained with functions sampled from a class of linear inverse problems (with a fixed prior), do transformers learn to utilize prior information about the problem to solve it effectively?* In other words to what extent do transformers behave as penalty based solvers or Bayesian-predictors during in-context learning for solving LIPs?

Our contributions can be summarized as follows. **(1)** We extend the experiments in Garg et al. [6] to multiple linear-inverse problems like sign-vector regression, low-rank regression, skewed-covariance regression and re-study sparse-regression through a Bayesian perspective . We show transformers remarkably agree with the penalty function based solvers from [4]. Further, we find that transformers also agree with the Bayesian predictor (wherever it is tractable). **(2)** We generalize the MICL setup to work with multiple function classes i.e. during pre-training the function f is sampled from a mixture of function classes (e.g. dense regression and sparse regression) instead of a single function class. We call this setup Hierarchical-Meta-ICL or **HMICL**. **(3)** We show that transformers are able to solve the LIPs in the HMICL setup as well, and their performance on each constituent function class is identical to the transformer model trained on that single function class. **(4)** Our work strengthens the Bayesian hypothesis for in-context learning [25] by showing that transformers can leverage information from different types of priors from pre-training data to effectively solve ill-posed LIPs.

2 Background

We first discuss the in-context learning setup for learning function classes as introduced in [6] (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 $P = \big(\boldsymbol{x}_1, f(\boldsymbol{x}_i), \cdots, \boldsymbol{x}_p, f(\boldsymbol{x}_p), \boldsymbol{x}_{p+1}\big)$, we sample inputs $\boldsymbol{x}_i \sim \mathcal{D}_{\mathcal{X}}$ i.i.d. for $i \in \{1, \cdots p+1\}$. A transformer model M_θ is trained to predict $f(\boldsymbol{x}_{p+1})$ given P, using the objective:

$$\min_{\theta} \mathop{\mathbb{E}}_{f,\boldsymbol{x}_{1:p}} \left[\frac{1}{p+1} \sum_{i=0}^{p} \ell\left(M_\theta(P^i), f(\boldsymbol{x}_{i+1})\right) \right], \tag{1}$$

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 $\boldsymbol{x}_{1:p} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)$. 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. [6].

At test time, we present the model with 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 $k \in \{1, \cdots, p\}$. A test for in-context learning for a model can be performed by measuring $\texttt{loss@}k$ at increasing values of k and checking if the error goes down as more examples are provided [14].

2.1 Hierarchical Meta-ICL

We generalize the MICL setup, where we train transformers on functions sampled from a mixture of function classes instead of functions sampled from a single function class. Formally, we define a mixture of m function classes by a set $\boldsymbol{\mathcal{F}} = \{\mathcal{F}_1, \cdots, \mathcal{F}_m\}$ and sampling probabilities $\boldsymbol{\alpha} =$

$[\alpha_1, \cdots \alpha_m]^T$ with $\sum_{i=1}^m \alpha_i = 1$. We assume the input distribution $\mathcal{D}_{\mathcal{X}}$ to be same for each class $\mathcal{F}_i$. More concretely, the sampling process for P is defined as:

$$\begin{aligned} \mathcal{F}_i &\sim \boldsymbol{\mathcal{F}} \text{ s.t. } \mathbb{P}(\mathcal{F} = \mathcal{F}_i) = \alpha_i \\ f &\sim \mathcal{F}_i \\ \boldsymbol{x}_j &\sim \mathcal{D}_{\mathcal{X}}, \forall j \in \{1, \cdots, p\} \\ \text{Finally, } P &= \left(\boldsymbol{x}_1, f(\boldsymbol{x}_1), \cdots \boldsymbol{x}_p, f(\boldsymbol{x}_p), \boldsymbol{x}_{p+1}\right) \end{aligned}$$

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 whether the Bayesian inference can be used to explain the behavior of in-context learning in transformers. 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 upon the inputs provided.

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 aka the *Bayesian predictor*. Specifically, for the MICL and HMICL setup, predictions of this ideal 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 (1) 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 behind this computation.

2.2 Model and training details

We use the decoder-only transformer architecture [22, 15] with 12 layers, 8 heads, and a hidden size of 256 in the architecture (unless specified otherwise). We use 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. [6] 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 we choose $\mathcal{D}_{\mathcal{X}}$ as the standard normal distribution i.e. $\mathcal{N}(0, 1)$, unless specified otherwise. For complete details of the experimental setup please check the Appendix (§A.2)

To accelerate training, we also use curriculum learning like Garg et al. [6] 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. Please refer to §A.2 in Appendix for a detailed discussion on the experimental setup.

3 In-context learning of linear inverse problems

We consider linear functions of the form, 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 different function classes is the distribution of $\boldsymbol{w}$ for different types of LIPs. Below we discuss different types of LIPs that we study:

Dense Regression. This represents the simplest case of linear regression, where the prior on $\boldsymbol{w}$ is the standard Gaussian i.e. $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}_d, \boldsymbol{I})$. Gaussian prior enables explicit computation of the PME which is equal to the minimum L_2-norm solution of the equations forming the in-context 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 min L_2-norm solution and can be used to compute the Bayesian predictor.

Skewed-Covariance Regression. 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 Bayes-optimal predictor that can be obtained by minimizing $\boldsymbol{w}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{w}$ w.r.t to the constraints $\boldsymbol{w}^T \boldsymbol{x}_i = f(\boldsymbol{x}_i)$.

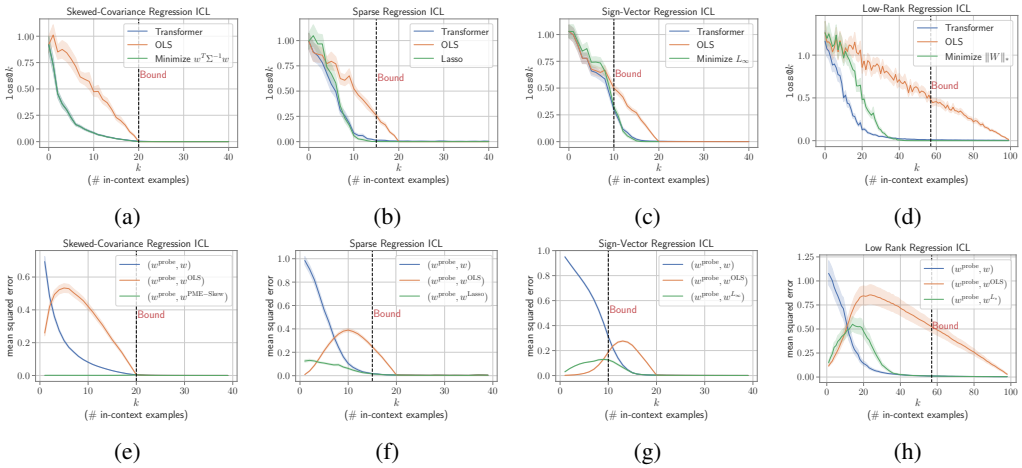


Figure 1: Comparing ICL in transformers for different linear functions with the relevant baselines. **Top**: `loss@`k values for transformers and baselines on skewed covariance, sparse, sign-vector, and low-rank regression tasks. **Bottom**: Comparing the errors between the implicit weights recovered 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.

Sparse Regression. Here 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. [6], to sample $\boldsymbol{w}$ for constructing prompts P, 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. While computing the exact PME appears to be intractable here, the MAP solution can be estimated using Lasso by assuming a Laplacian prior on $\boldsymbol{w}$ [19].

Sign-Vector Regression. 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 [11].

Low-Rank Regression. In this case, $\boldsymbol{w} \in \mathbb{R}^d$ 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 $\boldsymbol{W} = \boldsymbol{A}\boldsymbol{B}^T$.

For each function class above, there is a bound on the minimum number of in-context examples needed for 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 [4].

Note that the LIP is solved implicitly by the transformer and what it predicts is the output of the forward problem for a new input i.e. $\boldsymbol{w}^T\boldsymbol{x}_{\text{test}}$. To obtain the implied weight vectors, we use the approach by Akyürek et al. [1] that involves generating model's predictions $\{y_i'\}$ on the randomly sampled test inputs $\{\boldsymbol{x}_i'\}_{i=1}^{2d} \sim \mathcal{D}_{\mathcal{X}}$ (given $\boldsymbol{X}$ in the context) and then solving the system of equations to recover $\boldsymbol{w}^{\text{probe}}$. This gives us the solution to the LIP as implied by transformer during ICL.

3.1 Results

MICL 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 1b-1d compare the `loss@`k values on these tasks with different baselines. Additionally, we also compare the implied weights $\boldsymbol{w}^{\text{probe}}$ with the ground truth weights $\boldsymbol{w}$ as well as the solutions from different baselines (Figures 1f-1h).

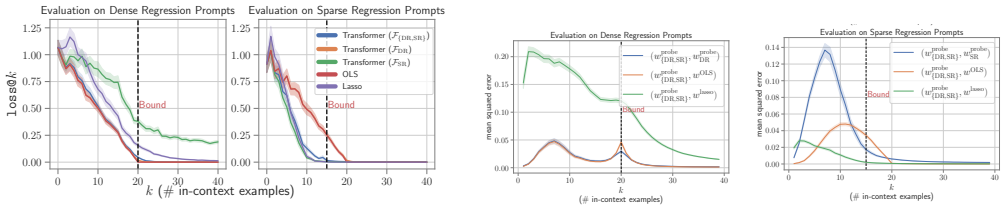


Figure 2: Comparing the performance of a transformer model trained on dense and sparse regression mixture $\mathcal{F}_{\{\text{DR, SR}\}}$ with baselines, as well as single task models, trained on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SR}}$ individually.

Since results for dense regression have been already covered in Akyürek et al. [1], we do not repeat them here, but for completeness provide them in Figure 3 of Appendix. For skewed-covariance regression, we observe that the transformer follows the Bayesian solution very closely both in terms of the `loss@`k values (Figure 1a) 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 1e). 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. in underdetermined region. The error curves of transformers for the tasks align closely with the errors of Lasso (Figure 1b), L_∞ minimization (Figure 1c), and L_* minimization (Figure 1d) baselines for the respective tasks . Interestingly, transformer performs better than L_* minimization baseline for low-rank regression. However, due to larger dimensionality of low-rank problem ($d = 100$), it requires a bigger model: 24 layers, 16 heads, 512 hidden size.

HMICL Results. We now discuss the results of transformers pre-trained on multiple types of linear-inverse problems. Due to scarcity of space, we report results for the mixture of Dense and Sparse regression ($\mathcal{F}_{\{\text{DR, SR}\}}$), but we have consistent observations for $\mathcal{F}_{\{\text{DR, Skew-DR}\}}$, $\mathcal{F}_{\{\text{DR, SVR}\}}$ and $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$ as well that we report in Appendix §A.3. During the evaluation, we test the mixture model 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 the test function class, e.g., a transformer model trained on the dense and sparse regression 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 similar for sparse).

As can be observed in Figure 2 (*left*), 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 on dense regression prompts. On the other hand, when evaluated on 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, yield much higher errors. We also recover the weights from the multi-task models when given prompts from each function class. In Figure 2 (*right*), we observe that the weights recovered by the mixture model start to agree with task-specific models once sufficient in-context examples are provided ($\approx$ recovery bound).

4 Conclusion

We showed that transformers can learn to solve different types of linear-inverse problems and their mixtures in-context. There are many interesting directions for future work. Due to difficulties in computing the Bayesian predictor, we were unable to establish for many LIPs that transformers do Bayesian prediction. This is an interesting theoretical challenge to resolve. Further, while we deal with linear-inverse problems in our paper and mainly study them to understand in-context learning in transformers, it would be interesting to explore if transformers can serve as solvers for practical inverse problems. Finally, in this paper we treated transformers as blackboxes: opening the box and uncovering the underlying mechanisms transformers use to do Bayesian prediction would be very interesting.

References

[1] 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.

[2] 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.

[3] 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, 2022. doi: 10.48550/arXiv.2210.05675. URL https://doi.org/10.48550/arXiv.2210.05675.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.201. URL https://aclanthology.org/2022.naacl-main.201.

[13] 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 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.759.

[14] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

[15] 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.

[16] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987. URL http://www.idsia.ch/~juergen/diploma.html.

[17] A. M. Stuart. Inverse problems: A bayesian perspective. *Acta Numerica*, 19:451–559, 2010. doi: 10.1017/S0962492910000061.

[18] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Springer Science and Business Media, 2012.

[19] 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.

[20] Andrei Nikolaevich Tikhonov. On the regularization of ill-posed problems. In *Doklady Akademii Nauk*, volume 153, pages 49–52. Russian Academy of Sciences, 1963.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

A Appendix

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

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

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

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

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

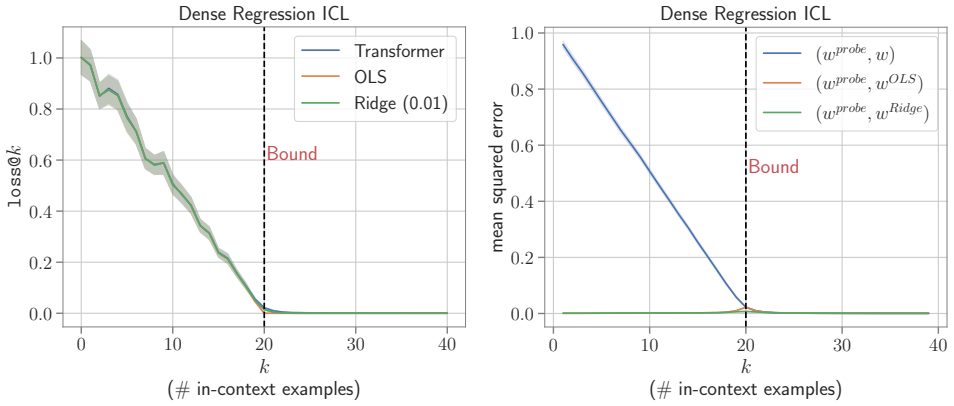
$$\begin{aligned}p_{\boldsymbol{\mathcal{F}}}(f|P) &= \frac{p_{\boldsymbol{\mathcal{F}}}(P|f)p_{\boldsymbol{\mathcal{F}}}(f)}{p_{\boldsymbol{\mathcal{F}}}(P)} = \frac{p(P|f)p_{\boldsymbol{\mathcal{F}}}(f)}{p_{\boldsymbol{\mathcal{F}}}(P)} = \frac{p(P|f)}{p_{\boldsymbol{\mathcal{F}}}(P)}\left[\alpha_1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

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 (2) we get

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

Table 1: The values of curriculum attributes used for each experiment. C_d and C_p denote the curriculum on number of input dimensions (d) and number of points (p) respectively.

Experiment	**Section**	C_d	C_p
Dense, Sparse and Sign-Vector Regression	§3	$[5, 20, 1, 2000]$	$[10, 40, 2, 2000]$
Low-Rank Regression	§3	Fixed ($d = 100$)	Fixed ($p = 114$)



(a) (b)

Figure 3: Results on the Dense Regression tasks mentioned in section §3.1.

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. s refers to the sparsity of sparse regression problems.

Function Family	**Normalization Constant**
Dense Regression	$\sqrt{d}$
Sparse Regression	$\sqrt{s}$
Sign-Vector Regression	$\sqrt{d}$
Skewed-Covariance Regression	$\sqrt{d}$

A.2 Experimental Setup

We use the decoder-only transformer architecture [22, 15] with 12 layers, 8 heads, and a hidden size (d_h) of 256 in the architecture (unless specified otherwise). 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. [6] 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 we choose $\mathcal{D}_{\mathcal{X}}$ as the standard normal distribution i.e. $\mathcal{N}(0, 1)$, unless specified otherwise.

Training and Implementation Details. We use Adam optimizer [9] to train our models. Our experiments were conducted on a system comprising 32 NVIDIA V100 16GB GPUs. The cumulative training time of all models for this project was $\sim$ 15,000 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. [6] code-base for our experiments. We use Pytorch[2] and Huggingface Transformers[3] libraries to implement the model architecture and training procedure. For the baselines against which we compare transformers, we use `scikit-learn`'s [4] implementation of OLS, Ridge and Lasso, and for L_∞ and L_* norm minimization given the linear constraints we use CVXPY[5].

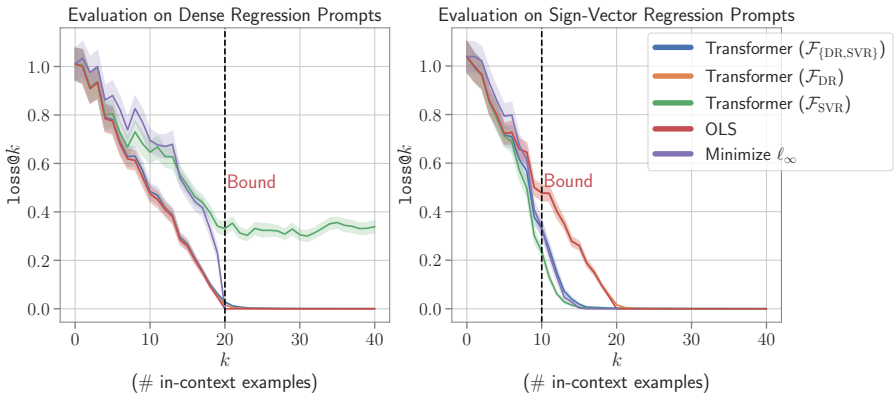
Curriculum Learning. To accelerate training, we also use curriculum learning like Garg et al. [6] for all our experiments where we start with simpler function distributions (lower values of d and p) at the beginning of training and increase the complexity as we train the model. We observe that curriculum 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. [6]. Table 1 states the curriculum used for each experiment, where the syntax followed for each column specifying curriculum is `[start,`

[2] `https://pytorch.org/`, citation to be added in final version

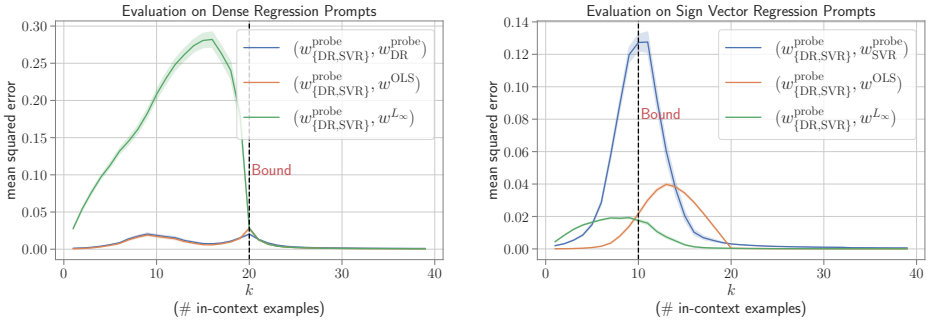
[3] `https://huggingface.co/docs/transformers/index`, citation to be added in final version

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

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



(a)



(b) (c)

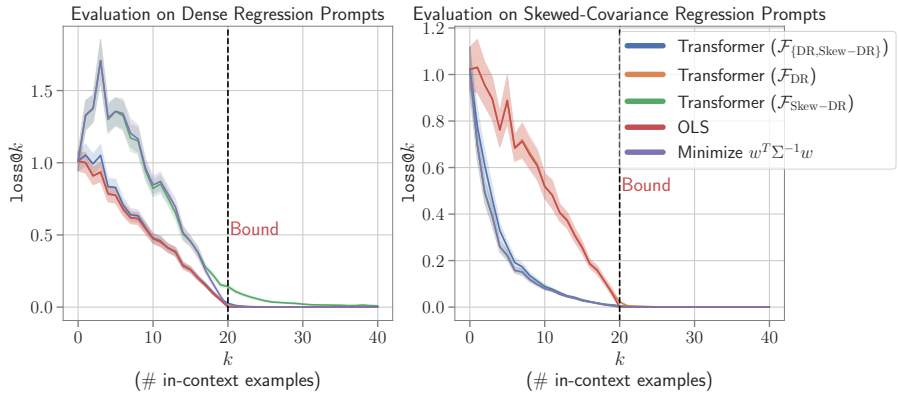
Figure 4: Comparing the performance of a Transformer model trained on dense and sign-vector regression mixture $\mathcal{F}_{\{\text{DR, SVR}\}}$ with baselines, as well as single task models, trained on $\mathcal{F}_{\text{DR}}$ and $\mathcal{F}_{\text{SVR}}$ individually. **Top**: Comparing `loss@`k 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.

`end, increment, interval]`. The value of the said attribute goes from `start` to `end`, increasing by `increment` every `interval` train steps.

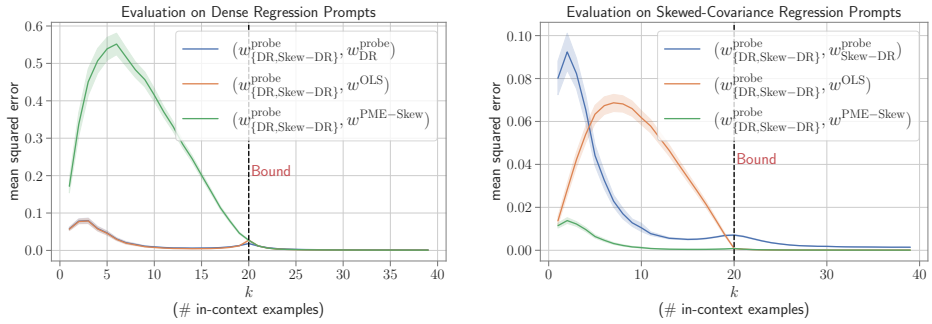
Normalization. We perform task normalization by ensuring that the outputs $f(x)$ for all the tasks have the same variance, which results in all the tasks providing a similar training signal to the model. To perform normalization, we simply divide the weights w sampled for the tasks by a normalization 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.

A.3 ICL on task mixtures

Here we detail some of the experiments with task mixtures that we discuss in passing in §3.1. Particularly, we describe the results for the mixtures $\mathcal{F}_{\{\text{DR, SVR}\}}$, $\mathcal{F}_{\{\text{DR, Skew-DR}\}}$ and $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$. As can be seen in Figure 4, 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 5, where the multi-task ICL model follows the Bayesian-predictor 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 6), 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 5: 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.

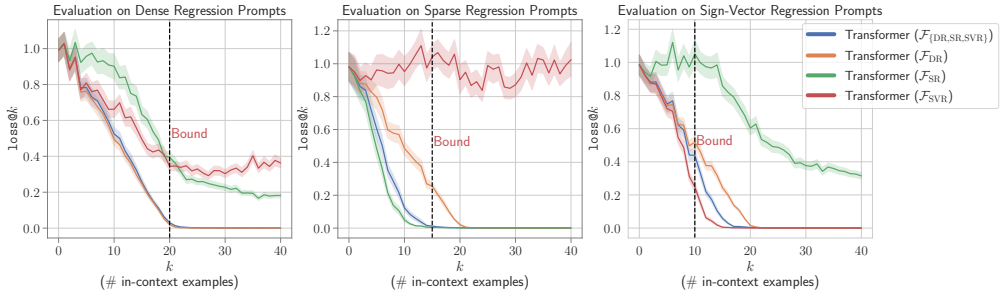


Figure 6: Comparing the performance of transformer model trained to in-context learn $\mathcal{F}_{\{\text{DR, SR, SVR}\}}$ mixture family with the corresponding single task models.