From Epoch to Sample Size: Developing New Data-driven Priors for Learning Curve Prior-Fitted Networks

Tom J. Viering[1] **Steven Adriaensen**[2] **Herilalaina Rakotoarison**[2] **Frank Hutter**[3,2]

[1]Delft University of Technology, the Netherlands
[2]Machine Learning Lab, University of Freiburg, Germany
[3]ELLIS Institute Tübingen

Abstract Learning Curve Prior-Fitted Networks (LC-PFNs) perform Bayesian learning curve extrapolation for epoch learning curves. This paper explores designing new priors for LC-PFNs, focusing on sample-size learning curves that relate training set size to performance. We use the Learning Curve Database (LCDB), which contains diverse learning curve data for machine learning models on tabular data, to develop two data-driven priors. The first method fits MMF4 and WBL4 parametric curve models to the LCDB and uses a Gaussian mixture model to represent the prior over parametric curve parameters. The second method directly trains the LCPFNs on the LCDB curves, which we call the Real Data LC-PFN. We set up a proper meta-learning curve extrapolation benchmark with cross-validation on the LCDB for a careful evaluation. We show that both proposed priors improve upon the original LC-PFN, with the Real Data LC-PFN providing best results, improving in 78% of the experiments upon the old prior for extrapolating learning curves. Our study illustrates how to systematically design new priors for LC-PFN's in a metalearning framework, opening up their use for various curve modeling tasks in machine learning and beyond.

1 Introduction

Learning Curve Prior-Fitted Networks (LC-PFNs) were designed to extrapolate epoch learning curves that depict performance versus iterations or epochs (Adriaensen et al., 2023). These networks approximate Bayesian inference by training a transformer on a large set of synthetic curves generated from the prior, thus learning to predict their continuation and approximate the true posterior distribution (Müller et al., 2021). Adriaensen et al. (2023) improved on the prior developed by Domhan et al. (2015) using domain knowledge and hand-crafted tuning.

In this work, we extend the LC-PFN technique to *sample-size* learning curves that plot performance versus training set size. Both types of curves are often referred to as "learning curves" (Viering and Loog, 2022), but they differ fundamentally: epoch learning curves track the performance of a single model over time, while sample-wise learning curves compare the performance of multiple models trained to convergence on varying amounts of data. Extrapolating epoch-wise learning curves is useful for speeding up hyperparameter tuning using multi-fidelity techniques (Klein et al., 2022; Wistuba et al., 2022; Kadra et al., 2023; Rakotoarison et al., 2024) and algorithm selection (Ruhkopf et al., 2022). Sample-wise learning curves offer different insights and can also expedite hyperparameter tuning or model selection, especially for models that do not produce epoch-wise curves, such as K-Nearest-Neighbor classifiers (Mohr and van Rijn, 2023; Viering and Loog, 2022). They have also been used to accelerate hyperparameter tuning of algorithms like Support Vector Machines through Multi-Fidelity Bayesian Optimization (Klein et al., 2017).

Additionally, sample size curves are valuable for estimating the data requirements of machine learning projects (Viering and Loog, 2022). In real-world applications where data or annotations are costly, accurately estimating the necessary labeling budget is crucial before starting large-scale projects. Approximately 51% of machine learning projects in industry face challenges due to

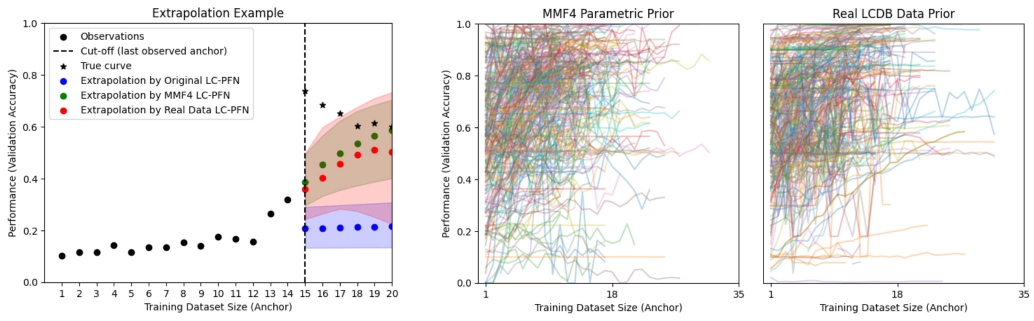

Figure 1: Left: Example extrapolations of a curve from the LCDB obtained by our two methods, contrasted to that of the original LC-PFN (more examples in Appendix, Figure 8). Anchor indicates the training set size. Middle / Right: samples produced by our parametric MMF4 LC-PFN and non-parametric Real Data LC-PFN prior (LCDB samples plus augmentation).

insufficient data (Dimensional Research, 2019). Traditionally, this estimation is done by fitting a parametric model to a learning curve and extrapolating it to determine the required data amount (Viering and Loog, 2022). This technique has been applied in various domains, such as machine translation (Kolachina et al., 2012), DNA microarray data (Mukherjee et al., 2003; Hess and Wei, 2010), and is now gaining interest in the deep learning community, where these curves are often referred to as "scaling laws" (Hestness et al., 2017; Mahmood et al., 2022).

Our study focuses on extrapolating learning curves for traditional machine learning models using a large database of learning curves from 20 classification models on 246 OpenML datasets, called the Learning Curve Database (LCDB) (Mohr et al., 2022). To apply LC-PFNs to these sample-wise curves, it is crucial to develop an accurate prior. Mohr et al. (2022) conducted the largest study on learning curve extrapolation, finding that most curves are well-modeled by the parametric models MMF4 and WBL4. However, Mohr et al. (2022) also noted that some curves exhibit unusual behaviors, such as peaks and non-monotonicity, where model performance decreases with more data (see also the surprising curves in Figure 1, left). These behaviors are challenging to model with parametric approaches, raising the question of how to develop a prior for these learning curves. Next to systematically developing two new datadriven priors for the LC-PFN, our study shows how to build a metalearning experiment to evaluate metalearned curve extrapolators on the LCDB.

2 Methods

We propose two data-driven approaches to develop priors for LC-PFNs, see Figure 1 for an overview (middle, right). The first approach models the curves using parametric forms and develops a data-driven prior over these parametric curves through density estimation. This approach maintains the Bayesian interpretation, and allows us to generate arbitrary amounts of training data, but may not capture all curve behaviors accurately because of the parametric assumption. The second approach, Real Data LC-PFN, involves training the LC-PFN model from scratch on real learning curve data from the LCDB. This method can learn intricate details missed by the parametric approach, such as multiple maxima, at the risk of overfitting the LCDB. One could argue that in training on real data, effectively using the data distribution as prior, we lose the Bayesian interpretation. However, we would like to argue that our augmented real data prior can be viewed as encoding the prior belief that *curves will look similar to curves seen thus far.* In what follows, we provide high-level descriptions of each approach; further details can be found in Appendix A.

2.1 Method 1: Parametric MMF4 and WBL4 LC-PFN

We perform curve fitting using the same procedure as Mohr et al. (2022) for all individual learning curves of the LCDB using the parametric forms MMF4 and WBL4. The parameter values of the fit and an estimate of the Gaussian noise (assumed to be homogeneous for a single curve, obtained after fitting using maximum likelihood) found from all individual curves form a new dataset which is modeled using density estimation. Before modeling the curve parameters, we transform the curve parameters using a quantile transform per dimension, so that the resulting data is normally distributed afterward to ensure proper scaling. We then model the density with a mixture of Gaussians with a full covariance matrix and we tune the number of mixtures on the validation set using the Negative Log Likelihood of the curve parameters per fold. We sample curve parameters and noise levels from the final tuned Gaussian Mixture Model (GMM), invert the quantile transformation to get the parameters for the parametric forms, and use them to generate new curves which are used to train the LC-PFN.

2.2 Method 2: Training LC-PFNs Directly on Real Data (Real Data LC-PFN)

We largely follow the experimental protocol of Adriaensen et al. (2023) to train on accuracy learning curves from the LCDB, with a few adjustments. We sample the curve length for the batch and then fill the batch with curves of the appropriate length, sampled uniformly from the LCDB. We select a cut-off point uniformly at random for each curve that is evaluated or trained on, where the points before the cut-off are observed points, and the points after the cut-off are extrapolation targets. These latter points are used for training the LCPFN with the Cross Entropy loss. As the training sets are relatively small, roughly 45.000 curves, compared to the 10M different synthetic curves used in the original LC-PFN and parametric LC-PFN, some degree of overfitting is likely. To counteract, we consider a variant adopting a simple custom *data-augmentation* (Shorten and Khoshgoftaar, 2019) scheme performing a random linear transformation on the original curve only for training, see Appendix A.3 for more details.

3 Experiments

First we briefly discuss the experimental setup; then, we delve into the results and discussion.

3.1 Experimental Setup

Both approaches are data-driven and therefore prone to overfitting, requiring a careful evaluation. We use five-fold cross-validation. One fold is used as validation (used for tuning hyperparameters of the GMM) and another fold is used as test; the remaining 3 folds are used for training. The folds are over 20 learners and 246 datasets. By cross-validating over learners and datasets, we can evaluate Unseen Datasets (UD), Unseen Learners (UL), and Unseen Datasets and Unseen Learners (UDUL). More information regarding the cross validation procedure is given in the Appendix (see also Figure 4). In terms of meta-learning jargon, the folds can be considered 'meta-train', 'meta-validation' and 'meta-test' sets, where these form learning curve datasets to learn from. When rotating folds, we rotate both learners and datasets at the same time. This way, we can perform 5-fold cross validation using only 5 iterations over learners and datasets. The evaluation is aggregated over meta-test sets, leading to 62795 curves for UD, 62852 UL, 21052 curves for UDUL in the test set.

The LCDB contains various learning curves, such as train, validation and test learning curves, that were obtained by splitting the original classification datasets in train, validation and test sets (not to be confused with our meta-train, meta-validation and meta-test sets of the learning curve data). We only use the validation accuracy curves of the LCDB since their performances are more independent. Because of the different splittings, the curves have 25 realizations, which we call *individual* curves. We extrapolate these individual learning curves which is a significantly harder extrapolation task than previously considered (in previous studies curves are averaged over multiple training sets to smoothen them (Mohr et al., 2022; Viering and Loog, 2022)). Finally, note that the LCDB training set sizes s are sampled according to $s = 2^{((i+7)/2)}$, where $i = 1, 2, \ldots, 35$. The curves generally do not have the same length due to finite data set sizes. We feed the transformers a set of (i, y)-values, where y is the validation accuracy. For more detail see Appendix A.4.

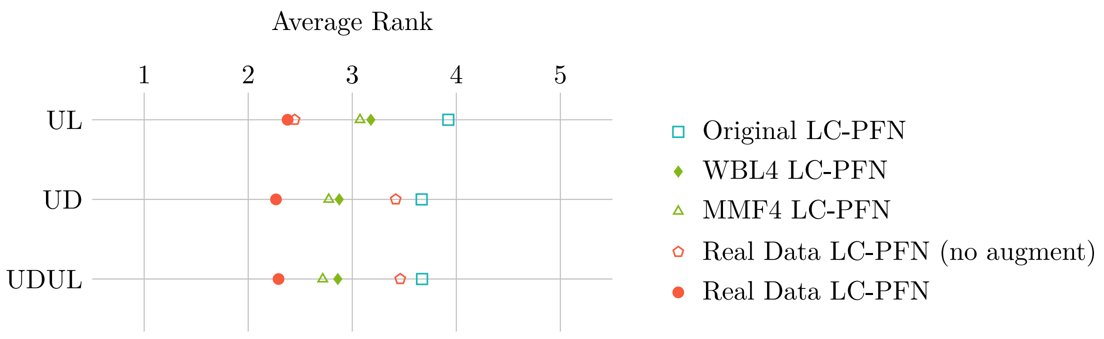

Figure 2: Aggregate results of Rank of the Negative Log Likelihood (NLL) per curve. Scenarios: Unseen Dataset (UD), Unseen Learner (UL), Unseen Dataset Unseen Learner (UDUL). Lower is better.

3.2 Results and Discussion

First, we give a high-level overview of the results in Figure 2. This figure shows the average rank across all evaluations in terms of the Negative Log Likelihood (NLL) per curve. We obtain one value per curve by first averaging the performances over all the unseen target anchors, and then we perform a second average over the number of observed anchors, to get to a single NLL per curve.

We observe that, as expected, the Original LC-PFN does not perform well, as its prior was designed for epoch curves. Furthermore, the Real Data LC-PFN without augmentation does not perform well and significantly overfits the small LCDB data, except for the Unseen Learner scenario. This overfitting was diagnosed by studying the epoch learning curve of the LC-PFN. This was expected because of the small amount of data compared to the amount of synthetic data used. Furthermore, we observe that for Method 1 using the curve generators, MMF4 ranks better than WBL4, which is in line with earlier findings (Mohr et al., 2022). The best rank is obtained by Method 2, i.e., the Real Data LCPFN that uses data-augmentation. For completeness, we also do a pairwise comparison of all methods with the original LC-PFN in Table 1, which agrees: the Real Data LC-PFN improves the most upon the original LC-PFN, in 78%-80% of the cases.

We now delve into a more detailed comparison with the 3 best approaches in Figure 3 for the most challenging scenario of the Unseen Learner Unseen Dataset (see the Appendix for the other scenario's). This evaluation is inspired by Kielhöfer et al. (2024) and Mohr et al. (2022). The empirical CDF over all evaluations (left plot) illustrates that the MMF4 and Real Data LC-PFN indeed seem to substantially improve over the original LC-PFN (the CDF is dominated). On the other hand, the Real Data LC-PFN and the MMF4 LC-PFN seem to perform similarly, especially, their CDF's cross for low NLL. To study the effect of the cut-off and prediction horizon, we rank these 3 methods for each (anchor observed, anchor target) pair (middle plot). Longer LCDB curves are rarer (see the right plot), thus, not all (observed, target) pairs weigh equally in the CDF.

The Real Data LC-PFN seems to work best in the majority of the cases; this is in line with expectations because the data augmentation addresses the overfitting issues and the MMF4-model has a strong parametric bias, which does not account for some of the strange learning curves in the LCDB. That it learns non-trivial patterns such as non-monotonicity can be observed in Figure 8 (Appendix). Returning to Figure 3, the Real Data LC-PFN does not work well for the largest anchor targets, we believe because these curves are rarer in the LCDB, which may lead to overfitting. For zero points observed, clearly original LC-PFN should not perform well, because it was not developed for training set size curves. However, it is a bit unclear to us why the original LC-PFN outperforms the MMF4 and Real Data LC-PFN baselines for 1 observed anchor. One conjecture is that this setting may be more prone to overfitting and requires stronger regularization. The results of the other scenarios are qualitatively similar (see Figures 6 and 7 in Appendix), except that the

Real Data LC-PFN has less issues with longer curves in Unseen Learner and Unseen Data scenarios, further justifying that it may be slightly overfitted to learners and / or datasets.

	MMF4 LC-PFN	WBL4 LC-PFN	Real Data LC-PFN (no augment)	Real Data LC-PFN
UL	70.99%	68.08%	72.68%	80.54%
UD	70.67%	67.85%	49.69%	78.50%
UDUL	71.56%	68.05%	48.88%	78.58%

Table 1: Percentage of cases the model is better than Original LC-PFN (Adriaensen et al., 2023)

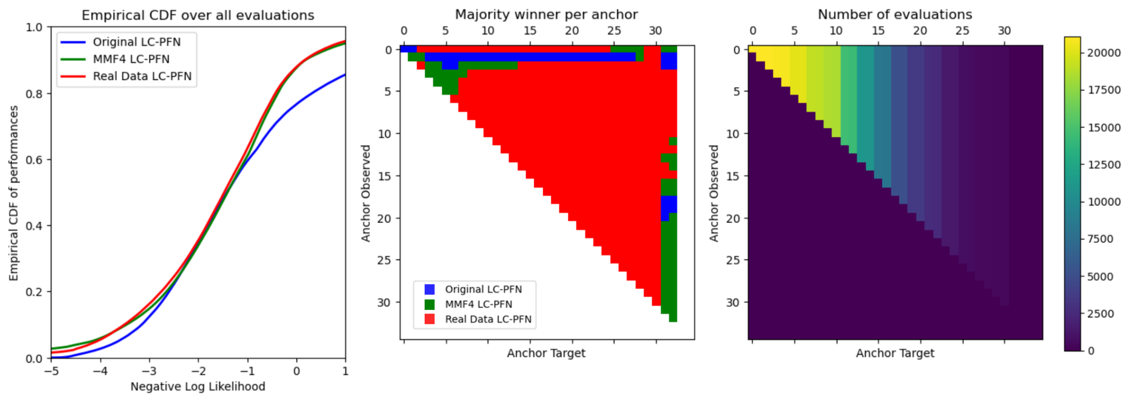

Figure 3: More detailed results for the scenario Unseen Dataset Unseen Learner (UDUL). Left: empirical CDF over all evaluations, showing that the MMF4 LC-PFN and Real Data LC-PFN outperform the Original LC-PFN. Middle: shows which algorithm wins the majority of the evaluations for a specific anchor observed and anchor target. Right: amount of curves under evaluation.

4 Future Work and Limitations

We have observed that the LC-PFN models trained on real data do not seem to converge to zero training loss. This may indicate suboptimal tuning. Therefore we want to include finetuning and early stopping on the validation in the future. Now, only the parametric LC-PFNs use the validation set for tuning hyperparameters, thus the comparison is in favor of the parametric LC-PFNs, yet the Real Data LC-PFN works best. Real and synthetic data could also be mixed. We believe the parametric prior could still be useful even though its performance is suboptimal, for example for interpretability. To that end a more simplified prior could be desirable — it is not clear whether it is necessary for such a complex parametric prior (tuning results in +-200 Gaussian mixtures). One difficulty is that for most curve parameters it is unclear what is a valid range (leading us to the quantile approach) and that the curve parameters seem to be dependent. Finally, it remains surprising to us that the Real Data LC-PFN performs well even without augmentation for the scenario Unseen Learner; suggesting that overfitting to datasets may in fact be also useful.

5 Broader Impact Statement

Extrapolating learning curves is useful for estimating the amount of data needed and can lead to improvements in multi-fidelity hyperparameter optimization algorithms. The first will likely have a positive impact on society due to less failed machine learning projects and ventures, and additionally may lead to cost savings for data collection. More efficient hyperparameter tuning is generally desireable in the context of green-machine learning: by making these algorithms more efficient, less compute-heavy tuning needs to be performed, leading to potential energy savings.

Acknowledgements. Frank Hutter acknowledges the financial support of the Hector Foundation. Steven Adriaensen, Herilalaina Rakotoarison, and Frank Hutter acknowledge funding by the European Union (via ERC Consolidator Grant Deep Learning 2.0, grant no. 101045765). Tom Viering, Steven Adriaensen, Herilalaina Rakotoarison, and Frank Hutter acknowledge funding by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. This work wouldn't have happened without a research visit by Tom Viering made possible by the TAILOR Collaboration Exchange Fund. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Funded by
the European Union

References

Adriaensen, S., Rakotoarison, H., Müller, S., and Hutter, F. (2023). Efficient bayesian learning curve extrapolation using prior-data fitted networks. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Dimensional Research (2019). Artificial Intelligence and Machine Learning Projects are Obstructed by Data Issues: Global Survey of Data Scientists, AI Experts, and Stakeholders. Technical report, Dimensional Research.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*.

Hess, K. R. and Wei, C. (2010). Learning curves in classification with microarray data. In *Seminars in oncology*, volume 37, pages 65–68. Elsevier.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. (2017). Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*.

Kadra, A., Janowski, M., Wistuba, M., and Grabocka, J. (2023). Scaling laws for hyperparameter optimization. *Advances in Neural Information Processing Systems*, 36.

Kielhöfer, L., Mohr, F., and van Rijn, J. N. (2024). Learning curve extrapolation methods across extrapolation settings. In *International Symposium on Intelligent Data Analysis*, pages 145–157. Springer.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pages 528–536. PMLR.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2022). Learning curve prediction with bayesian neural networks. In *International conference on learning representations*.

Kolachina, P., Cancedda, N., Dymetman, M., and Venkatapathy, S. (2012). Prediction of learning curves in machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22–30.

Mahmood, R., Lucas, J., Acuna, D., Li, D., Philion, J., Alvarez, J. M., Yu, Z., Fidler, S., and Law, M. T. (2022). How much more data do i need? estimating requirements for downstream tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 275–284.

Mohr, F. and van Rijn, J. N. (2023). Fast and informative model selection using learning curve cross-validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Mohr, F., Viering, T. J., Loog, M., and van Rijn, J. N. (2022). Lcdb 1.0: An extensive learning curves database for classification tasks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–19. Springer.

Mukherjee, S., Tamayo, P., Rogers, S., Rifkin, R., Engle, A., Campbell, C., Golub, T. R., and Mesirov, J. P. (2003). Estimating dataset size requirements for classifying dna microarray data. *Journal of computational biology*, 10(2):119–142.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. (2021). Transformers can do bayesian inference. In *International Conference on Learning Representations*.

Rakotoarison, H., Adriaensen, S., Mallik, N., Garibov, S., Bergman, E., and Hutter, F. (2024). In-context freeze-thaw bayesian optimization for hyperparameter optimization. *arXiv preprint arXiv:2404.16795*.

Ruhkopf, T., Mohan, A., Deng, D., Tornede, A., Hutter, F., and Lindauer, M. (2022). Masif: Meta-learned algorithm selection using implicit fidelity information. *Transactions on Machine Learning Research*.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48.

Viering, T. and Loog, M. (2022). The shape of learning curves: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7799–7819.

Wistuba, M., Kadra, A., and Grabocka, J. (2022). Supervising the multi-fidelity race of hyperparameter configurations. *Advances in Neural Information Processing Systems*, 35:13470–13484.

Submission Checklist

1. For all authors…

    (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

    (b) Did you describe the limitations of your work? [Yes]

    (c) Did you discuss any potential negative societal impacts of your work? [No]

    (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes]

2. If you ran experiments…

    (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes]

    (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] The code and the used splits will be released later, but we will release them to ensure that benchmarking on the LCDB will be done properly in future metalearning studies for curve extrapolators.

    (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] We used 5-fold CV.

    (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [No] No, however, we always report the number of curves under evaluation, which we believe is quite large ensuring significance of the results.

    (e) Did you report the statistical significance of your results? [No] No, see above.

    (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] Yes, we have used the LCDB (Mohr et al., 2022).

    (g) Did you compare performance over time and describe how you selected the maximum duration? [No] No, since all methods used the same amount of training time for the LC-PFN models and they were all trained with the same number of examples.

    (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

    (i) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We ablated the data augmentation and we ablated the parametric curve model, furthermore, we compared two approaches towards developing datadriven priors.

3. With respect to the code used to obtain your results…

    (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [No] No, we will release this later.

    (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [No] No, we will release this later.

    (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [No] No, we will release this later.

   (d) Did you include the raw results of running your experiments with the given code, data, and instructions? [No] No, we will release this later.
   (e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [No] No, we will release this later.

4. If you used existing assets (e.g., code, data, models)...
   (a) Did you cite the creators of used assets? [Yes] Yes, we cited the LC-PFN prior work and the LCDB.
   (b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [No] Not applicable.
   (c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] Not applicable.

5. If you created/released new assets (e.g., code, data, models)...
   (a) Did you mention the license of the new assets (e.g., as part of your code submission)? [No] Not applicable.
   (b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [No] No, we will release them later.

6. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No] Not applicable.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No] Not applicable.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No] Not applicable.

7. If you included theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [No] Not applicable.
   (b) Did you include complete proofs of all theoretical results? [No] Not applicable.

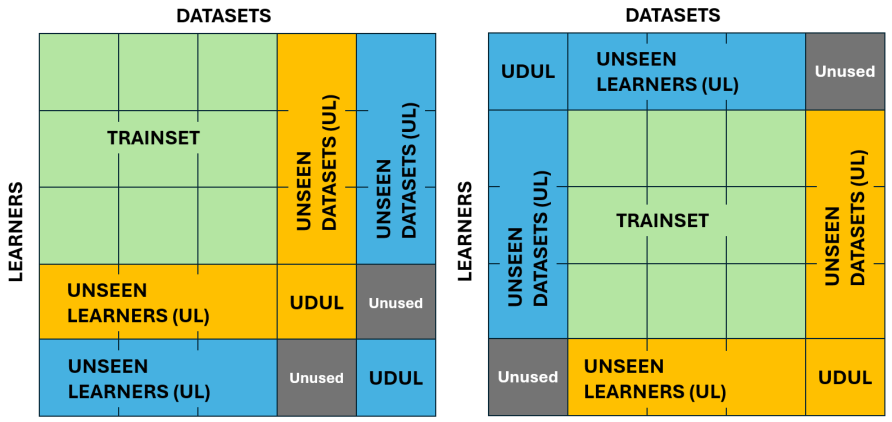

Figure 4: Cross Validation procedure for cross validating over the LCDB for generating meta-train (green), meta-validation (orange) and meta-test sets (blue). Gray sets are unused in our experiments. Left: first iteration of the cross validation, right: second interation of the cross validation, out of the 5 cross-validation iterations.

A Further Details about Methods and Experimental Setup

A.1 Cross Validation procedure and Background on LCDB

The Learning Curve Database by Mohr et al. (2022) is a database that consists of learning curves of 20 traditional machine learning algorithms (such as SVM, QDA, LDA, ...) on 246 datasets. The curves are generated by bootstrap-sampling multiple training, validation and test sets. These sets are used to build the learning curve: samples are discarded from the training set and performance is measured on the validation set. The training set sizes are sampled according to an exponential schedule, the training set sizes are also referred to as anchors.

The total classification dataset is split first in a temporary and test set, and the temporary set is splitted in train and validation sets. This splitting is governed by two random seeds: the outer seed controls the test split, and the inner seed controls the validation seed. Multiple inner and outer seeds are used, resulting in multiple (strongly correlated) learning curves for a pair of (learner, dataset)-combination.

Since the curves are strongly correlated, we should not randomly split the whole LCDB randomly in train, validation and test sets — because in that case we risk overly optimistic performance estimates, as very similar curves will occur in the train and test set (because they might correspond to the same learner and dataset but differ only by inner or outer seed). Instead, we use group cross-validation, where the groups are given by datasets and learners, to obtain more fair performance estimates. To avoid a combinatorial explosion, we use two cross-validations, one with respect to the learner, and one with respect to the dataset, and we rotate the folds simultaneously.

The process is illustrated in Figure 4. For clarity let us describe the procedure in words as well. Assume we have three learners: A, B, and C, and 3 datasets: 1, 2, 3. We are going to perform the procedure with a 3-fold cross validation over learners and datasets. In the first iteration, we will put (A,1) in the training set, and (B,1), (C,1) will end up in the validation and test sets respectively, which will be the unseen learner (UL) partition, since the dataset 1 was observed by the training set. (A,2) and (A,3) will end up in the validation and test sets respectively, and will be in the unseen dataset partition, since the learner A has been seen during training. (B,2) and (C,3) will end up in the unseen learner unseen dataset (UDUL) partition. This would be the first cross validation iteration. In the next iteration, all the indices are increased by one modulo 3, e.g. 1 turns into 2, 2 into 3, 3 into 1, and similarly for A,B,C. Some combinations remain unused, since it was not clear to us whether to assign them to validation or test-sets, in hindsight its clear that one has to uniquely break the tie, this could potentially enlarge the validation and test sets further for the partition UDUL.

One should be careful to note that there are essentially two levels on which splitting in train, validation and test sets is done. The first splitting is done by the LCDB, this splits datasets, such as MNIST, into train, validation and test sets to be able to generate learning curves. These curves form the data for our study. We split the database of curves again in different train, validation and test sets. Since we are essentially doing the dataset splitting on a higher level, its appropriate to call these meta-learn, meta-validate and meta-test sets. Note that in the first splitting, the splitting is done on tabular classification data, while in the second level, the splitting is done on a curve level. Thus, an entire curve ends up in the meta-train, meta-validation or meta-test set.

A.2 MMF4 and WBL4 LC-PFN Details

The parametric forms for MM4 is $y(x) = (ab+cx^d)/(b+x^d)$. We model x as the training set size, and y as the validation accuracy (note that they are *not* log-scaled before fitting). For WBL4 the formula is $y(x) = -bexp(-ax^d) + c$. We perform Levenberg-Marquadt fitting on the individual curves with 5 random restarts with finite differencing, where the best parameters are kept, following Mohr et al. (2022). Only the full-length learning curve is fitted with the mean squared error. After fitting, we take the mean squared residual as estimate for σ.

The quantile transformer uses 1000 quantiles to estimate the distribution of the parametric curve parameters per dimension (where the noise is an additional dimension). Samples within quantiles are lineairly interpolated with the output quantiles of the normal distribution. This transformation was performed, because for example parameter b of the MMF4 formula seems to have extremely large and unpredictable bounds (10^50) and thus cannot be expected to be well modeled by a Gaussian.

The Gaussian Mixture model thus models $p(a', b', c', d', \sigma')$ in the transformed space. These are transformed back using the inverse quantile transformation to obtain a, b, c, d, σ in the original space. Afterward, the synthetic curve is generated using the formula of the parametric form of WBL4 or MMF4, and Gaussian noise with mean zero and standard deviation σ^2 is added to each point on the curve. Finally, before providing the curve to the LC-PFN for training, the x-values are renumbered to i-values, to be consistent with the evaluation.

Next to the Gaussian Mixture model we also experimented with a Kernel Density Estimator using Gaussian Kernel, but we found that the Guassian Mixture Model generally gave better results in terms of NLL on the curve parameters in the transformed space on the validation sets, thus we resorted to the Gaussian Mixture Model. The number of mixtures was tuned via Gridsearch over 10^1 to 10^3, with equally spaced hyperparameters in log-space, with 100 steps from 1 to 3 (base 10). The hyperparameter used was different per fold. For each fold, the hyperparameter that resulted in the best average performance over the 3 scenario's (UD, UL, UDUL) in terms of the NLL was used for generating synthetic curves for training the LC-PFN.

Sometimes we observe curves that do not fall in the expected range of $[0, 1]$ for the accuracy learning curves. When the curve falls in the range $[0 - \epsilon, 1 + \epsilon]$ we clip the curve into the range $[0, 1]$, otherwise, the curve is rejected and not used for training the PFN. We set $\epsilon = \frac{1}{2}$.

Note that the training of these parametric LC-PFN's follows largely the same procedure as outlined in Section 2.2. We train with 10M synthetic curves. However, when sampling a batch, we first sample a batch of curves from the LCDB. We only keep the missing values of the batch, and we replace the actual curve values by synthetic curves as generated by this prior. The additional data-augmentation described in Section 2.2 is not used, since the curves are all already augmented by the learned noise.

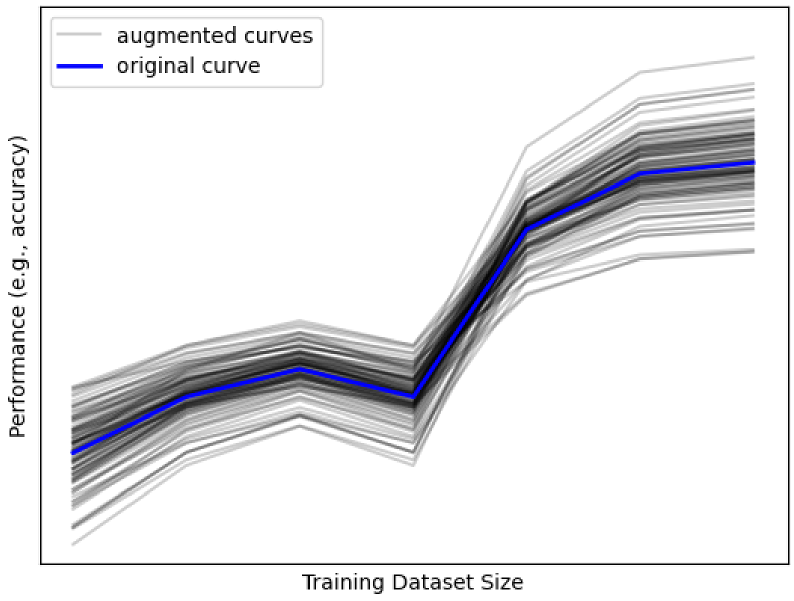

Figure 5: Illustration of the augmentation strategy used in Real Data LC-PFN.

A.3 Data-Augmentation Strategy

Data augmentation is a popular technique for counteracting overfitting in deep learning (Shorten and Khoshgoftaar, 2019) in the context of little training data. It involves creating and training on randomly perturbed variants of existing data points. In our experiments, we apply mostly small random linear transformation on the original curve that are guaranteed to be bounded in [0,1]. An example of the augmentations produced is shown in Figure 5. We realize this by (1) sampling 10 values uniformly at random in [0,1], (ii) for all 45 ordered pairs (a,b), with $a < b$, we consider the linear transformation mapping the smallest (y_{min}) and largest (y_{max}) curve value onto a and b, respectively, i.e.,

$$y_{\text{augmented}} = \frac{b-a}{y_{\text{max}} - y_{\text{min}}} \cdot (y_{\text{original}} - y_{\text{min}}) + a$$

(iii) From these candidate transformed curves, we pick the one with minimal point-wise aggregated mean squared difference, biasing towards smaller transformations. Crucially, in our experimental setup, data augmentations are only performed on the training sets, but not on the test sets, as the latter may lead to biased performance estimates.

A.4 Dealing with Missing Values and the Original LC-PFN and more minor details

The LCDB contains quite some missing values because some machine learning models do not complete training successfully; these errors seem to be irrecoverable. In our prior, when determining the length of the curve, we ignore missing values. For method 2, which trains the LCPFN on real curves, missing values in the input (observed partial curve) are dropped by removing the (i,y)-value pair, and i values with missing y's are never used as a target for the extrapolation. For the parametric methods, it would be beneficial for the transformer to also learn to deal with missing values, since it will observe these at test time. To this end, we sample a curve from the LCDB, and we use the missingness to also remove values from a synthetic curve as generated by a generators for method 1. The original LC-PFN (Adriaensen et al., 2023) is also fed in the context with missing values removed, and manages to deal with it despite not being trained with them. However, the original which was trained for training curves does observe radically different x-values. To soften this domain shift, we feed in the values $i = 1, \ldots, 35$ instead of the actual training set sizes s, which are more similar to the epoch numbers it was trained on.

All the LC-PFN models were trained with 10M example curves (either synthetic or real) with the same hyperparameters as in the original work of Adriaensen et al. (2023). We used 1000 buckets where for each different model the bucket edges were determined according to the prior.

B More Detailed Results

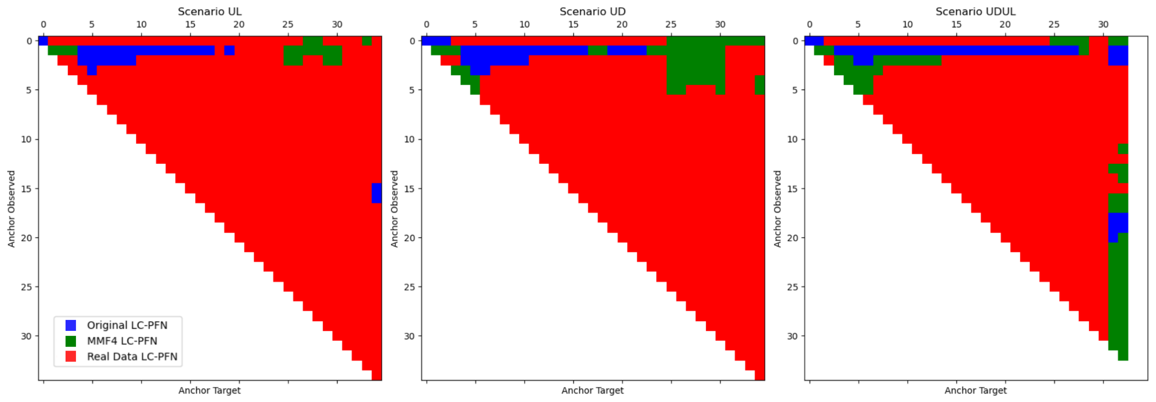

Figure 6: In this figure we display the method which wins the most for a particular anchor observed and anchor target. Winning means that it achieved the lowest Negative Log Likelihood (NLL). Winning the most means that, in the majority of the experiments for a particular (anchor observed, anchor target) pair, the method achieves the most wins. Meaning of the scenario's: (UD) Unseen Dataset, (UL) Unseen Learner, (UDUL) Unseen Dataset Unseen Learner.

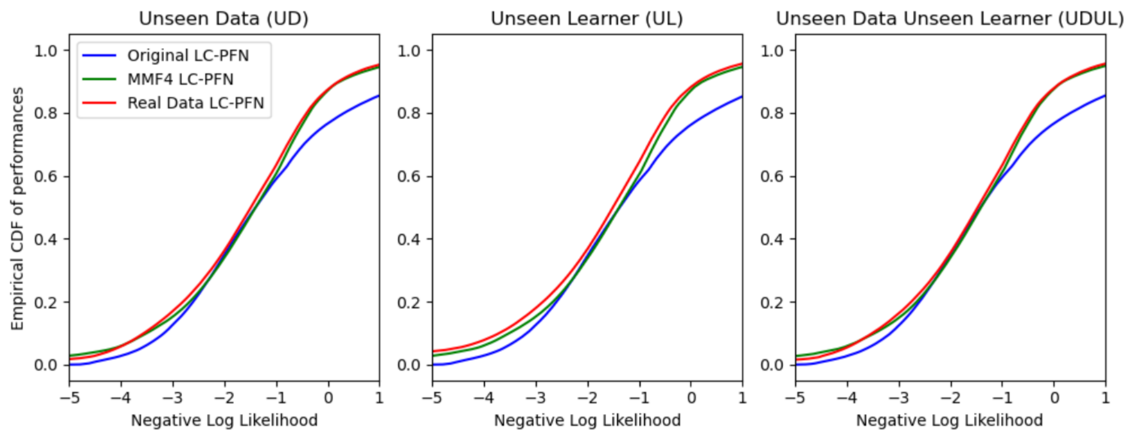

Figure 7: In this figure we display the empirical cumulative distribution function of each method for all scenarios.

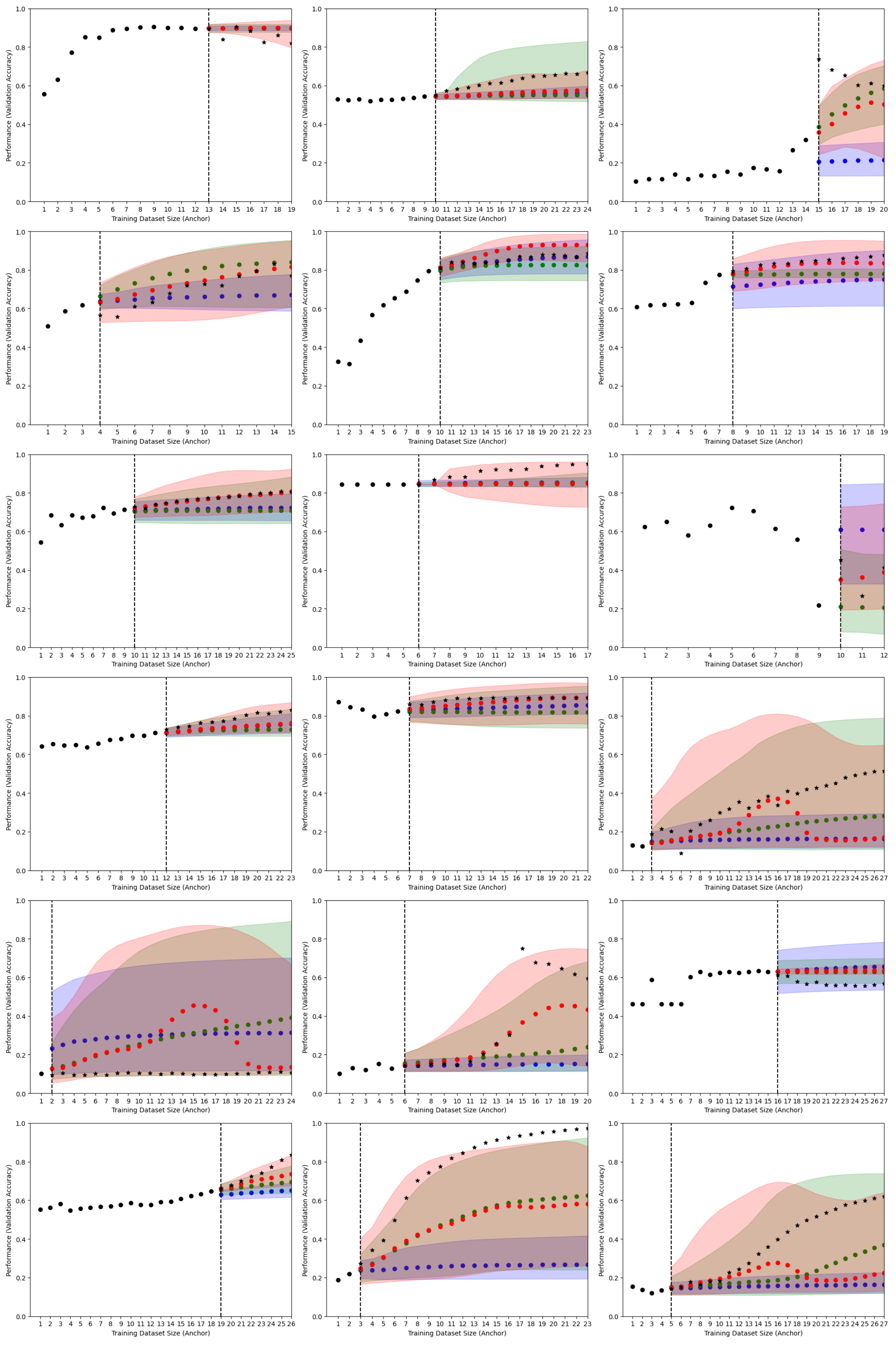

Figure 8: More Extrapolation Examples. See Figure 1 (left) for more info, legend, etc.