Does learning the right latent variables necessarily improve in-context learning?

Sarthak Mittal [†1 2] **Eric Elmoznino** [†1 2] **Leo Gagnon** [†1 2] **Sangnie Bhardwaj** [1 2 3] **Guillaume Lajoie** [*1 2] **Dhanya Sridhar** [*1 2]

Abstract

Large autoregressive models like Transformers can solve tasks through in-context learning (ICL) without learning new weights, suggesting avenues for efficiently solving new tasks. For many tasks, e.g., linear regression, the data factorizes: examples are independent given a task latent that generates the data, e.g., linear coefficients. While an optimal predictor leverages this factorization by inferring task latents, it is unclear if Transformers implicitly do so or instead exploit heuristics and statistical shortcuts through attention layers. In this paper, we systematically investigate the effect of explicitly inferring task latents by minimally modifying the Transformer architecture with a bottleneck to prevent shortcuts and incentivize structured solutions. We compare it against standard Transformers across various ICL tasks and find that contrary to intuition and recent works, there is little discernible difference between the two; biasing towards task-relevant latent variables does not lead to better out-of-distribution performance, in general. Curiously, we find that while the bottleneck effectively learns to extract latent task variables from context, downstream processing struggles to utilize them for robust prediction. Our study highlights the intrinsic limitations of Transformers in achieving structured ICL solutions that generalize, and shows that while inferring the right latents aids interpretability, it is not sufficient to alleviate this problem.

1. Introduction

Recent advancements in large language models (LLMs, Radford et al., 2019) showcase the Transformer architecture's (Vaswani et al., 2017) ability to perform novel tasks at inference through in-context learning (ICL, Brown et al., 2020). Indeed, LLMs can learn from instructions and demonstrations provided as input, without requiring gradient-based learning. While ICL plays a key role in many LLM abilities (Lu et al., 2024), such as instruction-following (Wei et al., 2022) and chain-of-thought reasoning (Wei et al., 2023), the factors that influence its generalization – particularly in out-of-distribution (OOD) settings – remain poorly understood. Although ICL can leverage a combination of instructions and demonstrations, our analysis focuses specifically on its ability to model predictions based on a task's training examples (demonstrations) provided in-context (Lampinen et al., 2024) beyond the modality of language.

A plausible hypothesis behind the success of ICL is that since many tasks are based on some low-dimensional latents (e.g., complex games are described completely through their rules, linear regression through its underlying coefficients), Transformers might generalize to novel queries by inferring the *task latents* from the context (Hendel et al., 2023; Todd et al., 2024; Yang et al., 2025). This describes a *parametric* (Bishop, 2006) modeling approach that breaks the prediction mechanism into two parts: 1) inferring the task latents (i.e. parameters) from the context, and then 2) using them to make predictions on novel queries. With the right prediction function, such an approach leads to systematic OOD generalization to new queries.

However, mounting evidence (Wang et al., 2023; Han et al., 2023; Song et al., 2024; Tang et al., 2023b; Bhaskar et al., 2024; Crosbie & Shutova, 2024) suggests that Transformers instead often employ more shallow solutions which rely on direct comparison of the query to demonstrations (reminiscent of induction heads, Olsson et al., 2022). This is closer in spirit to *non-parametric* approaches (such as nearest neighbors or kernel regression) which are known for their flexibility but poor generalization (Bishop, 2006). Since these solutions do not model the data generative process, they can be described as statistical shortcuts and risk poor performance on OOD context and queries – e.g., learning the actual linear predictor for linear regression can generalize to any distribution over training and test points, but nearest-neighbour based interpolation might not. Interestingly, the functional form of attention operations is almost identical to that of kernel regression (Tsai et al.,

[†]Equal Contribution, [*]Equal Supervision [1]Mila – Quebec AI Institute [2]Université de Montréal [3]Google DeepMind. Correspondence to: Sarthak Mittal <sarthmit@gmail.com>, Guillaume Lajoie <guillaume.lajoie@mila.quebec>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

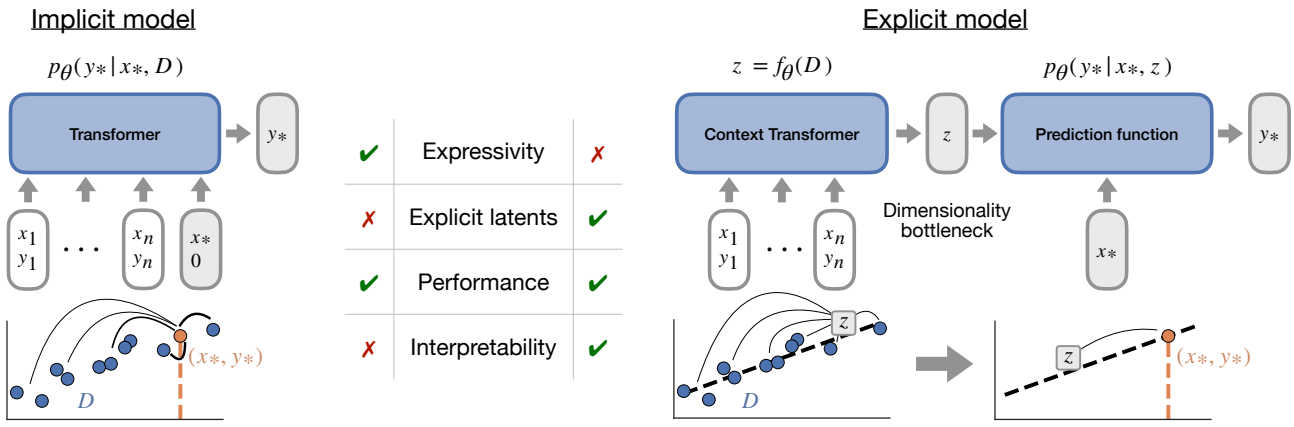

Figure 1. We compare the benefits of the implicit (*left*) and the explicit (*right*) model. Explicit models disentangle context aggregation and prediction into two separate functions with an inductive bias for inferring generative latent variables in order to solve the task. Implicit models are more expressive, but can learn non-parametric shortcut solutions that bypass latent variable inference.

2019; Han et al., 2023), making such solutions more natural for Transformers to express (Zhou et al., 2023).

In this paper, we aim to test the hypothesis that biasing Transformers against non-parametric solutions can improve their ICL performance by encouraging parametric modeling. We minimally modify the Transformer architecture to prevent such non-parametric shortcuts and compare the OOD performance of the resulting model to that of a traditional Transformer on a large array of ICL tasks. We call this altered architecture an *explicit model* because of its inductive bias of explicitly extracting structured latent variables to solve the tasks, and call the traditional Transformer architecture an *implicit model*. Specifically, the explicit model prevents the query from directly attending to demonstrations in the context by introducing a bottleneck between the processing of the context and the query (see Figure 1), similar to a conditional neural process (Garnelo et al., 2018a). To study the impacts of this inductive bias favoring parametric solutions, we need to establish that explicit models successfully recover task latents. As such, we consider synthetic and real tasks for which the latent mechanisms are well understood, and systematically analyse the impact of task latents on generalization by comparing explicit and implicit models.

We find that the explicit model does not outperform the implicit one on OOD data, challenging the aforementioned hypothesis that avoiding non-parametric solutions enhances generalization. Our investigation into this lack of improvement reveals that the issue often lies in the explicit model's prediction function which has to leverage the inferred latent variables for downstream predictions on the query. Our controlled experiments and analysis on the interpretable nature of the bottleneck revealed strong evidence that while the explicit model often extracts relevant task latents, these are not properly utilized by the prediction function.

While on one hand, our research demonstrates that using a simple bottleneck in a Transformer can improve interpretability and explicitly extract task-relevant latent variables, it also suggests that the limitations of Transformers in learning more structured and generalizable ICL solutions are not solely due to non-parametric shortcuts that skirt latent variable inference, but due to more fundamental architectural limitations. In sum, our contributions are:

- Formalizing a framework to test whether parametric ICL solutions generalize better out-of-distribution.
- Analyzing the benefits, or lack thereof, of inferring the true latents explicitly.
- Highlighting flaws in the prediction function and downstream latent utilization which hinders generalization.

2. Notation

Throughout the paper, we denote datasets with the symbol $\mathcal{D}$ which consists of a set of observations with inputs denoted via $\boldsymbol{x} \in \mathcal{X}$ and their corresponding outputs as $y \in \mathcal{Y}$. A task is defined by a functional mapping $g : \mathcal{X}, \mathcal{Z} \to \mathcal{Y}$ which maps observations $\boldsymbol{x}$ to labels y through some latents or parameters $\boldsymbol{z}$, eg. $y = \boldsymbol{z}^T \boldsymbol{x}$ for a linear regression task, or $y \sim \mathcal{N}(\cdot; \boldsymbol{z}^T \boldsymbol{x}, \sigma^2)$ for its stochastic counterpart. To ease readability, we will reserve $\boldsymbol{x}_* \in \mathcal{X}$ for the query, i.e. the test time observation we want to generalize to, and $y_* \in \mathcal{Y}$ its corresponding target. Finally, ψ denotes the parameters of the context aggregation component of explicit model, which inputs the dataset $\mathcal{D}$ and infers the corresponding parameters $\boldsymbol{z}_\psi(\mathcal{D})$, and γ the parameters of the prediction model which given a query $\boldsymbol{x}_*$ and parameters $\boldsymbol{z} \in \mathcal{Z}$, provides the prediction. For the implicit model, these operations are subsumed into a single model, with parameters φ.

3. Implicit vs. Explicit Inference

We look at ICL in the context of algorithmic problems where the task is to predict the target y_* from a query $\boldsymbol{x}_*$ when provided with some context examples $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$,

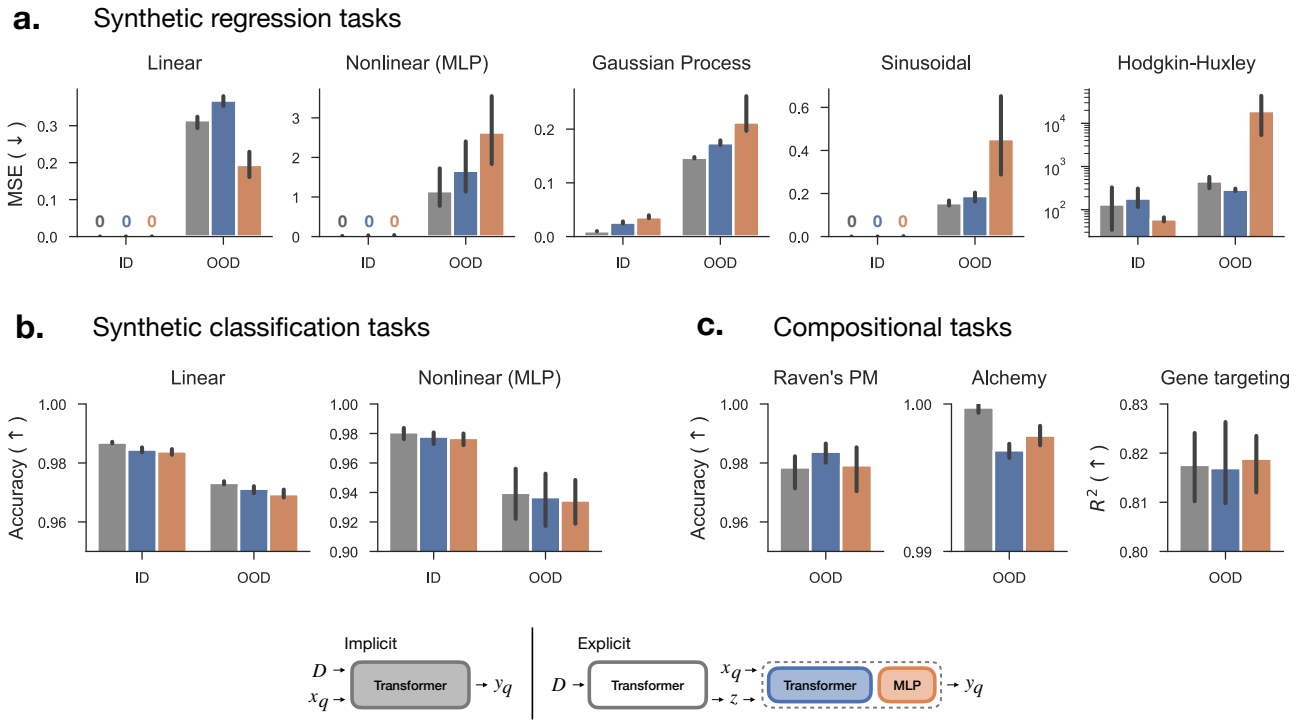

Figure 2. Comparison of implicit and explicit models in-distribution (ID) and out-of-distribution (OOD) across various domains: (a) synthetic regression, (b) classification, and (c) compositional generalization tasks. Implicit models are in shown gray, explicit models with Transformer prediction in blue, and with MLP prediction in orange. Further details about tasks is provided in Appendix B.

sharing a common underlying structure defined by the task latent $\boldsymbol{z}$ and a functional form g. The goal of ICL is to learn a function that can utilize the context set $\mathcal{D}$ to provide predictions for new queries $\boldsymbol{x}_*$. To achieve this, the model is trained on different draws of context sets $(\mathcal{D}_1, \mathcal{D}_2, ...)$ which share the same underlying functional mapping $g : \boldsymbol{x}, \boldsymbol{z} \rightarrow y$ but different realizations of the latent $\boldsymbol{z}$, for example $g(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{x}$ could be a linear regression system shared across different contexts $\mathcal{D}_1, \mathcal{D}_2, ...$, but the underlying latents could be different, i.e. $\mathcal{D}_1$ is generated from $\boldsymbol{z}_1$ while $\mathcal{D}_2$ from $\boldsymbol{z}_2$, similar to Von Oswald et al. (2023). We emphasize that in this setup, we are not training models to do next-token prediction as is done in language modeling; instead, given a fixed context $\mathcal{D}$ that includes n samples, we are attempting to make a prediction on a single novel query $\boldsymbol{x}_*$. We therefore do not use a causal Transformer, and we allow all tokens to attend to each other.

Often, ICL solutions are learned via maximum likelihood:

$$\arg\max_{\varphi} \; \mathbb{E}_{\mathcal{D},\boldsymbol{x}_*,y_*} \left[\log p_\varphi(y_*|\boldsymbol{x}_*, \mathcal{D})\right] \tag{1}$$

where p_φ represents the Transformer model and $\mathcal{D}$ is sampled from the parametric family defined through g. Thus, the transformer model p_φ must not only learn the form of the prediction function g, but also how to efficiently aggregate information from the context $\mathcal{D}$ to infer $\boldsymbol{z}$ for downstream predictions on $\boldsymbol{x}_*$. Thus, this general framework can be naturally decomposed into two distinct parts.

Context Aggregation. This component deals with inferring the task-dependent latent variables from the in-context examples such that the downstream prediction becomes conditionally independent of the context, i.e. inferring $\boldsymbol{z}$ from $\mathcal{D}$ such that $p(y_* \mid \boldsymbol{x}_*, \boldsymbol{z}, \mathcal{D}) = p(y_* \mid \boldsymbol{x}_*, \boldsymbol{z})$.

Predictive Modeling. This component refers to the process of estimating the predictive function that leverages context $\mathcal{D}$ to infer y_* from a query $\boldsymbol{x}_*$. In the above example, it refers to learning the functional mapping g once $\boldsymbol{z}$ has been extracted from context aggregation.

As discussed, Transformers do not have a clear incentive to make this explicit separation of context aggregation and predictive modeling. Instead, given context $\mathcal{D}$, they implicitly and jointly model both the function g along with $\mathcal{D}$-dependent latent variable $\boldsymbol{z}$ inference to directly provide predictions for the query $\boldsymbol{x}_*$, in contrast to separately estimating g and explicitly factorizing $\boldsymbol{z}$. Thus, in order to enforce explicit representation of $\boldsymbol{z}$, we propose a simple architectural modification where the query $\boldsymbol{x}_*$ cannot directly attend to the context, and the latent task representation is forced to summarize the context efficiently. Formally, we compare the following two models, which are illustrated in Figure 1.

Implicit Model. This refers to the traditional in-context learning computation performed by Transformer models. In this setup, given the set of observations $\mathcal{D}$ (context) and a query $\boldsymbol{x}_*$, the prediction y_* is modeled directly

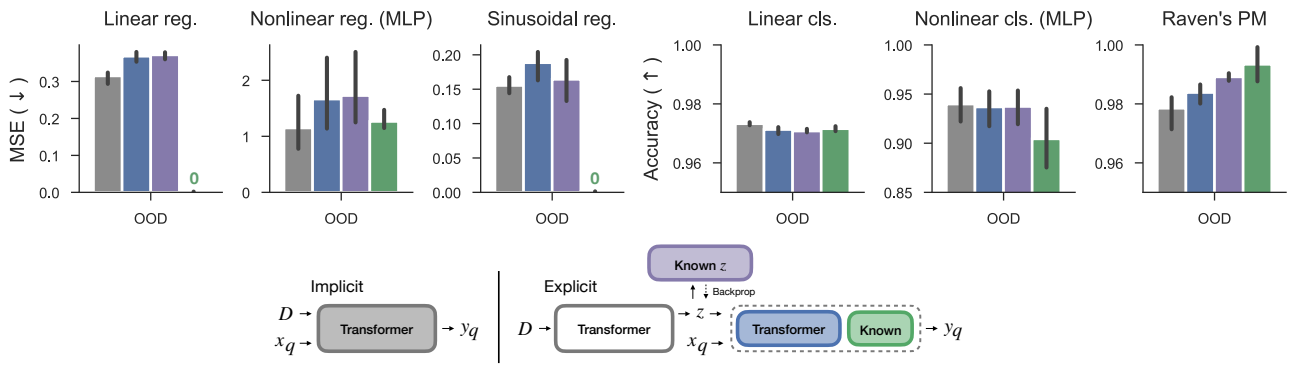

Figure 3. Performance on a subset of tasks where the true latents $\boldsymbol{z}$ and prediction function g are known. Implicit models are in shown gray, explicit models with Transformer prediction in blue, models trained with an auxiliary loss to predict the true latents in purple and those using the true prediction function in green. Using the known prediction function leads to significantly better OOD performance.

as $p_\varphi(y_*|\boldsymbol{x}_*, \mathcal{D})$, where p_φ is defined using a standard Transformer with parameters φ and is tasked with modeling both context aggregation and predictive modeling.

Explicit Model. This represents the architectural variation which minimally modifies the Transformer architecture by separating context aggregation and predictive modeling. It first constructs a task representation $\boldsymbol{z}_\psi(\mathcal{D})$ using the set of observations $\mathcal{D}$ and a context model $\boldsymbol{z}_\psi$ with parameters ψ (*context aggregation*) and another network p_γ to make a prediction for a new point $\boldsymbol{x}_*$ (*predictive modeling*) as $p_\gamma(y_*|\boldsymbol{x}_*, \boldsymbol{z}_\psi(\mathcal{D}))$. A key insight is that the *task latents* are invariant to the queries when modeling prediction. The context model is implemented with a Transformer $\boldsymbol{z}_\psi$ with weights ψ, and for the prediction function p_γ, we study both Transformers and MLPs with weights γ. Importantly, the output of the context model $\boldsymbol{z}_\psi(\mathcal{D})$ is a fixed-size vector with much lower dimensionality than the full context $\mathcal{D}$. This information bottleneck prevents the query $\boldsymbol{x}_*$ from attending directly to the context as in standard Transformers; instead, the context model must summarize $\mathcal{D}$ into underlying generative factors, thus ruling out potential shortcut solutions that bypass latent variable inference.

Implicit vs. Explicit. Assuming Transformers do in fact favour shortcut-based solutions, we first hypothesize when each setup should perform better given different task characteristics. If the data is generated with a parametric model with few underlying parameters $\boldsymbol{z}$ (e.g. a linear model $y = \boldsymbol{z}^T\boldsymbol{x}$), the right predictor can be precisely described using the parameters $\boldsymbol{z}$, making the explicit model better suited. In contrast, if the data is generated with a Gaussian Process (GP), the implicit model should be superior since by construction query prediction computes similarities with all points in the context. In this case, the task latents of GP-based data with RBF kernel are infinite dimensional (i.e. a point in function space), which cannot be captured in the finite-dimensional bottleneck of the explicit model. In general, we should expect the explicit model to be superior when the underlying true model is parametric and low-dimensional, but in case of a non-parametric or very high dimensional parametric model, the implicit model should be better.

Finally, we note that our aim is *not* to construct the best possible explicit model architecture – indeed, more sophisticated ones already exist based on amortized Bayesian inference (Garnelo et al., 2018b; Mittal et al., 2023). Instead, we are interested in investigating potential inductive biases for ICL by *minimally* modifying the standard Transformer architecture to remove certain shortcuts from the space of possible solutions. We leave the design of more performant architectures for future work and refer the readers to Appendix A for a detailed discussion of related work.

4. Experiments

Our goal is to use both synthetic and real tasks that capture the key elements of ICL applications to tease apart the effects of implicit and explicit models on both in-distribution (ID) and out-of-distribution (OOD) generalization.

Task Setup. We conduct experiments across a variety of settings that admit task latents, from synthetic regression and classification to reasoning problems. For reasoning tasks that require compositional knowledge, we consider Raven's Progressive Matrices (Raven's PM) (John & Raven, 2003), Alchemy (Wang et al., 2021), Gene Targeting (Norman et al., 2019) and reusable mixture of experts. A brief description of our tasks is provided below, with a more thorough explanation in Appendix B.

Regression Tasks. We consider different data-generating processes, e.g., linear: prediction $\boldsymbol{z}^T\boldsymbol{x}$ and latents $\boldsymbol{z}$, nonlinear (MLP): prediction with a neural network $g(\boldsymbol{x}, \boldsymbol{z})$ and latents as its weights, sinusoidal: prediction as a sum of sinusoids with different frequencies and the latents as its amplitudes.

Classification Tasks. Akin to the regression problems, we consider a linear and nonlinear (MLP) prediction for classification using an additional sigmoid activation on the output.

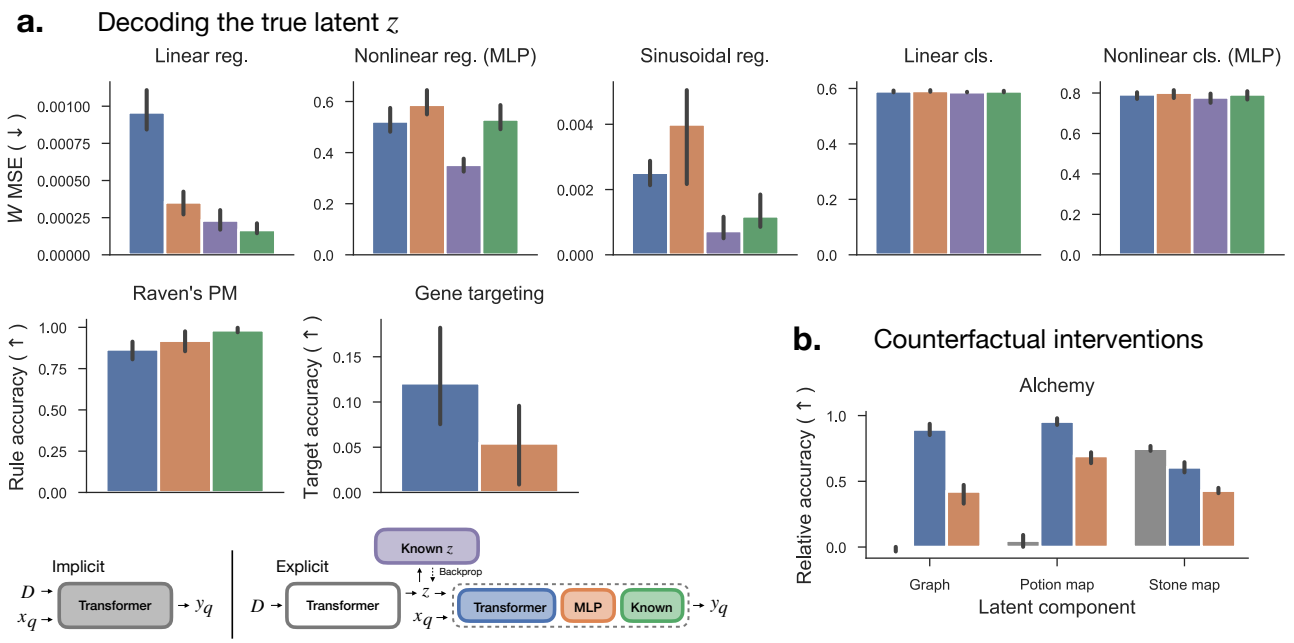

Figure 4. Explicit models are interpretable as the bottleneck allows (a) linearly decoding the true latent, and (b) intervening to obtain correct counterfactual predictions. Implicit models are shown in gray, explicit models with Transformer prediction in blue, and with MLP prediction in orange. Models in green use the true prediction function g, while models in purple use an additional auxiliary loss based on true latents. To evaluate decoding performance in (a), we linearly decode the true latent directly from concatenated context examples, which yields significantly worse performance than decoding from the bottleneck. Baseline performances in units of the plots are – Linear regression: 0.49, Nonlinear regression (MLP): 0.94, Sinusoid regression: 0.33, Linear classification: 0.86, Nonlinear classification (MLP): 0.97, Raven's PM: 0.5, and Gene targeting: 0.0. In (b), the "Relative accuracy" takes the baseline in account (details in C.4).

Raven's Progressive Matrices. A pattern-completion task used in IQ tests where individual object attributes evolve according to different rules. The task is to complete a sequence of frames to satisfy the underlying rule, which is the latent variable and can be based on colors, shapes, etc.

Alchemy. Here, a latent causal graph describes how different stones and potions interact to generate new stones. The task is to infer the next stone given some transitions.

Gene Targeting. It represents a real-world dataset of targeted gene knockouts and subsequent observations of gene expressions across many cells. The underlying latent variable is the set of genes that were knocked out in a given experiment, and the task is to infer the gene expressions of certain cells in an experiment given those of other observed ones.

Reusable Modular Mixture of Experts (MoE). In this task, we apply a sequence of operations g_l on the input $\boldsymbol{x}$, where the choice of expert applied at layer l is governed by a categorical latent $\boldsymbol{z}_l$. In particular, we apply five operations sequentially, leading to $y = g_{\boldsymbol{z}_5} \circ g_{\boldsymbol{z}_4} \circ \ldots \circ g_{\boldsymbol{z}_1}(\boldsymbol{x})$. Here $g_1, g_2, \ldots$ are shared across tasks and each task is uniquely identified by $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_5)$ which are the task latents. This represents a reusable mixture of experts system with a hierarchical and compositional decomposition.

Training and Evaluation. Tasks based on regression utilize the mean-squared error loss, while those based on classification use the cross entropy loss for training. Model architecture details are provided in subsection C.1. For ID evaluation, we consider the underlying task latent $\boldsymbol{z}$, context samples $\mathcal{D}$, and queries $\boldsymbol{x}_*$ to be sampled from the same distribution as during training. The challenge in this case is simply to generalize from finite data. For OOD evaluation, we test two different cases depending on the kind of task. For our synthetic regression and classification tasks, the task latent $\boldsymbol{z}$ and context samples $\mathcal{D}$ are sampled from the same distribution as at training time, but the queries $\boldsymbol{x}_*$ are sampled from a Gaussian distribution with higher ($3\times$) standard deviation, thus testing a form of out-of-domain generalization. For our reasoning-based problems, we evaluate on task latents $\boldsymbol{z}$ that were not seen at training. The task latent in each of these reasoning-based problems is composed of parts (e.g., in the Gene Targeting experiment, the latent is a binary vector of targeted genes), which allows us to test a form of *compositional* generalization (Wiedemer et al., 2023) in which all parts have full marginal support at training time, but novel combinations of those parts are evaluated at test-time.

For all tasks, implicit and explicit models were trained from scratch over a distribution of tasks latents, with a control for the number of parameters to provide a systematic comparison between the two. To better understand the shortcomings of different models, we also compare with

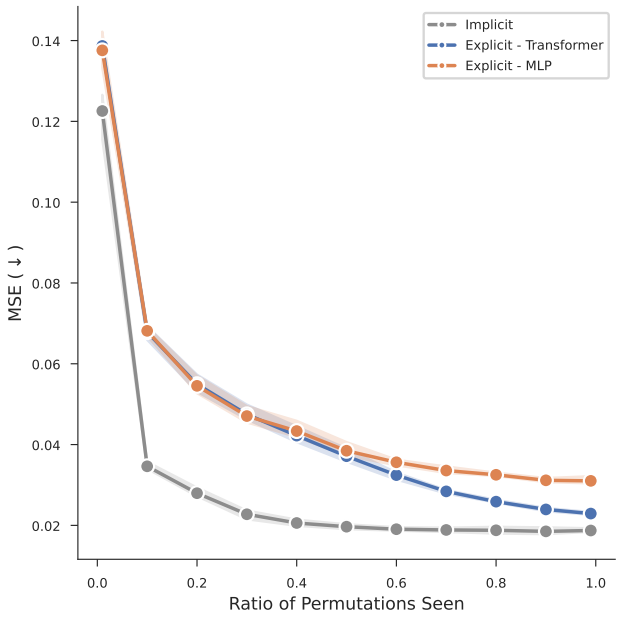

Figure 5. We conduct experiments on reusable modular MoE task where we train on a subset of combinations of experts, shown on the X-axis. Our results indicate that across different percentages of combinations seen during training, the implicit model consistently outperforms the explicit one at compositional generalization.

privileged oracles (known decoder – using ground-truth g function, and known latent variable – using an auxiliary loss that includes the ground-truth $\boldsymbol{z}$).

Explicit models do not outperform the implicit models. The first evaluation setting that we considered was the ID performance. In this case, we should expect both implicit and explicit models to perform equally well, even if implicit models learn shortcuts rather than ground-truth task latents. This is because those shortcuts are tuned to minimize prediction error within the same data distribution that is being evaluated. Across all our tasks, the results indeed confirmed this prediction. Specifically, during ID evaluation, all models were capable of making highly accurate predictions (Figure 2). While the performance of the implicit model was generally slightly better than that of the explicit models, the benefits were marginal. Effectively, all models solved the tasks similarly well.

While we expected that ID evaluation would be insufficient to demonstrate potential benefits of the explicit models, we expected to see differences in OOD settings. Both implicit and explicit models are sufficiently expressive to fit the training distribution. However, if an explicit model successfully learns the true task latents that generated the data while an implicit model learns statistical shortcuts that are specialized to the training distribution, we should expect the explicit model to generalize better OOD. As a reminder, for the synthetic regression and classification tasks in Figure 2 (a, b), OOD evaluation was done by sampling $\boldsymbol{x}_*$ from a normal distribution with wider standard deviation than was used at training ($3\times$), effectively evaluating if the models could extrapolate to points further out along their domain despite only being trained within a narrow distribution near the origin. For the compositional tasks in Figure 2 (c), we instead evaluated OOD performance by only training on certain configurations of the true latent variable $\boldsymbol{z}$ while evaluating on unseen ones. Importantly, at training time the models were shown data that included every possible value for each component of $\boldsymbol{z}$, but not every possible combination of these values were seen, thus evaluating a form of compositional generalization (Wiedemer et al., 2023). Similarly, for the reusable modular MoE task, we train the models on a fraction of all possible combinations in the latent $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_5)$ and evaluate on all combinations.

Surprisingly, and counter to our predictions above, we found that the explicit model provided no significant benefit in OOD settings. In synthetic regression tasks shown in Figure 2 (a), all models failed to generalize and obtained substantially worse performance than during ID evaluation, with the implicit model actually being the one that had a slightly lower degradation in performance. In classification and compositional tasks shown in Figure 2 (b, c), all models generalized fairly well OOD and with similar performance. Our results on reusable modular MoE task in Figure 5 further indicate that implicit models consistently outperform explicit ones across different proportion of latent combinations seen during training (X-axis). In summary, explicit models appear to provide no benefit across our tasks, both in terms of ID and OOD performance.

If the explicit model did learn the right latent variables in the bottleneck, it essentially implies that either the implicit model learns benign shortcuts (if at all) or that learning the right latent variables is not sufficient to improve generalization, both ID or OOD. In the following results, we see that the explicit model does indeed learn the right task latents.

Explicit models learn to infer the correct latent variable, but not how to use it. Why didn't the explicit model provide any benefit? Our initial hypothesis was that the implicit model could be susceptible to learning shortcuts that are sufficient to reduce the training loss and easy to express using self-attention between the query $\boldsymbol{x}_*$ and context $\mathcal{D}$. By summarizing the context in a bottleneck $\boldsymbol{z}_\psi$, the explicit model should instead be forced to extract the true latent variable in order to minimize the training loss, thus learning a solution that is closer to the actual data-generating process. There are then two possible explanations for the results in Figure 2: (1) the explicit models did not learn to extract the true latent variable despite inductive biases to do so induced by the bottleneck, or (2) they did extract the true latent variable but did not learn to use it in the correct way. We performed several experiments to test these two

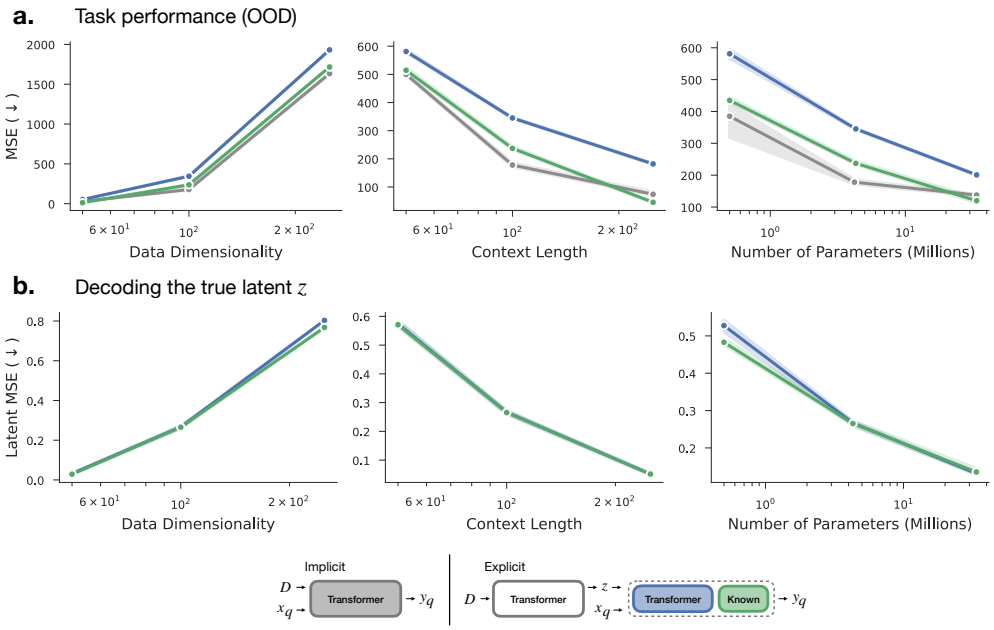

Figure 6. We analyze (a) Linear regression OOD performance and (b) latent variable linear decoding as a function of model and task parameters. Task performance scales similarly for implicit (gray) and explicit models (blue), while latent variable decoding scales similarly for the explicit model and models trained with the true prediction function g (green).

possibilities, and found strong evidence for the second.

To test whether or not the explicit models failed because they did not extract the correct latent variable, we added an additional supervised loss term to Equation 1, $||\boldsymbol{z} - W\boldsymbol{z}_\psi||^2$, to force the explicit model to recover the true task latent $\boldsymbol{z}$ up to a linear transformation, where W is a learnable parameter. The loss of this linear model was then backpropagated through the context model along with the task loss. Results in Figure 3 (purple) show that this auxiliary loss provided no improvement apart from minor increases in accuracy on Raven's PM, suggesting that incorrect latent variable inference does not explain the explicit model's suboptimal performance. Indeed, when we did not use the auxiliary loss as a training signal for the explicit model and just evaluated the quality of the learned task latents by training a separate linear predictor to predict $\boldsymbol{z}_\psi$, we found that we could still accurately predict the true latent variable (see Figure 4 (a)). This means that in the absence of any explicit training signal to predict the true task latent, the explicit model accurately learns task latents up to linear reparameterizations.

Given that the explicit model correctly infers the true latent variable in its bottleneck, we study whether the prediction function is suboptimally learned. In other words, despite the explicit model having access to the true $\boldsymbol{z}$, we hypothesize that $p_\gamma(y_*|\boldsymbol{x}_*, \boldsymbol{z}_\psi)$ does not learn the true data-generating process $y_* = g(\boldsymbol{x}_*, \boldsymbol{z})$, where g is the true prediction function – e.g., for linear regression $g(\boldsymbol{x}_*, \boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{x}_*$. To validate this hypothesis, we trained explicit models with the prediction function g hard-coded as an oracle. For instance, in the linear regression task, the $\boldsymbol{z}_\psi$ output by the context model was linearly projected to the same dimensionality as the true weights $\boldsymbol{z}$, after which the prediction function took its dot product with queried input $\boldsymbol{x}_*$. In this setting, if the explicit model extracts the correct latent variable, it should generalize perfectly both in and out of distribution. Our results in Figure 3 confirm that using the correct prediction function indeed provides substantially better OOD generalization and effectively solves most tasks. This finding has significant implications: it suggests that while learning the true task latents may be a necessary condition for generalization, this must also be supplemented with significant inductive biases in the prediction function – for instance, through novel architectures – so that these task latents can be leveraged correctly. In the absence of such inductive biases, inferring the correct task latent appears to provide no benefits in practice. We do note that our nonlinear regression tasks, where $\boldsymbol{z}$ represents the weights of an underlying MLP generating the data, were an exception to the results described here in that using an oracle prediction function performed poorly. In this case, we conjecture that the underlying latent variable is too difficult to accurately infer from the context, while shortcut-based solutions would avoid latent variable inference altogether to provide robust solutions.

Explicit models are easily interpretable. While explicit models do not provide performance benefits, the ability to extract the correct latent and summarize it in a single bottleneck can still be useful for interpretability. On tasks with known underlying latent variables, we were able to linearly decode them from $\boldsymbol{z}_\psi$ with high accuracy, meaning that the information is not only present but also easily accessible (Figure 4 (a)). The exceptions were complex

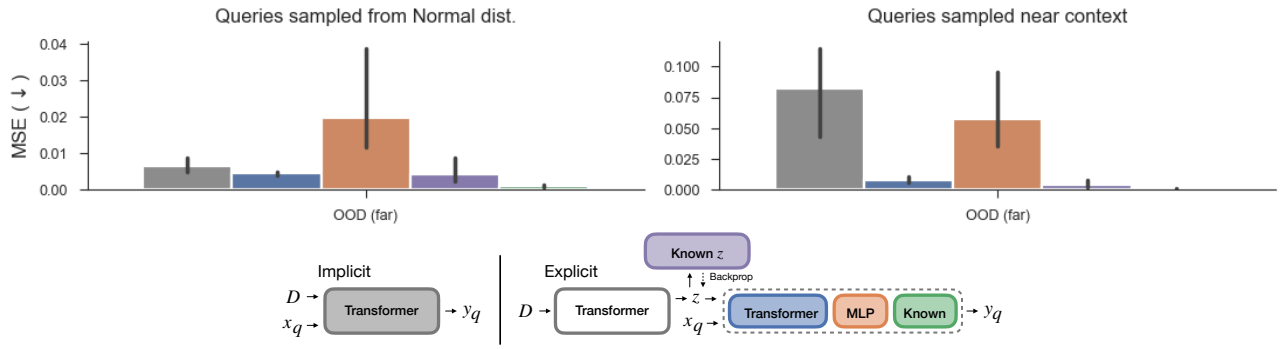

Figure 7. We analyze sinusoid regression by sampling the query either from the normal distribution (*left*) or close to the context (*right*) during training but always far from context during evaluation. We see that when queries are sampled close to the context, implicit models which can rely more on kernel-regression based nearest-neighbor solutions don't generalize far from context, while explicit models do.

nonlinear regression tasks where the latents are difficult to infer and classification tasks where more context samples are needed to precisely identify the decision boundary. In contrast, finding such clear task-relevant latents is immensely challenging in an ordinary Transformer trained to do ICL, given that they can be distributed across a mixture of many layers and token positions.

Furthermore, even when latent variables appear to be successfully identified using a linear decoder in some hidden layer of a Transformer, one often finds that the relationships merely amount to spurious correlations (Ravichander et al., 2021). For instance, if one manually modifies the activations in this hidden layer such that a different latent variable is predicted by the linear decoder, the model's predictions do not generally change in a way that is "counterfactually" consistent with this new latent (i.e., the prediction is not what it should have been under the new latent variable). We therefore used the Distributed Alignment Search (DAS) method from Geiger et al. 2023b (see subsection C.4) to search for units in the implicit and explicit models that can be manipulated to obtain correct counterfactual predictions. For the explicit model, we limited this search to the bottleneck $\boldsymbol{z}_\psi$. We found that using the explicit model, we were indeed able to manipulate $\boldsymbol{z}_\psi$ and obtain correct counterfactual predictions, but we were not able to successfully do this using the implicit model, as shown in Figure 4 (b). Explicit models might therefore be useful for both mechanistic interpretability and scientific discovery (Geiger et al., 2023a), where we do not know the underlying task latents and want to be able to easily uncover them from the trained model, and subsequently obtain a good predictor for an intervened system zero-shot given some knowledge about the intervention.

Scaling Trends across Different Properties. To better compare the implicit and explicit models, we investigated their OOD task performance on linear regression as we varied the different properties of the task (input dimensionality and context length) and the size of the model used (Figure 6 (a)). We found that performance in both models scaled similarly, but that the implicit model reliably outperformed the explicit one unless it used the known prediction function g. We also looked at the latent variable linear decoding accuracy in the explicit model as a function of these task and model properties (Figure 6 (b)). As expected, we found that the latent variable was easier to decode from the explicit model's bottleneck when there was less inherit uncertainty about its value (lower data dimensionality, longer context length) and when the explicit model was given more capacity. However, throughout the different settings, we see that while the explicit model does learn the true latent well, it is not sufficient to get a performance boost over the implicit models. Further details on the setup of these scaling experiments is provided in subsection C.2.

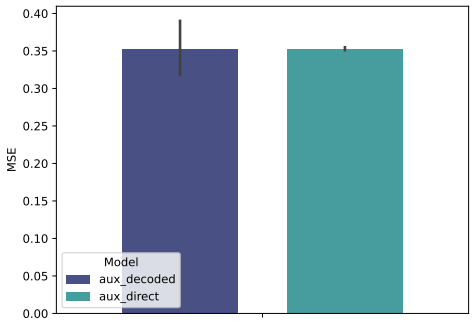

Figure 8. We compare using the auxiliary ground-truth task latent loss directly on the output of context aggregation, i.e. $\boldsymbol{z}_\psi(\mathcal{D})$ (aux_direct), or to a linear decoding from it (aux_decoding).

Impact of auxiliary loss on decoding from bottleneck. Additionally, we perform an experiment where instead of using an auxiliary loss obtained between the ground-truth task latents $\boldsymbol{z}$ and a decoding from the explicit model's bottleneck $||\boldsymbol{z} - W\boldsymbol{z}_\psi||^2$ (called aux_decoded), we instead force the bottleneck itself to be directly close to the ground-truth $||\boldsymbol{z} - \boldsymbol{z}_\psi||^2$ (called aux_direct). Since the prediction function relies on the bottleneck and not its decoding, removing this extra layer when providing additional supervision might allow the bottleneck to better

reflect the task latents and thereby aid prediction. Our results, however, indicate that doing so does not lead to any benefits on OOD evaluation for linear regression, further strengthening the conclusion that effective task latent inference is not the biggest problem in such models.

Extreme Shortcut Injection. Finally, we test whether injecting extreme shortcuts during training pushes implicit models to learn nearest-neighbor styled kernel-regression solutions as opposed to uncovering the underlying functional form. We consider two cases, where queries during training are sampled (a) randomly, or (b) near context points. The latter further incentivizes implicit models to learn nearest neighbour shortcuts. At evaluation, the queries are sampled far from the context. Our results on sinusoid regression in Figure 7 indicate that while implicit models perform well generally, they suffer considerably more in the presence of such injected shortcuts since explicit models distill the task latent from context independent of the query.

We further refer to Appendix D for a detailed analysis.

5. Conclusion

A commonly believed hypothesis is that Transformers do ICL through brittle statistical shortcuts rather than by inferring the underlying generative latent variables of the task, and that this explains their inability to generalize outside of the training distribution. Here, we empirically tested this hypothesis by minimally modifying the Transformer architecture through the use of a bottleneck that factorized the model into separate context aggregation and prediction functions, creating an inductive bias for explicit latent variable inference. While we confirmed that this model indeed learned to infer the correct latent variables across many ICL tasks, it surprisingly gave no improvement in performance for either in-distribution or out-of-distribution evaluation. Contrary to common belief, then, we showed that simply learning the correct latent variables for the tasks is not sufficient for better generalization because end-to-end optimization does not learn the right prediction model to leverage these latent variables.

Impact Statement

This paper provides a comparative analysis to better understand the capabilities of current in-context learners from a point of view of making them more robust and aligned with the true underlying models of the data. We believe that this is an important step towards understanding when and how machine learning models can rely on shortcuts and spurious correlations, and understanding whether such correlations can be mitigated through the use of conditional independence assumptions and bottlenecks, as is investigated in this work.

While we show a negative result that such bottlenecks do not substantially aid generalization, they do come with interpretability benefits which are extremely useful when deploying AI systems at scale. Finally, we hope that our analysis sparks controlled experiments to understand the mechanisms behind in-context learning better as well as incorporating and validating numerous inductive biases to see whether they do aid generalization and reduce the reliance on shortcuts.

Ackonwledgements

The authors would like to acknowledge the following researchers for valuable discussions and exchanges: Joao Sacramento, Johannes von Oswald. All authors acknowledge support from an unrestricted gift from Google inc. EE acknowledges support from Vanier Canada Graduate Scholarship #492702. SM acknowledges the support of PhD Excellence Scholarship from UNIQUE. DS acknowledges support from NSERC Discovery Grant RGPIN-2023-04869, and a Canada-CIFAR AI Chair. GL acknowledges support from NSERC Discovery Grant RGPIN-2018-04821, the Canada Research Chair in Neural Computations and Interfacing, and a Canada-CIFAR AI Chair.

References

Akyürek, E., Wang, B., Kim, Y., and Andreas, J. In-context language learning: Architectures and algorithms, 2024.

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes, 2018.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. A survey of meta-reinforcement learning, 2023.

Bhaskar, A., Friedman, D., and Chen, D. The heuristic core: Understanding subnetwork generalization in pretrained language models, 2024. URL https://arxiv.org/abs/2403.03942.

Bishop, C. *Pattern Recognition and Machine Learning*. Springer, January 2006. URL https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chan, S., Santoro, A., Lampinen, A., Wang, J., Singh, A., Richemond, P., McClelland, J., and Hill, F. Data distributional properties drive emergent in-context learning in transformers. *Advances in Neural Information Processing Systems*, 35:18878–18891, 2022.

Crosbie, J. and Shutova, E. Induction heads as an essential mechanism for pattern matching in in-context learning, 2024. URL https://arxiv.org/abs/2407.07011.

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Garg, S., Tsipras, D., Liang, P., and Valiant, G. What can transformers learn in-context? a case study of simple function classes, 2023.

Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., and Eslami, S. M. A. Conditional neural processes, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. Neural processes, 2018b.

Geffner, T., Papamakarios, G., and Mnih, A. Compositional score modeling for simulation-based inference. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11098–11116. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/geffner23a.html.

Geiger, A., Potts, C., and Icard, T. Causal abstraction for faithful model interpretation, 2023a.

Geiger, A., Wu, Z., Potts, C., Icard, T., and Goodman, N. D. Finding alignments between interpretable causal variables and distributed neural representations, 2023b.

Geiger, A., Wu, Z., Potts, C., Icard, T., and Goodman, N. D. Finding alignments between interpretable causal variables and distributed neural representations, 2024.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, November 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-00257-z. URL http://dx.doi.org/10.1038/s42256-020-00257-z.

Gilbert, L. A., Horlbeck, M. A., Adamson, B., Villalta, J. E., Chen, Y., Whitehead, E. H., Guimaraes, C., Panning, B., Ploegh, H. L., Bassik, M. C., et al. Genome-scale crispr-mediated control of gene repression and activation. *Cell*, 159(3):647–661, 2014.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Grau-Moya, J., Genewein, T., Hutter, M., Orseau, L., Delétang, G., Catt, E., Ruoss, A., Wenliang, L. K., Mattern, C., Aitchison, M., and Veness, J. Learning universal predictors, 2024.

Guo, Y., Hao, Y., Zhang, R., Zhou, E., Du, Z., Zhang, X., Song, X., Wen, Y., Zhao, Y., Zhou, X., Guo, J., Yi, Q., Peng, S., Huang, D., Chen, R., Guo, Q., and Chen, Y. Emergent communication for rules reasoning, 2023.

Hahn, M. and Goyal, N. A theory of emergent in-context learning as implicit structure induction, 2023.

Han, C., Wang, Z., Zhao, H., and Ji, H. Explaining emergent in-context learning as kernel regression, 2023.

Hastie, T. The elements of statistical learning: data mining, inference, and prediction, 2009.

Hendel, R., Geva, M., and Globerson, A. In-context learning creates task vectors, 2023.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey, 2020a.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. Meta-learning in neural networks: A survey, 2020b. URL https://arxiv.org/abs/2004.05439.

John and Raven, J. *Raven Progressive Matrices*, pp. 223–237. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0153-4. doi: 10.1007/978-1-4615-0153-4_11. URL https://doi.org/10.1007/978-1-4615-0153-4_11.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lampinen, A. K., Chan, S. C., Singh, A. K., and Shanahan, M. The broader spectrum of in-context learning. *arXiv preprint arXiv:2412.03782*, 2024.

Lu, S., Bigoulaeva, I., Sachdeva, R., Madabushi, H. T., and Gurevych, I. Are emergent abilities in large language models just in-context learning?, 2024. URL https://arxiv.org/abs/2309.01809.

McCoy, T., Pavlick, E., and Linzen, T. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3428–3448, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1334. URL https://aclanthology.org/P19-1334.

Mittal, S., Bracher, N. L., Lajoie, G., Jaini, P., and Brubaker, M. A. Exploring exchangeable dataset amortization for bayesian posterior inference. In *ICML 2023 Workshop on Structured Probabilistic Inference & Generative Modeling*, 2023. URL https://openreview.net/forum?id=Zt9A5LmNUG.

Nguyen, T. and Grover, A. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling, 2023.

Norman, T. M., Horlbeck, M. A., Replogle, J. M., Ge, A. Y., Xu, A., Jost, M., Gilbert, L. A., and Weissman, J. S. Exploring genetic interaction manifolds constructed from rich single-cell phenotypes. *Science*, 365(6455):786–793, 2019.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Pakman, A., Wang, Y., Mitelut, C., Lee, J., and Paninski, L. Neural clustering processes. In *International Conference on Machine Learning*, pp. 7455–7465. PMLR, 2020.

Pospischil, M., Toledo-Rodriguez, M., Monier, C., Piwkowska, Z., Bal, T., Frégnac, Y., Markram, H., and Destexhe, A. Minimal hodgkin–huxley type models for different classes of cortical and thalamic neurons. *Biological cybernetics*, 99:427–441, 2008.

Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., and Köthe, U. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE transactions on neural networks and learning systems*, 33(4):1452–1466, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ravichander, A., Belinkov, Y., and Hovy, E. Probing the probing paradigm: Does probing accuracy entail task relevance?, 2021.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pp. 1278–1286. PMLR, 2014.

Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?": Explaining the predictions of any classifier, 2016.

Singh, A. K., Moskovitz, T., Hill, F., Chan, S. C. Y., and Saxe, A. M. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation, 2024.

Song, R., Li, Y., Shi, L., Giunchiglia, F., and Xu, H. Shortcut learning in in-context learning: A survey, 2024. URL https://arxiv.org/abs/2411.02018.

Tang, R., Kong, D., Huang, L., and Xue, H. Large language models can be lazy learners: Analyze shortcuts in in-context learning. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, 2023a. doi: 10.18653/v1/2023.findings-acl.284. URL http://dx.doi.org/10.18653/v1/2023.findings-acl.284.

Tang, R., Kong, D., Huang, L., and Xue, H. Large language models can be lazy learners: Analyze shortcuts in in-context learning. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 4645–4657, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.284. URL https://aclanthology.org/2023.findings-acl.284/.

Tejero-Cantero, A., Boelts, J., Deistler, M., Lueckmann, J.-M., Durkan, C., Gonçalves, P. J., Greenberg, D. S., and Macke, J. H. Sbi – a toolkit for simulation-based inference, 2020.

Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models, 2023.

Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. Function vectors in large language models, 2024.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vettoruzzo, A., Bouguelia, M.-R., Vanschoren, J., Rögnvaldsson, T., and Santosh, K. Advances and challenges in meta-learning: A technical review, 2023.

Vig, J., Gehrmann, S., Belinkov, Y., Qian, S., Nevo, D., Sakenis, S., Huang, J., Singer, Y., and Shieber, S. Causal mediation analysis for interpreting neural nlp: The case of gender bias, 2020.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 35151–35174. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/von-oswald23a.html.

Wang, J. X., King, M., Porcel, N., Kurth-Nelson, Z., Zhu, T., Deck, C., Choy, P., Cassin, M., Reynolds, M., Song, F., Buttimore, G., Reichert, D. P., Rabinowitz, N., Matthey, L., Hassabis, D., Lerchner, A., and Botvinick, M. Alchemy: A benchmark and analysis toolkit for meta-reinforcement learning agents, 2021.

Wang, L., Li, L., Dai, D., Chen, D., Zhou, H., Meng, F., Zhou, J., and Sun, X. Label words are anchors: An information flow perspective for understanding in-context learning, 2023.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners, 2022. URL https://arxiv.org/abs/2109.01652.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Wiedemer, T., Mayilvahanan, P., Bethge, M., and Brendel, W. Compositional generalization from first principles, 2023.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference, 2022.

Yang, L., Lin, Z., Lee, K., Papailiopoulos, D., and Nowak, R. Task vectors in in-context learning: Emergence, formation, and benefit. *arXiv preprint arXiv:2501.09240*, 2025.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhang, H., Li, L. H., Meng, T., Chang, K.-W., and den Broeck, G. V. On the paradox of learning to reason from data, 2022.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization, 2023.

A. Related Work

In-Context Learning. In-Context learning (ICL) is an ability of certain trained models to take an entire task's dataset as input and parameterize solutions directly in their layer activations, which then condition subsequent computation on novel inputs from those same tasks. Generally, this ability is found in sequence models such as Transformers where the task dataset, or "context", corresponds to an earlier part of the sequence. ICL was first observed in large-scale pre-trained LLMs (Brown et al., 2020), and is similar in many respects to meta-learning (Chan et al., 2022). These LLM findings were subsequently expanded to more controlled settings outside of the language modality, where Transformer models were directly trained on task distributions such as linear regression (Von Oswald et al., 2023; Garg et al., 2023), hidden Markov models (Xie et al., 2022), compositional grammars (Hahn & Goyal, 2023), regular languages (Akyürek et al., 2024) and Turing machines (Grau-Moya et al., 2024), with a set of task observations defining the "context". These works highlight that Transformers are indeed able to model many types of complex task distributions, approaching in many cases the performance of the Bayes-optimal predictor, the a-priori optimal solution (Xie et al., 2022). Our work lies along similar lines but using more complex tasks and a systematic study into the differences between modeling the predictive space directly, or through a two-step process involving explicit inference of task latents.

Shortcuts in ICL. Shortcut learning is a phenomenon that has widely been observed in machine learning (Geirhos et al., 2020), and refers to where a model solves a task through statistical correlations that are accidental and thus not robust to even slight distribution shifts. A classical example of this in image classification is the usage of background cues to classify objects (Ribeiro et al., 2016). Similar mistakes are know to be very common in NLP (McCoy et al., 2019), specifically in reasoning tasks (Zhang et al., 2022). Particularly relevant to our work, many authors have shown that has shown that Transformers are very prone to relying on shortcuts when performing ICL (Tang et al., 2023a). For instance, (Olsson et al., 2022; Singh et al., 2024) have shown that *induction heads* play in important role in ICL by predicting that the continuation of a token will be the same as last time (i.e. $[a][b]\dots[a] \rightarrow [b]$). As shown by (Von Oswald et al., 2023) this motif can be used to do linear regression, and can generally be seen as a form of kernel regression (i.e. $p(y_q|x_q, x_{1:n}) \propto \sum_i K(x_i, x_q)y_i$, (Han et al., 2023)). This observation draws a link between those types of solutions and non-parametric inference methods in statistics (Hastie, 2009), of which kernel regression is a member. In contrast (Hendel et al., 2023; Todd et al., 2023) have concurrently shown that in some cases Transformers encode a "task vector" that they infer from the context and then use to do the prediction. There is therefore a need to better understand the nature of shortcuts in ICL and whether or not they can be easily avoided for better generalization. Our work explores this very question.

Neural Processes. The problem of solving new tasks in a zero-shot manner directly at inference is also closely tied to amortized Bayesian models (Kingma & Welling, 2013; Rezende et al., 2014; Radev et al., 2020; Geffner et al., 2023; Mittal et al., 2023). Conditional Neural Processes (CNPs) (Garnelo et al., 2018a) provide a framework akin to the explicit model, where the posterior predictive distribution is modeled through a bottleneck $\boldsymbol{z}_\psi$, i.e. $p_{\boldsymbol{\theta}}(y_*|\boldsymbol{x}_*, \mathcal{D}) = p_{\boldsymbol{\theta}}(y_*|\boldsymbol{x}_*, \boldsymbol{z}_\psi(\mathcal{D}))$. However, CNPs do not look at the relevance of $\boldsymbol{z}_\psi$ to the true latent $\boldsymbol{z}$, and use the DeepSets (Zaheer et al., 2017) architecture to model $\boldsymbol{z}_\psi$, though recent research generalizes this setting to use Transformers (Nguyen & Grover, 2023) and other architectural backbones as well (Kim et al., 2019; Gordon et al., 2019). Our approach with the explicit model, however, is to precisely question whether task-specific latents are encoded via $\boldsymbol{z}_\psi$ which is now instead modeled using a Transformer architecture. Analogously, Neural Processes (NPs) (Garnelo et al., 2018b; Pakman et al., 2020) augment CNPs with probabilistic modeling, where $\boldsymbol{z}$ is now modeled explicitly as a latent-variable in the Bayesian sense, i.e. the likelihood is now modeled as $p_{\boldsymbol{\theta}}(y|\boldsymbol{x}_*, \mathcal{D}) = \int p_{\boldsymbol{\theta}}(y|\boldsymbol{x}, \boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z}|\mathcal{D})d\boldsymbol{z}$, where $\boldsymbol{z}$ represents the latent variable and $\boldsymbol{\theta}$ the parameters of the likelihood model. The model is trained via the Evidence Lower-Bound (ELBO) with the amortized variational approximation $q_\varphi(\cdot|\mathcal{D})$. Once trained, predictions for new datasets can be made by simply performing inference over the *encoder* q_φ to obtain $\boldsymbol{z}$, and then leveraging this latent variable to eventually give the predictions via $p_{\boldsymbol{\theta}}(y|\boldsymbol{x}_*, \boldsymbol{z})$. Hence, while CNPs and the explicit model to share similarities in architecture, our goal is orthogonal in that we specifically use the explicit model to understand the impact of task-specific latent variable inference on ICL setups.

Meta-Learning. Meta-learning (Hospedales et al., 2020b;a) studies systems that can learn over two levels: rapidly through an inner-loop that is meta-learned using a slower outer-loop. The goal in such methods is to learn a good initialization common to the parameterized family of tasks, in a manner that obtaining a particular solution for a new task is fast from this initial point. The inner loop provides an optimization trajectory for a randomly drawn task from some initialization, which in itself is optimized in the outer loop to a good solution applicable for the global set of tasks. Typically, evaluation is done on some meta-validation set of tasks not seen during training. Task distributions can for example be a set of different classification/regression tasks (few shot learning, (Vettoruzzo et al., 2023)) or variations of a reinforcement learning (meta-RL, (Beck et al., 2023)). The goal is similar to ICL approaches in the sense that given a novel context $\mathcal{D}$,

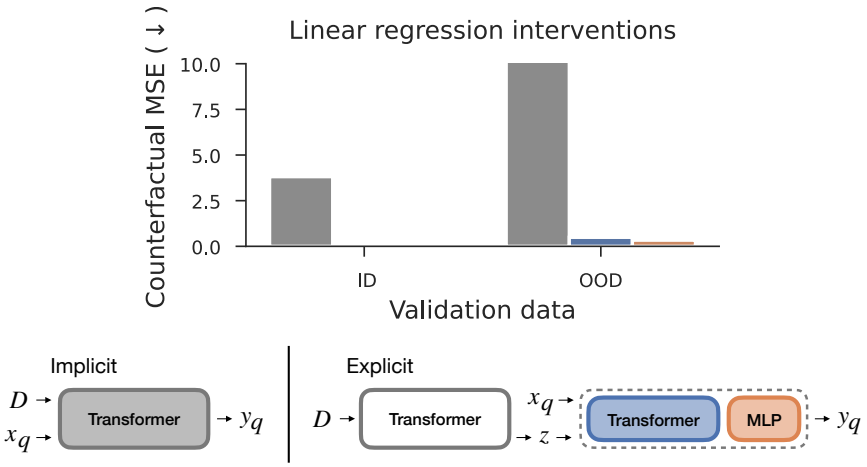

Figure 9. Same experiment as Figure 4b but with the linear regression task. Specifically, we use Distributed Alignment Search (DAS, (Geiger et al., 2023b), see Appendix C for details) to find the 10 dimensional subspace in each model with best simulates counterfactual interventions on the task vector (in this case, the weight of the linear regression). In both explicit model, the subspace is taken at the bottleneck. In the implicit one, we perform the DAS at all layer of the query token and report the best one. The reported metric is the MSE of the intervened model on the intervened regression problem (i.e. using the same query x but y's coming from the intervened linear regression weights, Appendix C for details).

one wants to make predictions for some query $\boldsymbol{x}_*$. However, a big difference is that ICL approaches bypass modeling a common initialization by working directly on the prediction side (implicit), or instead predict the optimal parameters directly zero-shot through inference on the context model (explicit).

Mechanistic Interpretability. Mechanistic interpretability is interested in understanding deep neural network's computations through interpretable abstraction, akin to what computational neuroscience does with the brain. (Alain & Bengio, 2018) introduced the foundational technique of linear "probes", which are linear models trained on the hidden state of a network to predict an abstract feature of the input; the success of which suggests that such a feature is used by the model. Since then, this naïve approach has been criticized for being potentially misleading (Ravichander et al., 2021); in many cases a feature can be linearly decoded from a model without the model using it. More reliable methods grounded in causality (Vig et al., 2020; Geiger et al., 2024) have now became the gold standard, and their use applied to Transformers has been exploding in popularity (Elhage et al., 2021).

B. Tasks

We consider the following tasks for our evaluations, specified by the data-generating prediction function $g : \boldsymbol{x}, \boldsymbol{z} \to y$ which is used to generate the ICL dataset, where $\boldsymbol{z}$ represents the task-specific latent variable.

B.1. Regression Tasks

For regression tasks, since $y \in \mathbb{R}$, we use the mean-squared-error loss to train the model.

Linear Regression. This refers to the task where y is obtained from an affine transformation on the input $\boldsymbol{x}$. In particular, $\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{z}^T\boldsymbol{x}$, where $\boldsymbol{z} \in \mathbb{R}^{i\times j} \sim \mathcal{N}(0, I)$. For our experiments, we set $\dim(\boldsymbol{x}) = 1$ and $\dim(\boldsymbol{y}) = 1$.

Nonlinear Regression using MLPs. Here, the labels y are obtained from a neural network which takes $\boldsymbol{x}$ as an input. In particular, $\boldsymbol{y} = g(\boldsymbol{x}, \boldsymbol{z}) = f_{\boldsymbol{z}}(\boldsymbol{x})$, where $f_{\boldsymbol{z}}$ is modeled as a Multi-Layer Perceptron (MLP) network with a 64 dimensional single hidden layer and ReLU nonlinearity. The distribution of the weights of the neural network is $\boldsymbol{z} \sim \mathcal{N}(0, I)$. For our experiments, we set $\dim(\boldsymbol{x}) = 2$ and $\dim(\boldsymbol{y}) = 1$.

Sinusoid Regression. For this task, the label y is obtained as a summation of sine functions with different frequencies and amplitudes, taking $\boldsymbol{x}$ as an input. Mathematically, we structure the system as $y = g(\boldsymbol{x}, \boldsymbol{z}) = \sum_{i=1}^{K} \alpha_i \sin(2\pi\lambda_i\boldsymbol{x})$, where λ_i's denote the frequencies and α_i's the amplitudes. The parameters for the system can be seen as $\boldsymbol{z} = \{\alpha_{1:K}\}$ while $\lambda_{1:K}$

remains fixed throughout. Additionally, for our experiments we set $K = 3$, and consider the distributions – $\lambda_i \sim \mathcal{U}(0, 5)$ and $\alpha_i \sim \mathcal{U}(-1, 1)$, and set $\dim(\boldsymbol{x}) = 2$ and $\dim(y) = 1$.

Gaussian Process Regression. While the other tasks considered had a parametric nature to it, this task on the other hand has more of a non-parametric nature. Here, the task is that $\boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{0}, K(\boldsymbol{X}, \boldsymbol{X}))$, i.e. the set of labels is sampled from a joint Gaussian distribution, akin to drawing a random function through a Gaussian Process (GP) prior and then evaluating it at different points $\boldsymbol{X}$; with K defining the kernel in the GP. In our case, we consider $K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right)$ as the RBF kernel and $\boldsymbol{X} = (\boldsymbol{X}_c, \boldsymbol{X}_q), \boldsymbol{Y} = (\boldsymbol{Y}_c, \boldsymbol{Y}_q)$ are the combined points for both the context and the queries, which are split after this sampling. Here the latents $\boldsymbol{z}$ has to store the kernel computations between the query and all the context points $\boldsymbol{X}_c$, as well as the corresponding context labels $\boldsymbol{Y}_c$. Storing this either involves storing the high-dimensional mapping of $\boldsymbol{X}_c$ which is defined by the kernel K, or storing all the points $\boldsymbol{X}_c$ themselves. This is thus very high dimensional and weakly structured.

Hodgkin-Hoxley ODE Prediction. This is an example of the task where the context $\mathcal{D}$ is not composed of *iid* entries, but instead observations from the Hodgkin-Huxley temporal dynamics model of neural activity unrolled through time :

$$C_m \frac{dV}{dt} = g_1 (E_1 - V) + \bar{g}_{Na} m^3 h (E_{Na} - V) + \bar{g}_K n^4 (E_K - V) + \bar{g}_M p (E_K - V) + I_{inj} + \sigma\eta(t)$$

Above, V represents the membrane potential which is the target of interest, $\boldsymbol{t}$ represents the different points at which observations are provided, C_m is the membrane capacitance, g_{l} is the leak conductance, E_{l} is the membrane reversal potential, $\bar{g}_c$ is the density of channels of type c (Na^+, K^+, M), E_c is the reversal potential of c, (m, h, n, p) are the respective channel gating kinetic variables, and $\sigma\eta(t)$ is the intrinsic neural noise. The right hand side of the voltage dynamics is composed of a leak current, a voltage-dependent Na^+ current, a delayed-rectifier K^+ current, a slow voltage-dependent K^+ current responsible for spike-frequency adaptation, and an injected current I_{inj}. Channel gating variables q have dynamics fully characterized by the neuron membrane potential V, given the respective steady-state $q_\infty(V)$ and time constant $\tau_q(V)$ (details in (Pospischil et al., 2008)).

Importantly, in our experiments, we fix all parameters but $(\bar{g}_{Na}, \bar{g}_K)$ to values in Tejero-Cantero et al. 2020 and solve the differential equation for 6,400 pairs $(\bar{g}_{Na}, \bar{g}_K) \in [0, 40]^2$ from $t = 0$ to $t = 120$ with 1000 time-steps. In other words, the Transformer has to regress to solutions of ordinary differential equations, where the task latents are $\boldsymbol{z} = \{\bar{g}_{Na}, \bar{g}_K]\}$, the observations are $\boldsymbol{x} = \boldsymbol{t}$ and $y = V$, such that $y = g(\boldsymbol{x}, \boldsymbol{z})$. Here g represents the unrolling of the differential equation.

B.2. Classification Tasks

For classification tasks, since y is a categorical measure, we use a cross-entropy loss for training.

Linear Classification. Akin to linear regression, here we consider the case that y is obtained by an affine transformation of $\boldsymbol{x}$ followed by a sigmoid function and a consequent sampling step. That is, $y = g(\boldsymbol{x}, \boldsymbol{z}) \sim \text{Categorical}(\text{Softmax}(\boldsymbol{z}^T \boldsymbol{x}))$ where $\boldsymbol{z} \in \mathbb{R}^{i \times j} \sim \mathcal{N}(0, I)$. For our experiments, we set $\dim(\boldsymbol{x}) = 2$ and $y \in \{0, 1\}$.

Nonlinear Classification Using MLPs. Here, the logits for the labels are instead obtained through a neural network taking $\boldsymbol{x}$ as an input, and not an affine transformation. Mathematically, this can be seen as $y = g(\boldsymbol{x}, \boldsymbol{z}) \sim \text{Categorical}(\text{Softmax}(f_{\boldsymbol{z}}(\boldsymbol{x})))$ where $f_{\boldsymbol{z}}$ is modeled as a Multi-Layer Perceptron (MLP) network with a 64 dimensional single hidden layer and ReLU nonlinearity. The distribution of the weights of the neural network is $\boldsymbol{z} \sim \mathcal{N}(0, I)$. For our experiments, we set $\dim(\boldsymbol{x}) = 2$ and $y \in \{0, 1\}$.

B.3. Compositional tasks

Reusable Modular Mixture of Experts (MoE). We consider a modular task which consists of sequential application of a choice of K experts $g_1, \ldots g_K$ over the input $\boldsymbol{x}$. In particular, the computational graph consists of L layers where at each layer l, expert $\boldsymbol{z}_l$ operates on the output of the preceding layer to give the successive output, i.e. $\boldsymbol{x}^l = g_{\boldsymbol{z}_l}(\boldsymbol{x}^{l-1})$. This task is compositional in nature because at each layer, any of the K experts can be called upon to perform a unit of computation and the choice of the expert is defined by the underlying task latent $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_L$, each of which are categorical with K possibilities. In our specific implementation, we set L to be 5, K to be 5, $\boldsymbol{x} \in \mathbb{R}^4$ and $\boldsymbol{y} \in \mathbb{R}^4$. Each expert g_i is parameterized as a linear layer followed by the tanh activation function. We enumerate all K^L possible combinations and then only use a subset of them during training, while randomly sampling all for evaluation.

Alchemy. Alchemy is a meta-reinforcement learning benchmark (Wang et al., 2021) where each environment is defined

by a set $\mathbf{z} = (\text{GRAPH}, \text{POTION MAP}, \text{STONE MAP})$ of rules about how some set of potions transforms some stones. We extracted from it an ICL classification dataset consisting of transformations $\mathbf{x} = (\text{STONE}, \text{POTION}) \rightarrow \mathbf{y} = \text{STONE}$. The transformations are compositional and symbolic; each potion affects only one of the three properties of stones (size, shape and color). An environment is specified by how observable stones and potions MAP to latent stones and potions, along with a GRAPH over these latent stones which specify the result of the Transformations. In total there is 109 GRAPH, 48 POTION MAP and 32 STONE MAPS, making for 167424 environments. We reserve 100,000 environments for evaluation and train of the remaining ones.

Raven's Progressive Matrices. Raven's Progressive Matrices (Raven's PM) is a reasoning task used for IQ tests (John & Raven, 2003). It consists of a 3x3 grid where each cell contains simple objects varying in a small number of attributes (number, shape, size, color), but the bottom right cell is left empty. Subjects must notice a pattern in how the cells change from left to right in the first two rows of the grid, and then use that same pattern to complete missing cell in the bottom row. This is done by selecting one answer among N possible provided options for the missing cell. We use a symbolic version of the dataset that addresses bias in the original version (Guo et al., 2023). In this dataset, objects at a cell have 4 discrete attributes with 40 possible values each. In our models, the context consists of the first two rows of the grid, the query consists of the last row with a masked out final cell, and the ground-truth latent variable is the underlying rule that generates a particular grid. Each rule is composed of a set of sub-parts, and we evaluate on unseen compositions.

Gene Targeting. We use Perturb-seq dataset collected by Norman et al. (2019) where researchers performed several genetic intervention experiments using CRISPR (Gilbert et al., 2014). In each experiment, either one or two genes were targeted and the resulting expressions across 5000 genes were observed across several cells. Here, we consider each CRISPR intervention experiment as a different context, the resulting cell genetic expressions as 5000-dimensional observations, and a left-out cell with half of the genetic expressions randomly masked out as the query. The task is to predict the missing genetic expressions for the queried cell. We evaluate on held on held out CRISPR experiments with novel pairs of targeted genes.

C. Model Details

In the following section, we describe the standard architectural details used for all the tasks, as well as specific differences in the architecture used for the scaling experiments. Finally, we also provide details about the distributed alignment search mechanism.

C.1. General Details

For our implicit model, we use a standard Transformer with 8 layers. In the explicit model, for context aggregation we parameterize $\boldsymbol{z}_\psi(\mathcal{D})$ using a standard Transformer with 4 layers, 256 dimensions latent, 512 dimensions MLP, and 4 heads. For the predictor p_γ, we consider two options: a ReLU-actiavtion based MLP with three hidden layers of size 512 and a Transformer with the same configuration as $\boldsymbol{z}_\psi(\mathcal{D})$.

For the implicit model, we format the prompt for prediction as $[\boldsymbol{x}_1, y_1] \dots [\boldsymbol{x}_n, y_n][\boldsymbol{x}_q, \emptyset]$, where every $[\cdot]$ represents a token. We use a distinct mask token $\emptyset$ to represent the target (which is the thing being predicted). For the explicit model, we first compute $[\boldsymbol{x}_1, y_1] \dots [\boldsymbol{x}_n, y_n]$ to $\boldsymbol{z}_\psi(\mathcal{D})$ with the context Transformer, then we give $[\boldsymbol{z}_\psi(\mathcal{D})][\boldsymbol{x}_q]$ to the predictor Transformer or $[\boldsymbol{z}_\psi(\mathcal{D}), \boldsymbol{x}_q]$ to the MLP.

For our experiments, the number of context points n is uniformly sampled from 16 to 128 for both training and evaluation. Training is done with new data being synthetically generated on the fly, and evaluation either based on the test set provided for real-world tasks or simulated data of 1000 different contexts for synthetic tasks. All the models were trained with a learning rate of 10^{-4} using the Adam optimizer (Kingma & Ba, 2014) for a 1000 epochs.

C.2. Scaling Experiments

For the scaling experiments, we only consider the linear regression case with a base configuration of: (a) $\boldsymbol{x}$ of dimensionality 100, (b) context size being sampled uniformly from $(75, 125)$, and (c) 8 heads, 8 layers, 512 hidden dimensions and 256 bottleneck dimension for the transformer models.

From this base configuration, we changed only one of the configurations at each time to test for scaling trends for each property independently. In particular, we ablated over $(50, 100, 250)$ for the dimensionality of $\boldsymbol{x}$, $(50, 100, 250)$ for the context length which was sampled from a ± 25 range and the model size. The smallest model size considered had 4 heads, 4

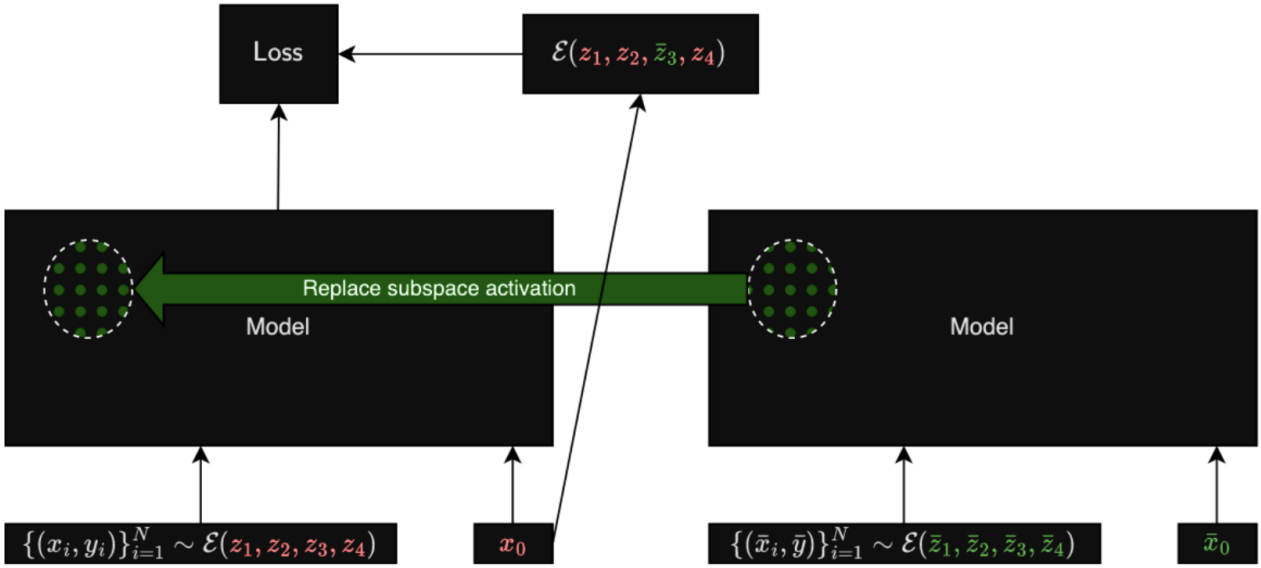

Figure 10. Illustration of the DAS training procedure

layers, 256 hidden dimensions and 128 feature dimensions. The medium multiplied each of these properties by $2\times$, and the biggest model subsequently multiplied it by $2\times$ again. For the explicit models, we considered the same scaling paradigms with the number of layers being split by half to accommodate a separate context model and prediction model.

All the models were trained with a learning rate of 10^{-5} using the Adam optimizer for 5000 epochs.

C.3. Compute Details

We train most of our models on single RTX8000 NVIDIA GPUs, where it takes roughly 3-6 hours for each experiment to run. Our scaling experiments on the other hand often required 1-2 days on single GPUs for training each model.

C.4. Distributed Alignment Search details

To find subspace causally associated with a task latent in Alchemy, we use a method based on Distributed Alignment Search (DAS) by (Geiger et al., 2023b). This procedure is performed for a location $L = \mathbb{R}^d$ (e.g. the bottleneck) and latent $i \in \{1, 2, 3\}$ (GRAPH, STONE MAP, POTION MAP).

First, we run with the model on $\mathcal{D}_z$ and $\mathcal{D}_{\bar{\boldsymbol{z}}}$ for every possible query x_*. We call $\boldsymbol{z}$ the base and $\bar{\boldsymbol{z}}$ the source and only differ by the ith latent. For every run, we record the activity of the source model at the location $l_z \in \bar{L}$. Then, we run the base model again but this time fixing the subspace of l defined by the orthogonal projection $\Pi \in \mathbb{R}^{d \times 10}$ to it's value in l_z. A single projection Π is learned over all possible combination $\boldsymbol{z}, \bar{\boldsymbol{z}}$ and x_* with a cross-entropy loss between the prediction of the base (intervened) model and the true counter-factual result of changing the latent z_i to $\bar{\boldsymbol{z}}_i$. See Figure 10 for an illustration of the process. A subspace is evaluated by looking at the accuracy of the counterfactual interventions over a dataset of held-out $\boldsymbol{z}, \bar{\boldsymbol{z}}$ pairs; a quantity called the Interchange Intervention Accuracy (IIA). In Figure Figure 4 (b) we report the validation IIA relative to a baseline corresponding to the counterfactual accuracy if we don't perform any intervention (because changing the latent sometimes doesn't change the prediction) $\frac{\text{IIA} - \text{BASELINE}}{1 - \text{BASELINE}}$.

D. Analysis of Experiments

Based on the empirical evidence presented in section 4, we finally provide details and analysis into the results to further the understanding of the conclusions. In particular, our key analysis includes

Explicit Models sufficiently uncover task latents. We see that in problems where the context provides enough evidence

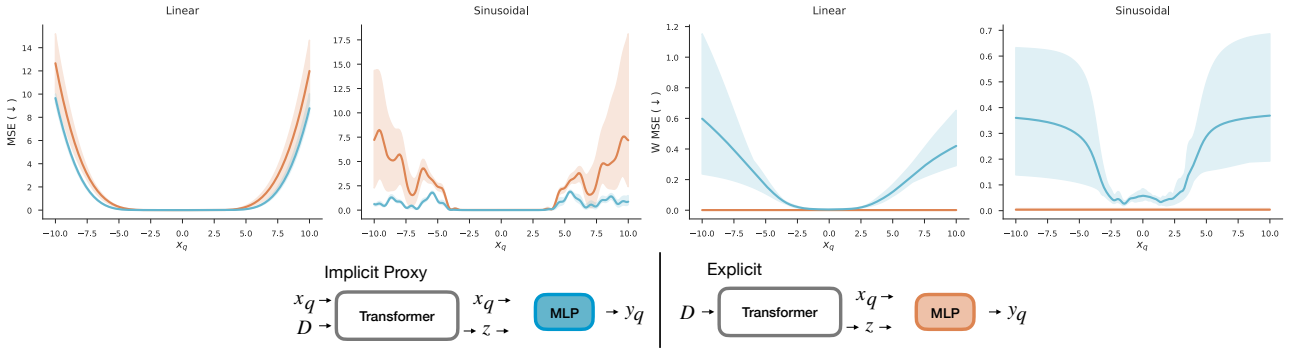

Figure 11. To further understand the difference between the explicit and implicit model, we utilize an implicit proxy model which shares the same architecture as the explicit model with MLP prediction with just *one key difference*: the task latent $\boldsymbol{z}$ depends on the query $\boldsymbol{x}_q$ as well. This task latent $\boldsymbol{z}$ can be understood as the final attention layer output of the implicit model, after which an additional MLP is utilized to provide prediction. Our findings on linear and sinusoidal regression demonstrate that as we move further and further out-of-distribution, the implicit proxy model performs better than the explicit model (left figures), but recovers the underlying true task latents worse (two right figures). This provides additional validation of our hypothesis.

to uncover the true task latents, explicit models are able to do so. In particular, this hints at the fact that explicit models do perform downstream prediction based on true task latents whenever these latents can be sufficiently identified from the context examples.

Explicit Models do not generalize better than implicit ones. Our analysis also reveals that while explicit models often do uncover the right task latents, they are still not able to surpass implicit models even on OOD generalization. This could be due to implicit models also uncovering the true underlying prediction function but in a distributed fashion, or explicit models not being able to leverage the learned latents in downstream prediction.

Learned downstream prediction is often sub-optimal. Our results indicate that it is indeed the case that while the explicit models do uncover the right latents, they fail to generalize well OOD because the downstream prediction function fails to generalize.

This is further strengthened by Figure 11 where additionally leverage the query in context aggregation, thus interpolating between explicit and implicit model while maintaining a bottleneck. Our results indicate that despite worse latent variable inference far from the query, it still has better predictive generalization when compared to explicit models. This further strengthens the claim that better latent variable inference is not the sole problem, and learning the right downstream prediction is as important.

Classification tasks vs. regression tasks. OOD performance is generally strong (across all models) for classification because decision boundaries are within the training domain and do not change beyond it. In contrast, for regression tasks, the function continues to change beyond the observed training domain, making OOD prediction more difficult. This is also why known prediction functions give little benefit in classification tasks: they are already solved well OOD with ordinary implicit and explicit models.

The explicit model with known prediction function does not give benefits in nonlinear (MLP) regression. This is because the problem of inferring an MLP's weights given some context examples is too difficult, so the explicit model opts for a different, non-parametric solution. This is supported by the latent variable decoding results in Fig. 5 (previously Fig. 4), which show that even with a known prediction function the explicit model does not learn to infer the correct latent variable for the nonlinear (MLP) regression task.

Impact of Output Dimensionality. In addition to our standard experiments, we consider linear regression with varying output dimensionality as another measure of difficulty of the task. Our results in Figure 14 showcase that the implicit model fares much better than the explicit model, however having a known prediction function leads to much better performance.

Size of Context Aggregator. While we primarily controlled for the total number of parameters, it is possible that context aggregation requires more parameters and thus equally splitting parameters between this aggregation and prediction may be the cause for suboptimal performance. To study this properly, we conduct experiments where the context aggregator is the

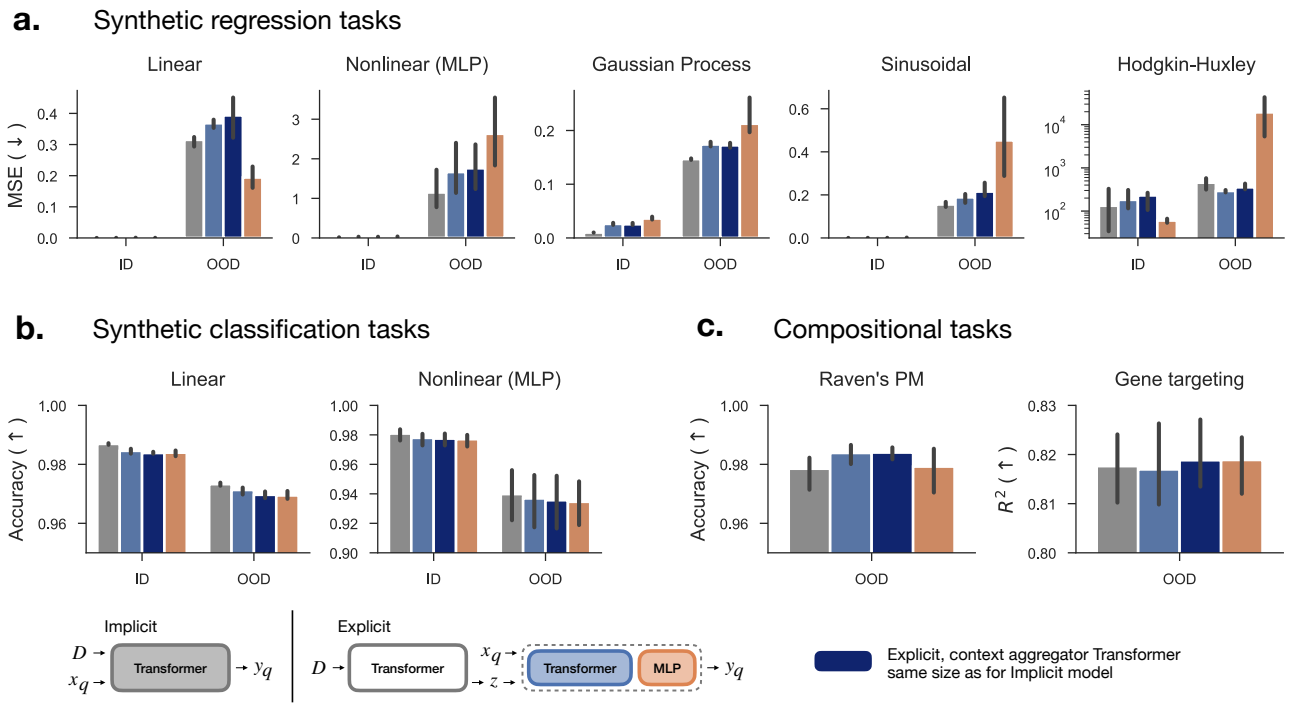

Figure 12. Similar to Figure 2, but with an additional explicit model where the context aggregator is identical in size and hyperparameters (M) as the implicit model. Implicit models are in shown gray, explicit with Transformer prediction in blue (light blue = total model size identical to implicit as in main paper, dark blue = context aggregator identical to implicit), and with MLP prediction in orange.

same size as the implicit model and we see similar results in Figures 12 and 13. In addition, we ablate with differently sized context aggregators and prediction models for different tasks in Tables 1 to 8

Visualization of Explicit Predictions. We visualize the failure of explicit model in learning the right prediction function, which can be seen in out of distribution for sinusoid regression in Figure 15.

E. Mathematical Formalism

In this section, we provide a formal distinction between the implicit and explicit model. In both the approaches, the goal is to model the true posterior predictive $p(y|\boldsymbol{x}, \mathcal{D})$; however the two methods model it through different conditional independence setup.

Implicit Model. In this setup, we model the predictive distribution as $p_\varphi(y|\boldsymbol{x}, \mathcal{D})$, where the training is done as

$$\arg\max_{\varphi} \mathbb{E}_{\boldsymbol{x},y,\mathcal{D}} \left[\log p_\varphi(y|\boldsymbol{x}, \mathcal{D})\right] \tag{2}$$

and then given a query $\boldsymbol{x}$ and dataset $\mathcal{D}$, the inference is done simply by sampling or estimating the mean of $p(y|\boldsymbol{x}, \mathcal{D})$.

Explicit Model. Contrary to the implicit model, the explicit model parameterizes the predictive distribution as $p_\gamma(y|\boldsymbol{x}, \boldsymbol{z}_\psi(\mathcal{D}))$, with a similar training procedure as above. Note that the predictive distribution only interacts with the dataset $\mathcal{D}$ through the latent $\boldsymbol{z}_\psi(\mathcal{D})$ while the implicit model allows unconstrained access to $\mathcal{D}$.

Implicit Proxy Model. To better understand the differences that play a role from architectural differences and parameterizations, we use exactly the same architecture as the explicit model to obtain a version of the implicit model. Such a model parameterizes the predictive distribution as $p_\gamma(y|\boldsymbol{x}, \boldsymbol{z}_\psi(\mathcal{D}, \boldsymbol{x}))$, with a similar training procedure as above. Note that the only difference with the explicit model here is that the conditional dependence of the query and the task latents is broken.

We refer the reader to Figure 16 for a plate diagram of the corresponding architectures.

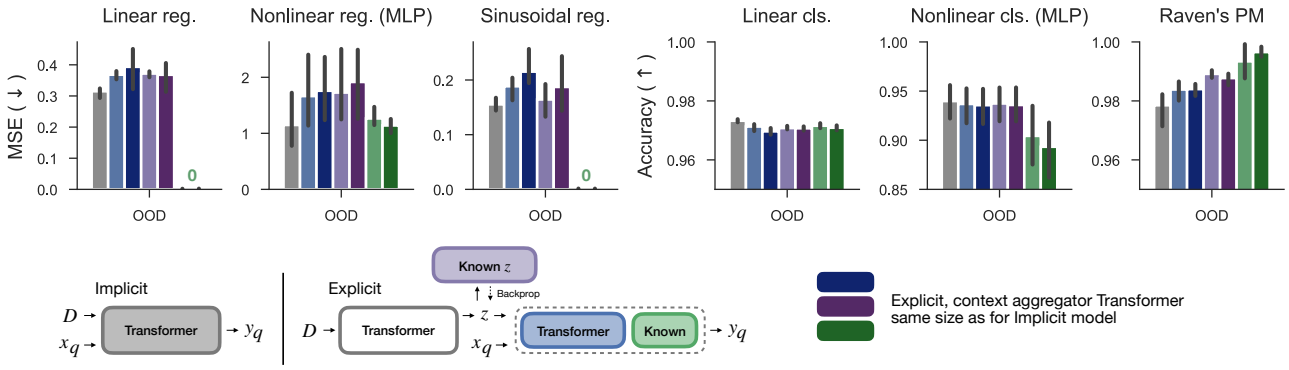

Figure 13. Similar to Figure 3 but with an additional explicit model where the context aggregator is identical in size and hyperparameters (M) as the implicit model. Performance on tasks where the true latents $\boldsymbol{z}$ and prediction function g are known. Implicit models are in gray, explicit models with Transformer prediction in blue, models trained with an auxiliary loss to predict true latents in purple and those using the true prediction function in green. Using the known prediction function leads to significantly better OOD performance. Lighter color indicates total model size identical to implicit as in main paper, darker = context aggregator identical to implicit.

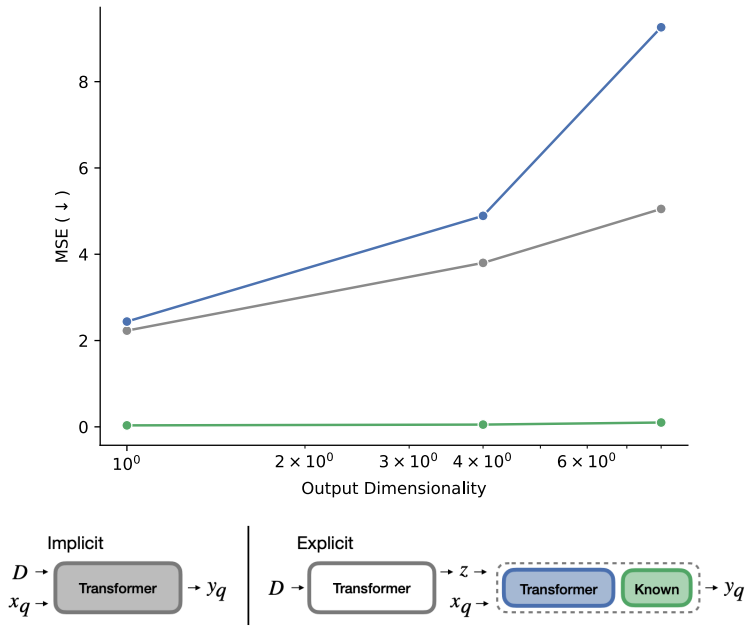

Figure 14. We study the impact of output dimensions on the performance of explicit and implicit models for linear regression with 8-dimensional inputs. The output dimensionality is considered to be 1, 4 and 8 dimensional, and our results indicate that implicit methods (gray) outperform explicit ones (blue), but the same explicit models with known prediction function (green) scales much better.

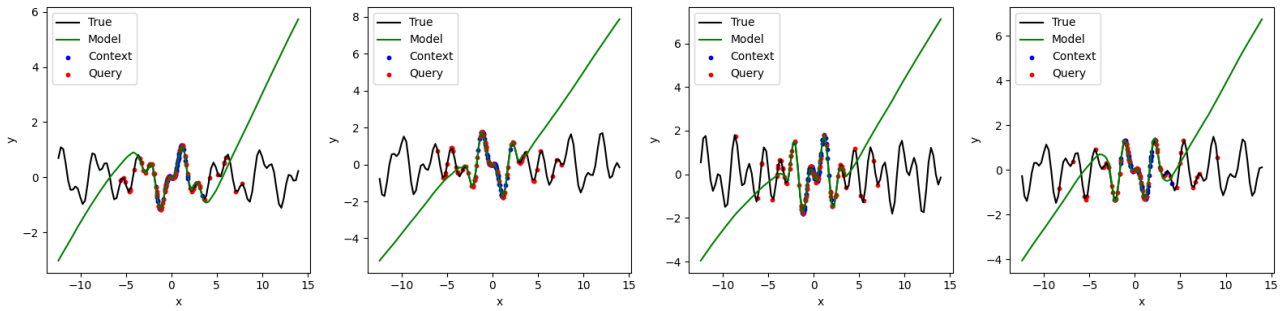

Figure 15. Illustration of explicit model with MLP prediction on the sinusoid task OOD. True function is shown in black, model output in green, context points in blue and query points in red. Our results indicate the failure of learned prediction function away from context.

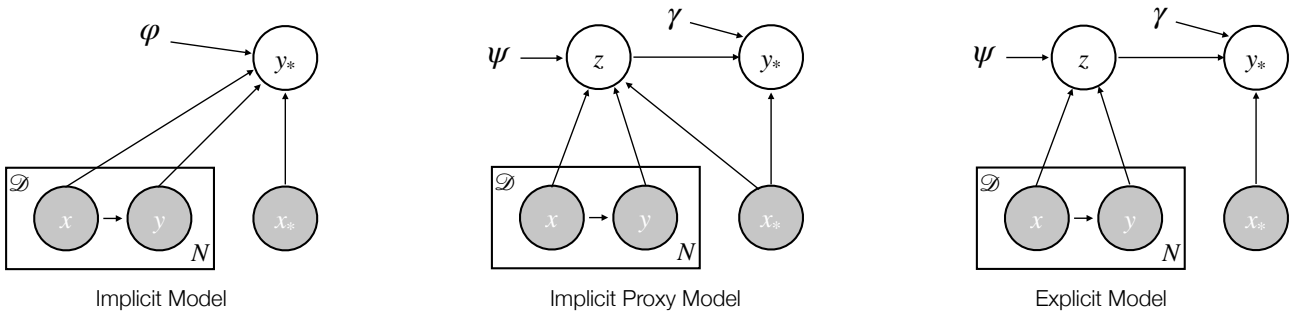

Figure 16. Plate diagram for the implicit model (left), implicit proxy model (middle) and the explicit model (right), where gray blocks refer to observed variables and white refers to unobserved variables. Trainable parameters are indicated without circles. In the explicit model case, $\boldsymbol{z}$ is currently modeled as a dirac measure defined via the trainable parameters ψ. One can see the implicit proxy model as very similar to the implicit model where output of the last attention layer corresponding to the query token is further processed to give prediction. Its similarity to the explicit model is also clear as it shares exactly the same parameterization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	L_2 Loss (↓)		
				IID	**OOD Query**	**OOD Latent**
Explicit-Known	4	-	2.1	**0.011**	**0.016**	0.782
	6	-	3.2	0.012	0.019	0.923
	8	-	4.2	0.013	0.023	1.391
Explicit-MLP	4	4	3.0	0.015	0.169	0.883
	4	6	3.6	0.017	0.174	0.728
	4	8	4.1	0.015	0.082	0.929
	6	4	4.1	**0.011**	0.201	1.047
	6	6	4.6	0.013	0.131	0.509
	6	8	5.1	0.012	0.077	**0.439**
	8	4	5.1	0.012	0.165	1.043
	8	6	5.7	0.017	0.171	1.020
	8	8	6.2	0.016	0.082	1.091
Explicit-Tsf	4	4	4.2	0.012	0.267	0.856
	4	6	5.3	0.014	0.335	1.014
	4	8	6.3	**0.011**	0.314	0.970
	6	4	5.3	0.025	0.389	1.128
	6	6	6.3	**0.011**	0.379	1.091
	6	8	7.4	0.012	0.289	0.981
	8	4	6.3	0.012	0.354	1.171
	8	6	7.4	0.013	0.311	1.134
	8	8	8.4	0.012	0.369	1.373
Implicit	4	-	2.1	0.013	0.258	0.817
	6	-	3.2	**0.011**	0.261	0.638
	8	-	4.2	**0.011**	0.287	0.661

Table 1. Analysis of different design choices for linear regression where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization where either the query is OOD or the underlying latent is.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	*Accuracy* (↑)		
				IID	**OOD Query**	**OOD Latent**
Explicit-Known	4	-	2.1	96.381	97.779	96.288
	6	-	3.2	96.331	97.746	96.324
	8	-	4.2	96.426	97.788	96.299
Explicit-MLP	4	4	3.0	96.297	97.646	96.150
	4	6	3.6	96.176	97.410	95.989
	4	8	4.1	96.338	97.772	95.964
	6	4	4.1	96.289	97.679	96.125
	6	6	4.6	95.885	97.154	95.803
	6	8	5.1	96.167	97.503	96.039
	8	4	5.1	96.194	97.496	95.761
	8	6	5.7	96.158	97.508	96.224
	8	8	6.2	95.915	97.006	95.758
Explicit-Tsf	4	4	4.2	96.386	97.760	96.239
	4	6	5.3	96.288	97.676	96.163
	4	8	6.3	96.453	97.874	96.349
	6	4	5.3	96.343	97.722	96.228
	6	6	6.3	96.415	97.853	96.365
	6	8	7.4	96.019	97.335	96.074
	8	4	6.3	96.288	97.733	96.306
	8	6	7.4	95.811	96.976	95.976
	8	8	8.4	96.304	97.729	96.068
Implicit	4	-	2.1	96.525	97.996	96.149
	6	-	3.2	**96.557**	98.036	**96.440**
	8	-	4.2	96.554	**98.147**	96.357

Table 2. Analysis of different design choices for linear classification where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization where either the query is OOD or the underlying latent is.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	L_2 Loss (↓) IID	L_2 Loss (↓) OOD Query
Explicit-Known	4	-	2.1	0.001	0.002
	6	-	3.2	**0.000**	**0.001**
	8	-	4.2	0.001	0.002
Explicit-MLP	4	4	3.0	0.001	0.295
	4	6	3.6	0.001	0.254
	4	8	4.1	0.001	0.208
	6	4	4.1	0.001	0.298
	6	6	4.6	0.001	0.307
	6	8	5.1	0.001	0.229
	8	4	5.1	0.001	0.399
	8	6	5.7	0.002	0.266
	8	8	6.2	0.001	0.222
Explicit-Tsf	4	4	4.2	0.001	0.179
	4	6	5.3	0.001	0.196
	4	8	6.3	0.001	0.186
	6	4	5.3	0.001	0.212
	6	6	6.3	0.001	0.230
	6	8	7.4	0.001	0.211
	8	4	6.3	0.002	0.201
	8	6	7.4	0.001	0.209
	8	8	8.4	0.001	0.203
Implicit	4	-	2.1	0.001	0.186
	6	-	3.2	**0.000**	0.143
	8	-	4.2	**0.000**	0.153

Table 3. Analysis of different design choices for sinusoid regression where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	L_2 Loss (↓) IID	L_2 Loss (↓) OOD Query
Explicit-MLP	4	4	3.0	0.039	0.205
	4	6	3.6	0.033	0.185
	4	8	4.1	0.037	0.181
	6	4	4.1	0.034	0.186
	6	6	4.6	0.030	0.183
	6	8	5.1	0.029	0.174
	8	4	5.1	0.030	0.186
	8	6	5.7	0.032	0.176
	8	8	6.2	0.031	0.175
Explicit-Tsf	4	4	4.2	0.028	0.181
	4	6	5.3	0.027	0.172
	4	8	6.3	0.027	0.172
	6	4	5.3	0.025	0.170
	6	6	6.3	0.024	0.169
	6	8	7.4	0.027	0.171
	8	4	6.3	0.025	0.171
	8	6	7.4	0.028	0.173
	8	8	8.4	0.030	0.177
Implicit	4	-	2.1	0.011	0.148
	6	-	3.2	**0.010**	0.148
	8	-	4.2	**0.010**	**0.147**

Table 4. Analysis of different design choices for GP regression where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	L_2 Loss (↓) IID	L_2 Loss (↓) OOD Query
Explicit-Known	4	-	2.1	0.084	1.323
	6	-	3.2	0.074	1.168
	8	-	4.2	0.073	**1.135**
Explicit-MLP	4	4	3.0	0.051	2.658
	4	6	3.6	0.051	2.708
	4	8	4.1	0.057	2.054
	6	4	4.1	0.051	2.814
	6	6	4.6	0.052	2.256
	6	8	5.1	0.058	2.395
	8	4	5.1	0.047	2.465
	8	6	5.7	0.053	2.061
	8	8	6.2	0.054	1.906
Explicit-Tsf	4	4	4.2	0.039	2.052
	4	6	5.3	0.044	2.185
	4	8	6.3	0.046	2.331
	6	4	5.3	0.042	2.154
	6	6	6.3	0.050	2.118
	6	8	7.4	0.052	2.351
	8	4	6.3	0.043	1.945
	8	6	7.4	0.044	2.148
	8	8	8.4	0.048	2.178
Implicit	4	-	2.1	**0.024**	1.443
	6	-	3.2	**0.024**	1.487
	8	-	4.2	**0.024**	1.509

Table 5. Analysis of different design choices for MLP regression where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	*Accuracy* (↑) IID	OOD
Explicit-Known	4	-	2.1	94.426	90.753
	6	-	3.2	94.633	91.086
	8	-	4.2	94.492	91.443
Explicit-MLP	4	4	3.0	94.888	92.775
	4	6	3.6	94.971	92.861
	4	8	4.1	94.465	92.342
	6	4	4.1	94.878	92.807
	6	6	4.6	94.933	93.024
	6	8	5.1	94.851	93.097
	8	4	5.1	94.818	93.036
	8	6	5.7	94.822	93.024
	8	8	6.2	94.878	92.985
Explicit-Tsf	4	4	4.2	95.074	92.990
	4	6	5.3	95.060	92.988
	4	8	6.3	95.157	93.190
	6	4	5.3	94.989	93.010
	6	6	6.3	95.060	93.275
	6	8	7.4	95.078	93.119
	8	4	6.3	95.010	93.133
	8	6	7.4	95.058	93.121
	8	8	8.4	94.969	92.954
Implicit	4	-	2.1	95.285	**93.394**
	6	-	3.2	**95.325**	93.336
	8	-	4.2	95.324	93.351

Table 6. Analysis of different design choices for MLP classification where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as out of distribution generalization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	*Accuracy* (↑) IID	OOD
Explicit-Known	4	-	2.1	99.864	99.844
	6	-	3.2	**99.876**	**99.875**
	8	-	4.2	99.545	99.539
Explicit-MLP	4	4	3.4	97.929	97.894
	4	6	4.0	97.551	97.532
	4	8	4.5	97.804	97.802
	6	4	4.5	97.394	97.383
	6	6	5.0	97.524	97.517
	6	8	5.5	98.527	98.514
	8	4	5.5	29.883	29.587
	8	6	6.1	98.240	98.216
	8	8	6.6	96.726	96.677
Explicit-Tsf	4	4	4.2	98.522	98.513
	4	6	5.3	98.467	98.455
	4	8	6.3	98.006	98.005
	6	4	5.3	98.494	98.459
	6	6	6.3	97.930	97.874
	6	8	7.4	98.566	98.591
	8	4	6.3	98.280	98.260
	8	6	7.4	97.750	97.744
	8	8	8.5	98.468	98.456
Implicit	4	-	2.1	98.359	98.334
	6	-	3.2	98.332	98.309
	8	-	4.2	97.164	97.176

Table 7. Analysis of different design choices for RAVEN's progressive matrices where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on in-distribution generalization as well as compositional out of distribution generalization.

Model	# Context Layers	# Prediction Layers	# Parameters (M)	R^2 (↑) Performance
Explicit-MLP	4	4	9.5	0.815
	4	6	10.0	**0.827**
	4	8	10.5	0.814
	6	4	10.5	0.814
	6	6	11.0	0.818
	6	8	11.6	0.819
	8	4	11.6	0.828
	8	6	12.1	0.820
	8	8	12.6	0.817
Explicit-Tsf	4	4	8.1	0.829
	4	6	9.1	0.824
	4	8	10.2	0.823
	6	4	9.1	0.808
	6	6	10.2	0.812
	6	8	11.3	0.812
	8	4	10.2	0.810
	8	6	11.3	0.818
	8	8	12.3	0.821
Implicit	4	-	4.7	0.819
	6	-	5.7	0.823
	8	-	6.8	0.817

Table 8. Analysis of different design choices for gene targeting experiments where implicit and explicit models are tested with different number of parameters or different parameter split between context aggregation and prediction. We conduct experiments on compositional out of distribution generalization.