TOWARDS PRINCIPLED DATA GENERATION FOR PRIOR-DATA FITTED NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

As natural sources of data are becoming exhausted, synthetic data is playing an increasingly important role for pre-training foundation models. Perhaps surprisingly, synthetic data has been shown to surpass natural data in downstream performance. For example, prior-data fitted networks (PFNs), which create an implicit prior through data generation, are trained entirely on synthetic data, but provide state-of-the-art results in tabular prediction. Despite this success, the design of synthetic data remains largely heuristic, requiring expensive trial-and-error rather than principled understanding. In this paper, we work towards a science of data generation, using PFNs to systematically understand the relationship between properties of the data and its performance in downstream tasks. We demonstrate that the synthetic data generation procedure shapes the training dynamics and inductive biases of the model, and thus the optimal pre-training data distribution may be distinct from the downstream task distribution. We identify two key properties of the prior, diversity and complexity, and show that increasing either tends to improve generalization, even if the prior is moved farther from the downstream task. Finally, we find that the complexity of the data generation process is predictive of the model's real-world performance.

1 INTRODUCTION

Synthetic data has become increasingly important to machine learning, enabling practitioners to augment finite natural datasets with infinite machine-generated data. These synthetic training datasets give us more precise control over model behavior (Abdin et al., 2024; Maini et al., 2024). Synthetic data has also been used to train transformers capable of solving particular tasks, such as numeric linear algebra (Charton, 2022) and in-context linear regression (Garg et al., 2023; Yang et al., 2024).

Prior-data fitted networks (PFNs) are deep learning models that learn to approximate Bayesian prediction by training on synthetically-generated data drawn from a prior (Müller et al., 2024a). The PFN paradigm has been particularly popularized in recent years by a variety of foundation models for tabular data and time series forecasting. In these domains, we can declare rich assumptions about the data, such as the existence of causal relationships between features in tables, or seasonality in time series, in the form of a synthetic data-generating distribution. This data-generating distribution essentially acts as the prior under which we perform approximate Bayesian inference.

Hollmann et al. (2022) present TabPFNv1, a foundation model for tabular data that was pre-trained only on synthetically-generated tables yet achieves remarkable accuracy on real-world tabular classification problems. Hollmann et al. (2025) expand upon TabPFNv1 by introducing a more complex pre-training distribution, adding support for tabular regression, and improving the model architecture, resulting in the widely-used tabular foundation model TabPFNv2. Their work demonstrated that a foundation model pre-trained on rich synthetically-generated data can achieve state-of-the-art performance on a wide variety of real-world datasets, outperforming traditional approaches such as XGBoost. Various later works build upon TabPFN by adjusting the synthetic pre-training distribution (Grinsztajn et al., 2025; Qu et al., 2025; Zhang et al., 2025).

Similarly, in time series forecasting, pre-trained foundation models have also garnered significant excitement in recent years. These models have extensively leveraged synthetically-generated data to obtain high-quality zero-shot forecasting performance on real-world downstream tasks (Dooley et al., 2023; Ansari et al., 2024; Das et al., 2024; Taga et al., 2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

While these works offer some guiding principles for synthetic data generation, the design of these synthetic priors remains a black art, depending primarily on heuristics and expensive trial-and-error. In this paper, we work towards a science of data generation, using PFNs to systematically understand the relationship between properties of the data and performance in downstream tasks.

We first make the observation that on OOD data, PFN predictive distributions can deviate substantially from the true Bayesian posterior they have been trained to approximate. Since PFNs are evaluated on real-world data which is likely to be OOD, the intuition that the best training data distribution is to match the downstream distribution may not apply exactly. For example, we show that PFNs often provide superior generalization to the true Bayesian posterior predictive which they have been trained to optimize!

Equipped with this understanding, we analyze the impact of two powerful levers to control synthetic data generation: *diversity* and *complexity*. Firstly, we show that increasing the diversity of the training data can often be a valuable way to improve the generalization of PFNs, even when this makes the true Bayesian posterior predictive worse! We then establish a trade-off between these benefits of increasing diversity and the risk of underfitting particular slices of the pre-training distribution. Secondly, we consider the effects of increasing the complexity of the data-generating distribution, showing a strong empirical connection between complexity and downstream performance.

2 Background

2.1 Bayesian prediction

Consider a setting in which a context $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ is used to make predictions for a test input x_{test} about test output y_{test}. Bayesian inference provides a principled way to model the predictive distribution $p(y_{\text{test}}|x_{\text{test}}, \mathcal{D}_{\text{train}})$.

From the Bayesian perspective, the dataset is assumed to be generated by an underlying data-generating mechanism parameterized by h, where each hypothesis h represents a specific relationship between inputs and outputs. For instance, h might represent a linear function with a particular slope and intercept. We can represent our prior beliefs over these hypotheses through $p(h)$, and predictions can be formed by marginalizing over the set of hypotheses to obtain the Bayesian posterior predictive distribution $p(y_{\text{test}} \mid x_{\text{test}}, \mathcal{D}_{\text{train}}) = \int p(y_{\text{test}} \mid x_{\text{test}}, h)\, p(h \mid \mathcal{D}_{\text{train}}) dh$. This distribution represents a weighted mixture of all hypotheses consistent with the observed data.

In some models, this integral can be computed in closed form. However, for many priors and likelihoods, exact Bayesian inference is intractable. In such cases, we instead resort to approximate methods, such as Markov chain Monte Carlo, which yields asymptotically exact samples from the posterior, or variational inference.

2.2 Prior-data fitted networks

Prior-Data Fitted Networks (PFNs) (Müller et al., 2024a) are designed to perform approximate Bayesian prediction through in-context learning. The key idea is to train a neural network to make predictions on many datasets drawn from the prior.

Specifically, we define a data generation process which consists of a distribution over hypotheses $p(h)$, and each hypothesis then defines a conditional distribution $p(y|x, h)$ from which we can sample datasets $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n+1}$. For each dataset, we split the data into a context set $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ and a hold-out query point $(x_{\text{test}}, y_{\text{test}})$. The PFN is trained to output the conditional distribution $p(y_{\text{test}}|x_{\text{test}}, \mathcal{D}_{\text{train}})$ by minimizing the expected negative log-likelihood across datasets drawn from the prior $\mathcal{L}_{\text{NLL}}(\theta) = \mathbb{E}_{h\sim p(h)} \mathbb{E}_{\mathcal{D}\sim p(\mathcal{D}|h)}[-\log f_\theta(y_{\text{test}}|x_{\text{test}}, \mathcal{D}_{\text{train}})]$.

Müller et al. (2024a) show that optimizing this objective with respect to θ is equivalent to minimizing the KL divergence between the true Bayesian posterior predictive distribution $p(y_{\text{test}}|x_{\text{test}}, \mathcal{D}_{\text{train}})$ and the model predictions $f_\theta(y_{\text{test}}|x_{\text{test}}, \mathcal{D}_{\text{train}})$.

In practice, PFNs are typically implemented using Transformer-based architectures (Müller et al., 2024a) and often contain explicit inductive biases optimized for in-context learning. Most models are designed to be permutation invariant with respect to the ordering of the training examples through

their self-attention mechanisms. PFNs may also have specialized attention mechanisms which encode the asymmetric relationship between context points and test points. Furthermore, the output layer often consists of a specialized prediction head which produces flexible distributions rather than point estimates. As a result, although PFNs are trained with the goal of approximating Bayesian inference under a specific prior, the quality of these predictions is often shaped by the interaction of the prior with the model's inductive biases.

3 Priors Specify More Than the Posterior

In classical Bayesian inference, the prior serves a clear role: it specifies our beliefs before observing any data, and it also induces a posterior distribution which dictates predictive performance. From this perspective, when designing synthetic data generators for training foundation models, it may seem reasonable to focus exclusively on the posteriors that they induce, and therefore aim to match the prior as closely as possible to the downstream task distribution.

Under idealized assumptions, focusing only on the alignment between the prior and the downstream task is well-justified: when a model performs exact probabilistic inference over its prior, then matching the prior to the downstream task distribution is Bayes-optimal. Any other prior would strictly increase the expected loss: a prior that is more diverse would unnecessarily allocate probability mass to irrelevant parts of the input space, while a prior that is too narrow would fail to cover important variations of the task. However, this matching strategy relies on the assumption that PFNs perfectly perform Bayesian inference over their training priors. If the prior design also influences the model's inductive biases and learning dynamics, then we need to consider more than just the induced posterior when generating data.

In this section, we demonstrate that PFNs often behave differently than the priors that they were trained on, even when trained to convergence under standard metrics. We evaluate the PFN on synthetic settings where we can analytically compute the Bayesian posterior predictive distribution, and we find that PFNs often diverge from the true posterior, particularly in regions which have low probability under the prior. Moreover, we identify settings where PFNs actually generalize more gracefully to OOD examples compared to their Bayesian posteriors. These results demonstrate that the prior can encode useful inductive biases beyond simply specifying the target distribution, motivating a broader perspective on prior design for PFNs.

3.1 Lines and Sines

We start our analysis with an illustrative scenario where we can easily visualize the difference in behavior between PFNs and the true Bayesian posterior predictive. Following the setup of Müller et al. (2024b), we train a PFN with the prior generation procedure of two distinct types of functions: lines or sine curves, and we provide details about the training regime in Appendix B. Specifically, we first sample training points x uniformly from $[-1, 1]$, and we then generate functions

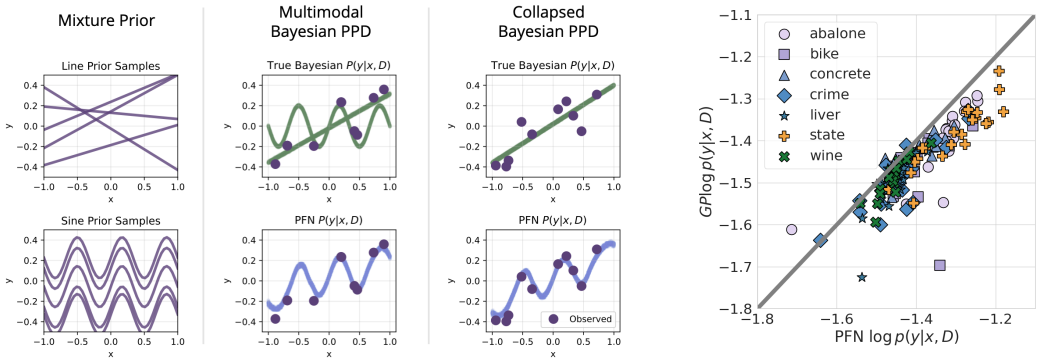
$$f_{\text{line}}(x) = m\,x + b, \qquad m, b \in \{-1, -0.99, \cdots, 1\}$$

$$f_{\text{sin}}(x) = 0.2\sin(3\pi x) + \phi, \qquad \phi \sim U(-1, 1)$$

where f_{final} is then set to either f_{line} or f_{sin} with equal probability, and the target $y = f_{\text{final}}(x) + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$. We visualize samples from this prior in the left panel of Figure 1, and our experiments are conducted in the low noise setting of $\sigma^2 = 0.01$.

We approximate the Bayesian posterior predictive distribution by discretizing the uniform prior space into a finite grid and marginalizing using a weighted average. We find that the PFN closely mirrors the behavior of Bayesian inference for the in-distribution setting, where the data is drawn from either the line or the sine, as seen in Figure A.1. However, the behavior starts to diverge when we look at settings which have very low probability under the prior.

In Figure 1, we visualize the differences between the analytically computed Bayesian posterior predictive (top), and the PFN prediction (bottom) for a few representative datasets. In the first setting, the observations $\mathcal{D}$ lead to comparable likelihoods under both f_{line} and f_{sin}, where $p(\mathcal{D}|\text{line}) \approx p(\mathcal{D}|\text{sin})$. The Bayesian posterior predictive therefore assigns probability mass to both hypotheses, resulting in a multimodal distribution. In contrast, PFNs do not display this same multimodality,



(a) Lines and Sines

(b) Performance on UCI

Figure 1: **PFNs often do not perform Bayesian inference in OOD settings. (Left):** On low-likelihood settings, the true Bayesian posterior predictive distribution may display multimodal behavior or mode collapse. In contrast, the PFN predictions display very different behavior and instead predict an increasing sine curve. **(Right): PFNs can beat Bayesian posterior inference on real-world data.** We plot the log posterior predictive values (higher is better) for the PFN (x-axis) vs the true GP posterior predictive values (y-axis) across 7 distinct UCI datasets. Points below the $y = x$ line indicate settings where the PFN surpassed the GP.

even when using a discretized Riemann output head (Müller et al., 2024a), which can, in principle, represent multimodal predictions. Instead, the PFN interpolates between the explanations, producing a single smooth prediction resembling a sloped sine wave. In the second setting, the Bayesian posterior assigns negligible probability to the sine curves and instead concentrates its probability mass on the linear hypothesis. The PFN again behaves very differently from the Bayesian posterior: rather than collapsing on one particular mode, the model again interpolates between observations.

We note that this behavior should not be interpreted as Bayesian model averaging, where different weights are placed on hypotheses from the prior. Instead, we see PFNs interpolate between different hypotheses and produce predictions that do not correspond to valid posterior samples or mixtures under the prior. In summary, **PFNs do not perform exact Bayesian inference and instead display inductive biases which are shaped by the training distribution**. Müller et al. (2024b) also observe that PFNs can deviate from the true Bayesian posterior predictive, but they attribute this behavior to approximation error. We instead find that this behavior is systematic and can be beneficial for generalization, as we will show in the next sections.

3.2 Generalization Under Polynomial Degree Shift

We now study whether the PFN's ability to make predictions which are different than Bayesian posterior inference could be beneficial for generalization under distribution shift. We consider a setting where we train a PFN on a prior consisting of polynomials of degree $d \leq 3$, and we evaluate the model on unseen polynomials of larger degrees. We compare the PFN with the exact posterior predictive distribution under the training mixture, as well as an oracle which has access to the true degree of the polynomial. We provide further details about the training procedure in Appendix B.

We see in Figure 2 that PFN and the true Bayesian posterior predictive perform similarly on polynomials of degree 3, which are in-distribution. However, when evaluated on unseen polynomials of degree $d > 3$, the PFN consistently outperforms the true Bayesian posterior predictive, achieving better MSE and log-likelihoods (Figure A.2).

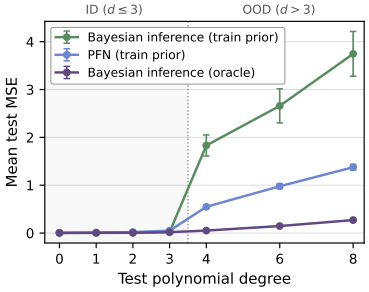


Figure 2: **PFNs can generalize better than Bayesian inference on OOD polynomials**. We plot the mean and standard error over 512 tasks.

We also note that the oracle posterior remains substantially better than the PFN. Given a correctly specified prior over the test degree and the coefficients, the oracle is Bayes-optimal for these tasks. We therefore do not claim that PFNs can outperform correctly specified Bayesian inference. Instead, **we focus on the settings where the prior is *misspecified*, and we find instances where PFN can generalize better than exact inference**. This illustrates how the inductive biases of PFNs can help under distribution shifts. We provide further experimental details and visualizations in Section A.2.

3.3 Generalization on UCI Benchmarks

Even though PFNs are trained on synthetic data, we often care about their performance on real-world datasets. We now study the performance of PFNs in real-world tasks from the UCI repository (Kelly et al., 1994). For this experiment, we train a PFN model on synthetic data generated from Gaussian processes (Rasmussen & Williams, 2006) with an RBF kernel. For each task, we sample the length scale ℓ and the noise scale σ from the discrete set $\{0.01, 0.03, 0.1, 0.3, 1.0, 3.0\}$. Because this set corresponds to a finite mixture of Gaussian processes, we can compute the exact Bayesian posterior predictive distribution by marginalizing over the discrete hyperparameters. This allows us to perform a controlled comparison between the PFN and the true Bayesian posterior predictive distribution.

We evaluate the PFN and the analytical posterior predictive on several UCI datasets. For each UCI dataset, we randomly select a numerical column as the input x and designate the target variable as y. We then sample $N = 25$ training points and $J = 100$ test points. We measure the mean log posterior predictive probability $\frac{1}{BJ}\sum_{bj} \log p(y^\star_{b,j}|x^\star_{b,j}, \mathbf{x}_b, \mathbf{y}_b)$ over $B = 10$ batches for each input-target pairing, and we select 25 random input-target pairs per UCI dataset.

In Figure 1 (right), we see that the performance of our trained PFN surpasses the posterior predictive values from the actual Gaussian process in a majority of settings. These findings mirror the conclusions from the previous section: although the prior strongly influences the inductive biases of the PFN, the model is not restricted to exact Bayesian inference during test time. Instead, PFNs appear to learn strategies which enable them to adapt to unlikely inputs, leading to improved performance in OOD settings. **We find that PFNs can generalize better to real-world data compared to their true Bayesian posterior predictive distribution.**

This behavior may seem surprising, since PFNs are trained to approximate the Bayesian posterior predictive under their prior. However, this interpretation assumes the network has enough capacity and compute to represent that posterior exactly. In practice, finite capacity and design choices such as the architecture introduce their own inductive biases. Furthermore, our earlier experiments show that a mismatch between training and test tasks can allow the PFN to outperform the posterior under its prior, and we find that the UCI datasets differ from the synthetic GP datasets. We explore both of these factors in Section A.3.

4 The Role of Diversity

Our findings demonstrate that PFNs do not simply mirror the exact Bayesian posterior predictive distribution induced by their prior; instead, they exhibit behaviors which systematically deviate from exact posterior inference. This suggests that PFNs have learned inductive biases which are shaped by their training procedure and the corresponding design of the synthetic data generation procedure.

One valuable lever available to us when designing synthetic pre-training datasets is to adjust the *diversity* of the data. Conceptually, diversity represents the breadth of the training distribution and provides a notion of the similarity between sampled tasks. In this section, we explore how diversity can be used to improve the performance of PFNs, especially in OOD settings.

4.1 When Diversity Trumps a Better-Specified Prior

We consider three simple synthetic scenarios where we can separately control the diversity of the training distribution and the distance between the prior and the downstream task: Gaussian mean estimation, linear regression, and 2D classification (detailed experimental setup described in Section A.4). These settings are chosen to be simple enough that we can analytically compute the Bayesian posterior predictive distribution, while still being rich enough to demonstrate the effects of diversity on downstream performance.

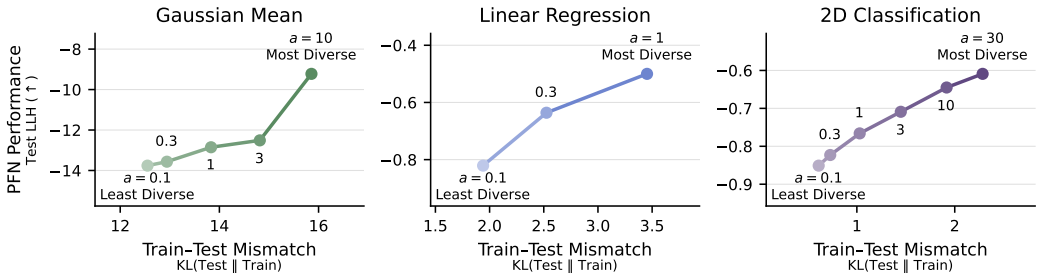


Figure 3: **Diverse priors can outperform less diverse ones, even if they are further from the downstream task**. We vary the diversity a of the training prior for three different settings: Gaussian mean estimation (left), linear regression (middle), and 2D classification (right). In each setting, we evaluate the model on a downstream task which is outside the support of the training distribution. We find that increasing the diversity of the training prior improves downstream performance, even though it moves the prior further from the downstream task.

We first consider the Gaussian density estimation problem where the observations x_i are generated from $\mu \sim \text{Uniform}(0, a)$ and $x_i \mid \mu \sim \mathcal{N}(\mu, 1)$. We then consider the downstream task $\mu = -1$. Classical Bayesian reasoning suggests that larger a values should lead to worse performance, since the prior will concentrate more probability mass further from the test values. Indeed, we see in Figure 3 (left) that the KL divergence between the prior and the downstream task increases with a, indicating that the prior is moving further from the downstream task. However, we see that the PFN trained on this more misaligned prior actually achieves better test log-likelihoods! The prior that is more diverse leads to better generalization on the downstream task. This differs from the behavior of the true Bayesian posterior predictive, which performs worse as a increases (Figure A.5). We repeat this experiment for two additional settings (details in Section A.4) and find similar results.

These experiments highlight a key difference between theoretical Bayesian inference and PFNs: the diversity of the prior can outweigh better alignment with the downstream task. In fact, **a more diverse prior for the training data can improve the generalization of PFNs, even when the true Bayesian posterior predictive becomes worse!**

We hypothesize that this improved downstream performance is achieved since the more diverse prior encourages the model to learn a local algorithm to predict the mean of the input data points, rather than simulate the likelihood for different prior means and take a Bayesian model average. As such, the model is able to generalize successfully to OOD inputs. This result is similar to the findings in Raventós et al. (2023), who find that transformers trained on larger numbers of regression tasks shift from the Bayes-optimal predictor over training towards ridge regression. However, in their setting, increasing diversity also decreases the distance between the training and test distributions, so it is unclear whether the improved generalization is due to the increased diversity or the decreased distance. In contrast, our experiments decouple these two effects, and we find that increasing diversity improves generalization even when it moves the prior further from the downstream task.

Additionally, our results provide an additional perspective on the performance gains reported by TabPFNv2.5 (Grinsztajn et al., 2025) over TabPFNv2 (Hollmann et al., 2025). In this series of work, increasing the diversity of the prior, among other improvements, led to boosts in performance on real-world tasks. While this improved performance may simply result from the more diverse prior better matching the true distribution of downstream tasks, it may also indicate that the model is being encouraged to learn more generally useful representations, independent from the closeness of its prior to the downstream distribution.

4.2 Measuring Diversity

Although diversity can be explicitly tuned in simple settings such as the Gaussian example above, synthetic priors are often defined by complex procedures where it may be harder to characterize. Therefore, it is necessary to develop a general measurement of diversity that can be applied across different priors and captures meaningful differences in their behaviors. We propose a simple proxy: the standard deviation of the loss over validation batches at the end of training. Intuitively, a more diverse prior produces validation batches which are more different from one another, leading to a higher variance in the loss. Applied to the Gaussian setting, our diversity proxy closely tracks the effect of a on downstream performance, achieving a very high correlation of 0.96 (Figure 4).

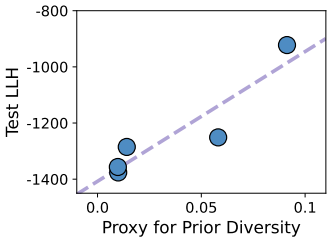


Figure 4: **Our proxy for diversity has a strong correlation with downstream performance** ($r = 0.96$). We compare each model in the Gaussian setting.

4.3 The Cost of Increased Diversity

Although there are clear benefits to increasing the diversity of the synthetic data, it may also lead to practical drawbacks. From a Bayesian perspective, the probability mass over the true downstream distribution becomes unnecessarily diluted, which should worsen in-distribution performance. However, PFNs do not perform exact Bayesian inference (Section 3), and therefore it is unclear whether this trade-off arises in practice.

We study this question on two widely used tabular foundation models: TabPFNv1 (Hollmann et al., 2022) and TabPFNv2 (Hollmann et al., 2025). Although the full details of the training prior are not publicly available, Hollmann et al. (2025) state that TabPFNv2 is "built on a preliminary version of [TabPFNv1]" and additionally supports regression tasks, categorical data, and more, suggesting a more diverse prior. TabPFNv2's other changes (like architecture and compute) improve real-world performance so we would expect them to help rather than hurt in our setting. Therefore, this comparison is a conservative estimate of the cost of diversity.

We evaluate both models on tasks drawn from a subset of the TabPFNv1 prior. Specifically, we generate causal graphs of a fixed depth by sampling an MLP with tanh activations, adding Gaussian noise after each layer, and randomly dropping edges. From each causal graph, we randomly select five nodes as input features x and one node as the target y. Because TabPFNv2 has probability mass across a wider range of datasets, it is therefore expected to achieve slightly lower performance in this setting. As expected, TabPFNv2 performs slightly worse than TabPFNv1 (Figure 5), illustrating a potential cost of increased diversity. However, this effect is small, and since real data is unlikely to match any synthetic prior exactly, the robustness gained from diversity is likely to outweigh the costs in practice.

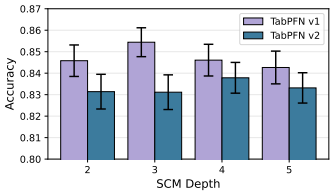
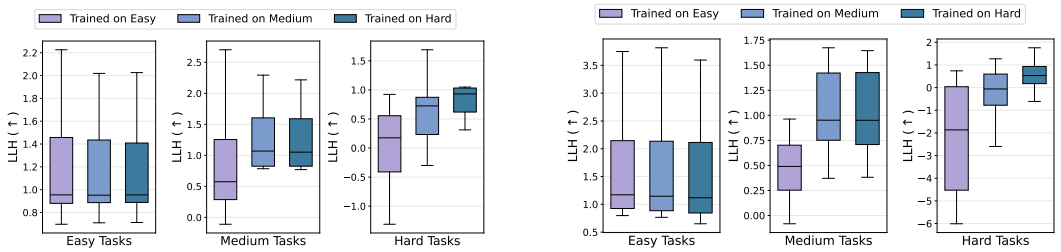


Figure 5: **More diverse priors may hurt performance on data drawn from a narrower prior**: TabPFNv1 slightly outperforms TabPFNv2 on data from the TabPFNv1 training distribution.

5 The Role of Complexity

Apart from prior diversity, we also explore the complementary dimension of the *complexity* of the prior distribution and study its impact on model generalization. Classical intuitions from Occam's razor and simplicity bias suggest that models should be trained on relatively simple problems that encourage the model to learn compact solutions. At the same time, training on more challenging tasks may enable the model to learn richer representations and better generalize to downstream tasks.

In this section, we systematically evaluate these hypotheses by benchmarking PFNs trained on priors with varying complexities. Specifically, we use controlled variations of the structural causal model



(a) Node Retention Probability

(b) Edge Retention Probability

Figure 6: **More complex priors perform well across synthetic settings of varying difficulties.** We train models with the SCM prior and vary the node and edge retention probabilities between 0.5 (easy), 0.7 (medium) and 0.9 (hard). We evaluate on the same set of downstream tasks and find that less complex priors fail to generalize to complex settings.

(SCM) data generation procedure from Hollmann et al. (2022), which has been widely used in follow-up work and has demonstrated strong performance across diverse downstream tasks.

The SCM prior is designed to generate datasets which resemble structured and causal processes. Conceptually, we want to sample a directed causal graph whose nodes correspond to random variables and edges correspond to dependencies between them. We first construct a fully-connected multilayer perceptron (MLP), where each node corresponds to a scalar random variable. We then induce sparsity by randomly removing nodes or edges according to specified probabilities. Each remaining node computes its value as a function of its parents, followed by an addition of independent noise. After this process, a random subset of the nodes are designated as observed input features x, one node is designated as the target y, and the remaining nodes are treated as unobserved latent variables.

Within this framework, we vary task complexity along three axes (easy/medium/hard): (i) *edge retention* (0.5/0.7/0.9), where higher values yield denser graphs; (ii) *node retention* (0.5/0.7/0.9), where higher values introduce more latent variables; and (iii) *noise level* (0.05/0.1/0.15), where higher values reduce signal. Along each complexity dimension, we train three models with data generated at the fixed complexity levels, and we specify the full training details in Appendix B. This design allows us to understand how the complexity of the training distribution impacts the model's ability to generalize across different downstream tasks.

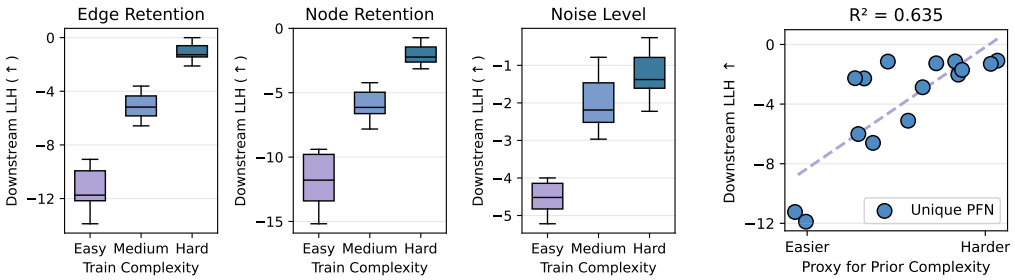
5.1 Controlled Synthetic Task

Given a downstream task of a fixed complexity, what complexity should you use to train your model? To answer this question, we start in the synthetic setting where we can control the complexity of the downstream task by sampling from fixed configurations of the SCM prior. In Figure 6 and Figure A.6, we find that training with more complex data is generally performant across a diverse range of downstream complexities. Although the models trained on hard data perform slightly worse than those trained on easy data when tested on the easy task, they still generalize reasonably well. In contrast, models trained on easy data entirely fail on hard tasks.

5.2 Generalization to Real-World Benchmarks

We evaluate this suite of models on real-world regression tasks. We evaluate on two different tabular datasets: our first benchmark consists of 60 regression datasets from UCI (Kelly et al., 1994). For each dataset, we randomly select 5 columns as features to predict the target, and we use a context-length of 50 and evaluate on the remaining points. We also evaluate on TabArena (Erickson et al., 2025), the gold standard benchmark for tabular foundation models and the benchmark used to evaluate the state-of-the-art models. We evaluate on all regression datasets from TabArena after dropping categorical features and excluding the two datasets with over 80 features.

Our empirical results in Figure 7 and Figure A.8 demonstrate that the models trained on the most complex data also have the strongest downstream performance. This finding actually contradicts the conventional wisdom presented in Hollmann et al. (2022), where the prior is explicitly designed



(a) Results when varying individual dimensions of complexity

(b) Summary across all runs

Figure 7: **Complex priors generalize better to downstream tasks**. We vary the complexity of the prior among three dimensions, edge retention probability, node retention probability, and input noise level. **(a):** We find that PFNs trained on more complex settings perform better and obtain higher likelihoods on 60 real-world regression tasks sampled from UCI datasets. **(b):** To summarize the effect of prior complexity across many different experiment settings, we use a scalar proxy that captures the intrinsic difficulty of the data generation process. We see in the right plot that this complexity proxy is strongly predictive of the downstream performance, achieving a high R^2.

with a simplicity bias in mind. By varying the complexity one axis at a time within a fixed family rather than introducing new generating procedures (den Breejen et al., 2025), we can attribute the improvements to specific properties of the prior. We conclude that **the complexity of the training dataset can be a strong predictor of a PFN's downstream performance on real-world datasets.**

5.3 Measuring Complexity

To make these findings actionable for prior design, we now introduce practical and computable proxies which correlate with downstream performance. Our first proxy is quantile-based: we measure the 90% quantile of the trained model's log-likelihood on prior-generated data. This metric summarizes the model's performance on the "easier" examples in the dataset; a lower final log-likelihood indicates that even the simplest inputs in the training dataset provide a non-trivial challenge to the model. We find that this measure of complexity is highly predictive of downstream success, achieving an R^2 value of 0.635 with UCI (Figure 7, right) and 0.48 on TabArena (Figure A.8).

Our second proxy is complementary to the first and builds on *epiplexity* (Finzi et al., 2026), the measure of the structural information in dataset X that can be extracted under computational constraints. Formally, given a compute budget T and a universal prefix Turing machine, we decompose the total information by finding the optimal program $P^* = \arg\min_{P \in \mathcal{P}_T} |P| + \mathbb{E}_X[\log 1/P(X)]$, where $|P|$ represents the model's description length and $\mathbb{E}[\log 1/P(X)]$ is the time-bounded entropy. Following the prequential coding approach of Finzi et al. (2026), we estimate $|P^*|$ as the area under the training loss curve above the final loss; this yields a correlation of 0.44 with downstream log-likelihood on UCI, as seen in Figure A.8, demonstrating that prior complexity predicts generalization.

6 Conclusion

Our work demonstrates that effective prior design should consider not only the downstream distribution but also how the generated data interacts with inductive biases of models. We show that PFNs are strongly shaped by their inductive biases encoded in both the training distribution as well as model architecture, and that these biases can lead them to diverge from Bayesian inference.

We further identify diversity and complexity as two key axes of prior design, and introduce practical proxies to measure them. Across synthetic settings and real-world benchmarks, increasing either property tends to improve downstream generalization, and our proxies consistently show this correlation. We hope our findings and proxies enable practitioners to approach prior design with explicit consideration of how their choices influence model generalization.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 Technical Report. *arXiv 2412.08905*, 2024.

Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, Jasper Zschiegner, Danielle C. Maddix, Hao Wang, Michael W. Mahoney, Kari Torkkola, Andrew Gordon Wilson, Michael Bohlke-Schneider, and Yuyang Wang. Chronos: Learning the Language of Time Series. *arXiv:2403.07815*, 2024.

François Charton. Linear algebra with transformers. *Transactions in Machine Learning Research (TMLR)*, 2022.

Abhimanyu Das, Weihao Kong, Rajat Sen, and Yichen Zhou. A decoder-only foundation model for time-series forecasting. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=jn2iTJas6h.

Felix den Breejen, Sangmin Bae, Stephen Cha, and Se-Young Yun. Fine-tuned in-context learning transformers are excellent tabular data classifiers, 2025. URL https://arxiv.org/abs/2405.13396.

Samuel Dooley, Gurnoor Singh Khurana, Chirag Mohapatra, Siddartha Naidu, and Colin White. ForecastPFN: Synthetically-Trained Zero-Shot Forecasting. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Mutalik Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data, 2025. URL https://arxiv.org/abs/2506.16791.

Marc Finzi, Shikai Qiu, Yiding Jiang, Pavel Izmailov, J Zico Kolter, and Andrew Gordon Wilson. From entropy to epiplexity: Rethinking information for computationally bounded intelligence. *arXiv preprint arXiv:2601.03220*, 2026.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes. *arXiv:2208.01066*, 2023.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, Mihir Manium, Rosen Yu, Felix Jablonski, Shi Bin Hoo, Anurag Garg, Jake Robertson, Magnus Bühler, Vladyslav Moroshan, Lennart Purucker, Clara Cornu, Lilly Charlotte Wehrhahn, Alessandro Bonetto, Bernhard Schölkopf, Sauraj Gambhir, Noah Hollmann, and Frank Hutter. Tabpfn-2.5: Advancing the state of the art in tabular foundation models, 2025. URL https://arxiv.org/abs/2511.08667.

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

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. UC Irvine Machine Learning Repository, 1994. URL https://archive.ics.uci.edu.

Pratyush Maini, Skyler Seto, He Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the Web: A Recipe for Compute and Data-Efficient Language Modeling. *arXiv 2401.16380*, 2024.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference, 2024a. URL https://arxiv.org/abs/2112.10510.

Samuel Müller, Noah Hollmann, and Frank Hutter. Bayes' power for explaining in-context learning generalizations, 2024b. URL https://arxiv.org/abs/2410.01565.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=0VvD1PmNzM.

C. Rasmussen and C. Williams. *Gaussian processes for machine learning*. MIT Press., 2006.

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-Bayesian in-context learning for regression. *Advances in neural information processing systems*, 36:14228–14246, 2023.

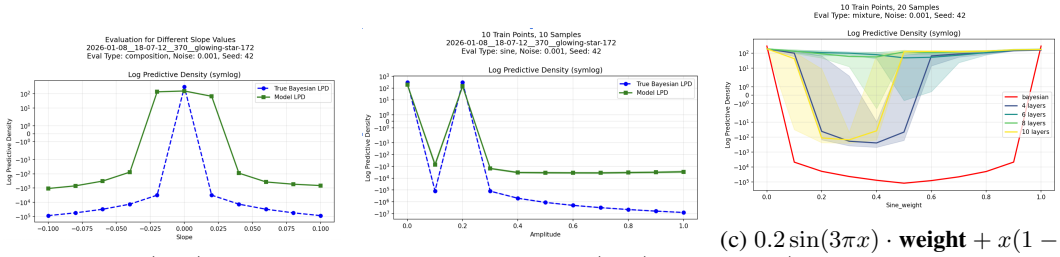
Ege Onur Taga, M. Emrullah Ildiz, and Samet Oymak. TimePFN: Effective Multivariate Time Series Forecasting with Synthetic Data. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2025.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped Transformers are Better at Learning Learning Algorithms. *International Conference on Learning Representations (ICLR)*, 2024.

Xiyuan Zhang, Danielle C. Maddix, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael W. Mahoney, Cuixiong Hu, Huzefa Rangwala, George Karypis, and Bernie Wang. Mitra: Mixed synthetic priors for enhancing tabular foundation models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=t8YRsWY6HM.

A Additional Empirical Results

A.1 Lines and Sines



(a) $y = 0.2\sin(3\pi x) + \textbf{slope} \cdot x$
ID regime: slope = 0

(b) $y = \textbf{amplitude} \cdot \sin(3\pi x)$
ID regime: amplitude = 0.0 or 0.2

(c) $0.2\sin(3\pi x) \cdot \textbf{weight} + x(1 - \textbf{weight})$
ID regime: weight = 0 or 1

Figure A.1: **PFNs can generalize better than true Bayesian posterior inference on OOD settings.** The ID settings are either $y = 0.2\sin(3\pi x) + \phi$ or $y = mx + b$ for $\phi, m, b \sim U[-1, 1]$.

We create a family of OOD settings:

- Sine waves with various slopes $y = 0.2\sin(3\pi x) + \text{slope} \cdot x$
- Sine waves of various amplitudes $y = \text{amplitude} \cdot \sin(3\pi x)$
- Interpolation between sines and lines $y = 0.2\sin(3\pi x) \cdot \text{sine_weight} + x \cdot (1 - \text{sine_weight})$

For each of these evaluation setups, there is an in-distribution setting which corresponds exactly to the parameters used during training, and there are OOD settings which smoothly extrapolate beyond these values. For instance, for sloped sine waves $y = 0.2\sin(3\pi x) + \text{slope} \cdot x$, when the slope = 0, this is equivalent to the ID setting that the model was trained on which only considered flat sine waves. We visualize the results in Figure A.1 (left), and we see that PFNs and Bayesian models have very similar performances on the ID setting with 0 slope. However, as soon as we move away from the ID setting, we see that the PFN generalizes better and is able to retain higher predictive densities compared to the prior that it was trained on. We observe similar behavior across the other OOD settings. Our results in these experiments consistently point to PFNs behaving in ways which do not match exact Bayesian inference for the prior that they were trained on.

A.2 Polynomial Shift

For each training task, we sample a degree uniformly from 0,1,2,3, draw inputs uniformly from [-1,1], and draw independent Gaussian coefficients. We draw independent Gaussian coefficients for every term in an orthonormalized Legendre basis and divide the coefficients by $\sqrt{d+1}$ to keep the variance of the function outputs consistent across degrees. We add Gaussian observation noise with standard deviation 0.05. Each task has eight context observations and 16 training query observations. We train one PFN for 15,000 steps with batch size 64 using Gaussian negative log likelihood. At evaluation, we use 512 shared tasks per degree, each with eight context and 64 test observations, for degrees 3, 4, 6, and 8, using the same input, coefficient, scaling, and noise rules. The training-prior Bayesian predictor exactly marginalizes over the four possible training degrees; the degree-matched predictor uses the test generator's degree and coefficient prior. We report mean squared error and mean predictive log likelihood on the same test tasks for all predictors.

A.3 UCI Generalization

To measure a proxy for local nonstationarity, for each point, we measure the mean squared label difference to its 5 nearest neighbors in standardized input space, then compute the coefficient of variation across points. UCI has median 1.171 versus 0.866 for GP samples, with 54.2% of UCI tasks above the GP 95th percentile. For the residual tail behavior, we fit ordinary least squares and compute the excess kurtosis of the residuals. UCI has median 0.673 versus -0.082 for GP samples, with 40.7% of UCI tasks above the GP 95th percentile. These checks show that many UCI tasks exhibit uneven

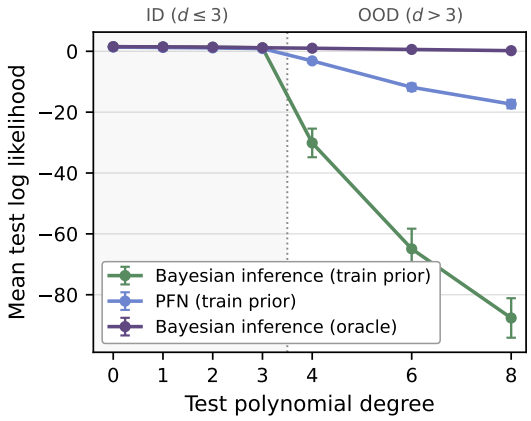


Figure A.2: **Predictive log likelihood under polynomial degree shift.** The setup and model colors match Figure 2. Points show mean test log likelihood over 512 shared tasks per degree, with 95% intervals for each model mean. Higher is better. Bayesian Prediction is the exact posterior under the training mixture; Oracle Prediction uses the test generator.

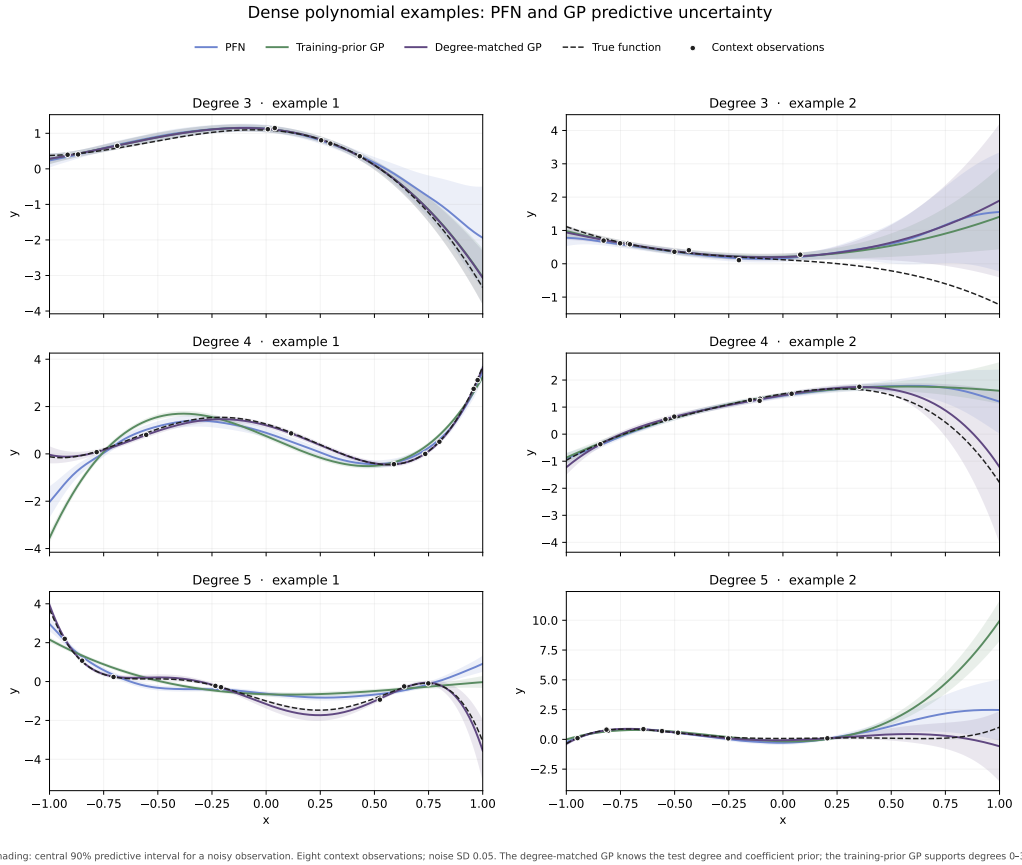


Figure A.3: **Six illustrative polynomial task draws.** Two draws each at test degrees 3, 4, and 5 show predictive means and central 90% intervals for a noisy observation. Dots are the eight context observations; the dashed line is the noiseless function. The model colors match Figure 2. Degree 5 is shown for illustration and is not part of the 512-task summary above.

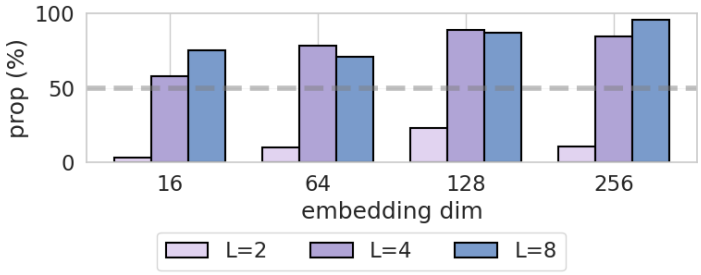


Figure A.4: **Design choices such as model architecture can significantly impact PFN generalization**. We show the proportion of sampled UCI datasets where the PFN model achieves a better posterior density compared to its theoretical Bayesian posterior predictive density as we vary the model capacity through the size of the embeddings and the number of layers.

local structure and heavier-tailed residuals than typical samples from the GP prior, supporting our claim that real-world tabular tasks can be distributionally mismatched to PFN training priors.

We further analyze how architectural choices such as the number of embedding dimensions and the depth of the network influence our model's behavior. These experiments followed the same mixture of GP prior and training steps as described in the previous section. We first note that all models achieve similar training loss, indicating that their approximations for the in-distribution posterior predictive are similar. However, we find that these architectural differences play a significant part in the model's performance on OOD: In Figure A.4, we plot the proportion of UCI experiments where PFNs achieve a better likelihood compared to their theoretical posterior predictive. Moving from the largest model capacity, with an embedding dimension of $D = 256$ and $L = 8$ layers, we see that decreasing the number of layers consistently leads to worse performance on the downstream task. Similarly, reducing the embedding size and thereby decreasing the capacity of the model also limits the model's ability to learn patterns which are useful for OOD generalization.

A.4 DIVERSITY

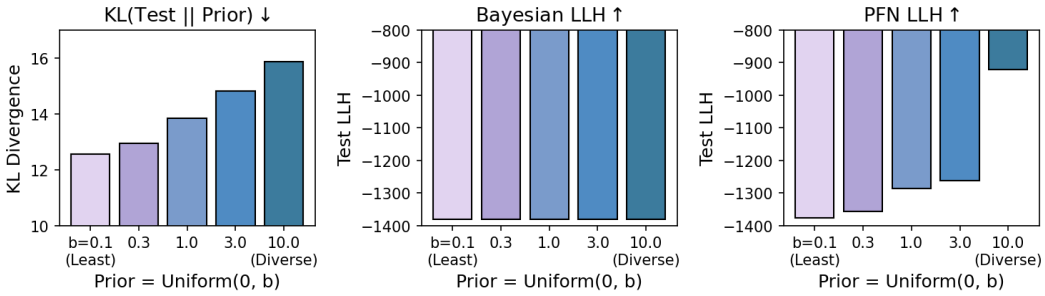


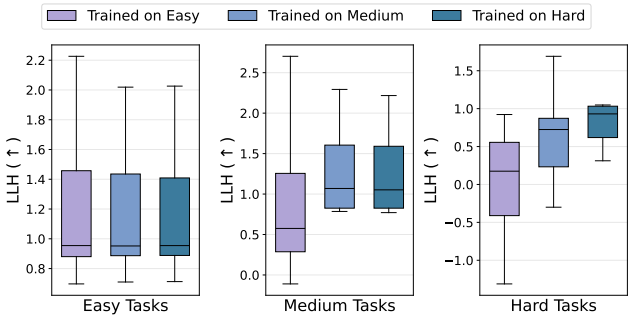
Figure A.5: **Bayesian and PFN performance under increasing prior diversity in Gaussian mean estimation.** For the downstream task $\mu = -1$, the training prior is $\mu \sim \text{Uniform}(0, b)$, with b increasing from left to right. The left panel shows the divergence between the downstream task and the prior; the middle and right panels show Bayesian posterior predictive and PFN test log-likelihoods, respectively, on 100 holdout observations. Results average 20 evaluation draws per prior width.

We include further training details in Appendix B.

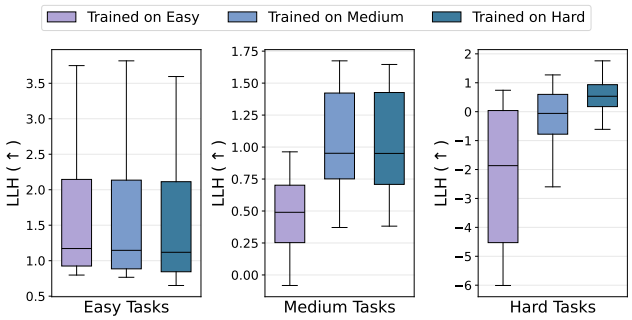
Linear Regression We ask whether the same behavior as Gaussian mean estimation occurs in a more complex setting, where the downstream task is a linear regression problem. During training, we generate lines with positive slopes $m \sim \text{Uniform}(0, a)$, $x_i \sim \text{Uniform}(-1, 1)$, and $y_i \sim \mathcal{N}(mx_i, 0.1)$, where a controls the diversity of the training distribution. At test time, we evaluate on the downstream task $m = -0.3$. As in the Gaussian mean setting, we find that increasing the diversity of the training distribution leads to better generalization on the downstream task, even though it moves the prior further from the downstream task (Figure 3, middle).

2D Classification We extend this analysis to a 2D classification problem. We draw inputs from $\mathcal{N}(0, I_2)$ and generate binary labels using decision boundaries through the origin whose slopes are sampled from $m \sim \text{Uniform}(0, a)$. We then evaluate on a boundary with slope $m = -1$. As in the previous two settings, we find that increasing the diversity of the training distribution leads to better generalization on the downstream task, even though it moves the prior further from the downstream task (Figure 3, right).

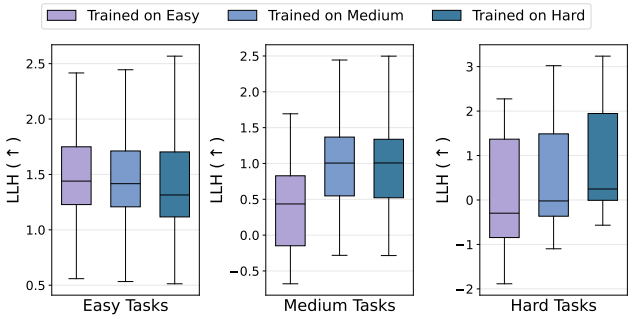
A.5 COMPLEXITY



(a) Node Retention Probability

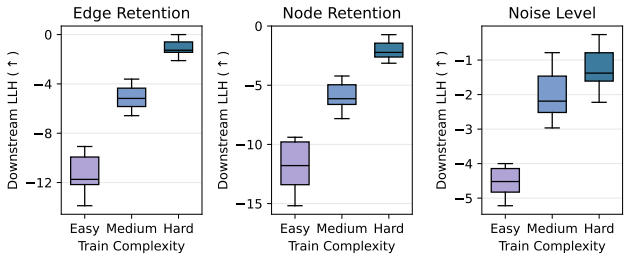


(b) Edge Retention Probability

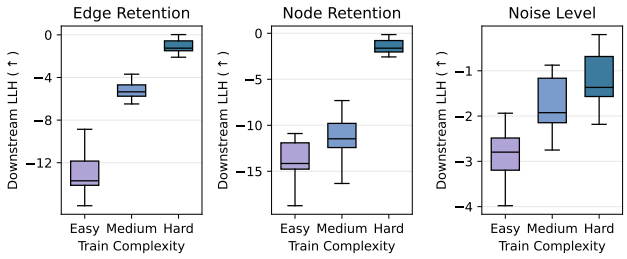


(c) Noise Level

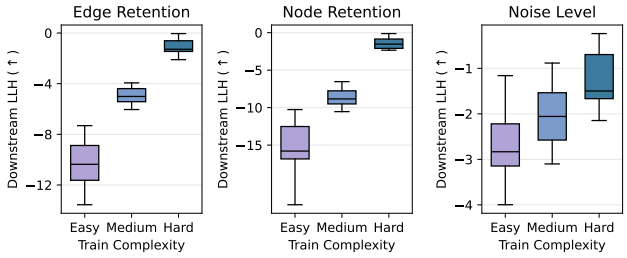
Figure A.6: Models trained on more complex priors perform well across a wide range of complexities, but models trained on overly simple data do not generalize to more complex distributions. We train models on the SCM prior and vary the hyperparameters corresponding to three different difficulty settings as specified.



(a) Models with 4 layers and 128 width

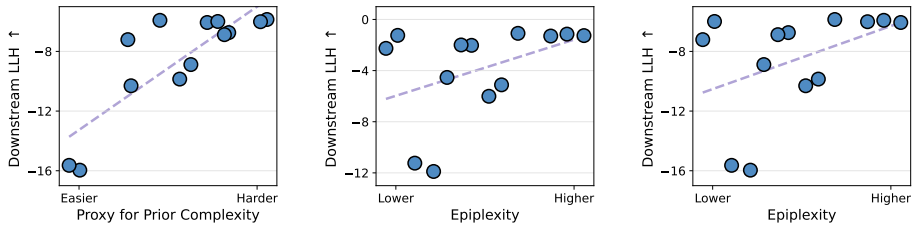


(b) Models with 6 layers and 256 width



(c) Models with 8 layers and 512 width

Figure A.7: **Increasing the complexity consistently leads to improved performance on UCI regression datasets across model sizes.** We replicate Figure 7 across more architectures.



(a) Proxy: Train loss quantile, Task: TabArena

(b) Proxy: Epiplexity, Task: UCI datasets

(c) Proxy: Epiplexity, Task: TabArena

Figure A.8: **The complexity of the prior data-generating distribution is correlated with stronger downstream performance across various datasets.** We replicate the proxy analysis from Figure 7 with different proxies and downstream tasks and find that our conclusions are robust.

A.6 GP PREDICTIONS COMPARED TO PFN PREDICTIONS ON UCI

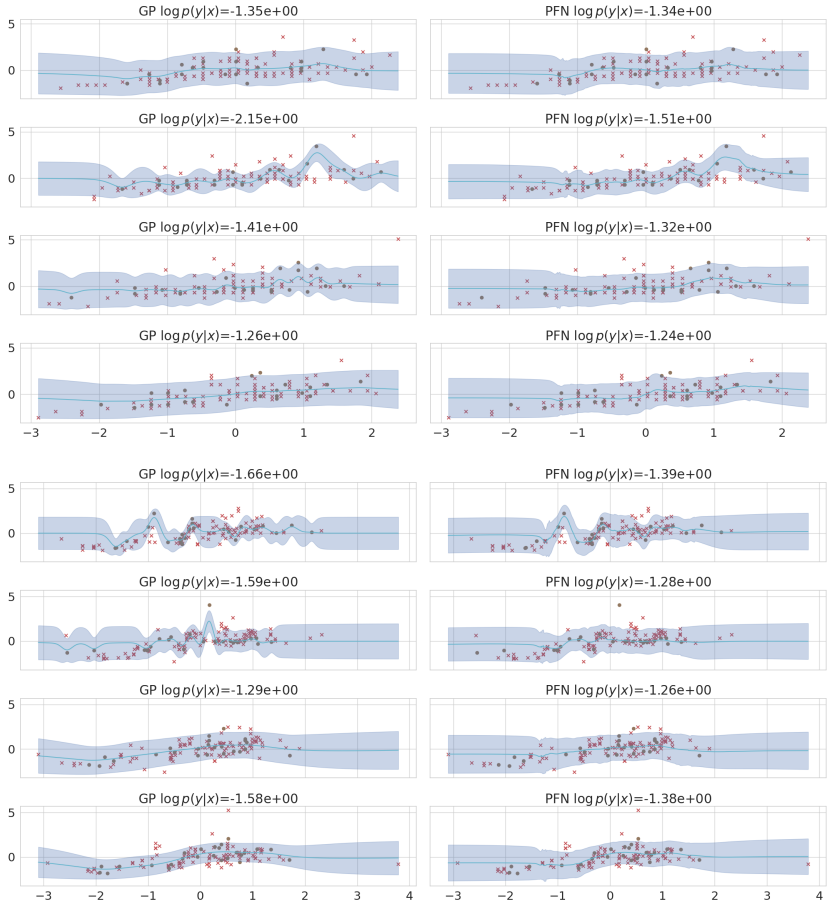


Figure A.9: Log posterior predictive values of our trained PFN models vs the actual GP log posterior predictive on Abalone UCI (first 4 rows) and on State UCI (last 4 rows).

In Fig. 1, we showed how a PFN trained on GP data outperformed the actual GP PPD, and we illustrate the posterior predictive distribution on UCI benchmarks in Fig. A.9. All the experimental details can be found in Section 3.3.

We argue that this graceful generalization to OOD data is a critical reason why PFNs generalize well in practice.

B TRAINING DETAILS

Lines and sines. We train the PFN for 100,000 batches. The model has four Transformer layers, an embedding dimension of 128, four attention heads, and a feedforward dimension of 512. We select the learning rate and batch size by in-distribution loss from Adam learning rates $\{10^{-4}, 10^{-3}, 3 \times 10^{-3}\}$ and batch sizes $\{32, 64\}$. The Bayesian comparator marginalizes over the line grid and the continuous uniform sine offset under the stated observation model.

Polynomial shift. For each training task, we sample a degree d uniformly from $\{0, 1, 2, 3\}$ and inputs uniformly from $[-1, 1]$. We draw an independent standard Gaussian coefficient for every term in an orthonormalized Legendre basis and divide each coefficient by $\sqrt{d+1}$, keeping the expected function variance constant across degrees. We add Gaussian observation noise with standard deviation 0.05. Each training task has eight context and 16 query observations. The PFN has three Transformer layers, embedding dimension 96, four attention heads, and feedforward dimension 192. It has a Gaussian output head and is trained by Gaussian negative log likelihood for 15,000 steps with batch size 64 using AdamW with learning rate 3×10^{-4} and weight decay 10^{-4}.

At evaluation, we generate 512 shared tasks at each degree $d \in \{3, 4, 6, 8\}$ using the same input, coefficient, scaling, and noise rules. Each task has eight context and 64 test observations. The posterior under the training prior marginalizes exactly over its four degree components. The degree-matched posterior uses the test generator's degree and coefficient prior. We compute both posteriors by Gaussian conditioning within each component and, for the training-prior posterior, weighting the components by their context-data likelihoods. We report mean squared error and mean predictive log likelihood on the same test tasks for all methods.

Gaussian-process prior and UCI comparison. We generate training tasks from a one-dimensional zero-mean Gaussian process with kernel $k_\ell(x, x') = \exp[-(x - x')^2/(2\ell^2)]$. The kernel parameter ℓ and the observation-noise variance are each sampled from $\{0.01, 0.03, 0.1, 0.3, 1, 3\}$, and inputs are sampled uniformly from $[-1, 1]$. The PFN uses four Transformer layers, embedding dimension 128, four attention heads, feedforward dimension 256, and a Gaussian output head. We train it for 100,000 batches of 32 tasks with Adam at learning rate 10^{-4}. The number of context observations varies from 3 to 50, and each training task includes 100 query observations. For the Bayesian comparator, we compute the posterior predictive for each of the 36 hyperparameter combinations and average them using their posterior weights given the context observations.

For each UCI input–target pairing, we form evaluation problems with 25 context and 100 test observations, as described in Section 3.3. We use the same problems to evaluate the PFN and the finite-mixture Gaussian-process posterior.

Diversity PFNs for Gaussian mean, linear regression, and 2d classification. The model has four Transformer layers, embedding dimension 128, four attention heads, feedforward dimension 512, and a Gaussian output head. We train for 100,000 batches of 32 tasks using Adam with learning rate 10^{-4}.

SCM-prior complexity. The generator begins with a four-layer MLP with eight hidden units and three input causes. We use tanh activations, Gaussian-initialized weights, and additive Gaussian noise at non-root nodes. Five nodes after the first layer are selected as observed inputs and another node as the target. In separate experiments, we vary edge retention, node retention, or noise level while holding the other generator settings fixed, as described in Section 5. We train each PFN for 500,000 batches with a fixed context length of 50 observations. The PFN has eight Transformer layers, embedding dimension 128, four attention heads, and feedforward dimension 512.

C AI ASSISTANCE

We used Codex and Claude throughout the research process to assist with writing and debugging code, monitoring training and evaluation runs, and editing the manuscript for clarity and errors. We reviewed and validated all AI-assisted code. The experimental design and scientific conclusions were determined by the authors.