Rethinking Description Length: A TabPFN-Based Approximation of Bayesian Mixture Codes

Afiq Abdillah Effiezal Aswadi [1] **Ria Jeffrey** [1] **Susan Wei** [2]

Abstract

The Minimum Description Length principle, a model selection framework based on Occam's razor, is typically studied through universal codes, and the associated codelengths. Blier & Ollivier (2018) studied the MDL principle for deep neural networks. They compare various codes, and find that a prequential code, based on neural networks trained through stochastic gradient descent, has shorter codelength than variational and two-part codes. Recent developments in deep learning point to a better way to approximate the Bayes mixture code than the variational code. Specifically Hollmann et al. (2023) present a transformer architecture, called Tabular Prior-Data Fitted Networks (TabPFNs), which are trained on synthetic data generated from a vast array of prior-likelihood pairs, and is encouraged to learn the corresponding Bayes posterior predictive distribution. We then use TabPFN to induce a code through in-context learning and demonstrate on real world datasets from the OpenML-CC18 suite that the resulting code is consistently shorter than the prequential code corresponding to MLPs.

1. Introduction

Minimum Description Length (MDL) (Rissanen, 1978; Hansen et al., 2001) is a mathematical formulation of Occam's razor for statistical models. The MDL principle has historically been studied through the notion of universal codes (Rissanen, 1978). Essentially, a code is probability distribution. Universal coding is a method of losslessly compressing data without knowledge of the data-generating distribution. Examples of universal codes include the two-part code (Rissanen, 1983), the Bayes code (Grünwald et al., 2005), prequential code (Rissanen, 1984; Dawid, 1984), and the normalised maximum likelihood (NML) code (Barron et al., 1998; Myung et al., 2006).

Coding methods differ in the way they compress information losslessly. To every coding method is an associated codelength: the length of the code required by the model to losslessly describe the data, i.e., the negative log density of the probability distribute. A shorter codelength implies a better rate of compression. Relative to the data-generating distribution, the code with the shortest codelength is considered optimal. We note that it is impossible to have a universal code that optimally compresses every possible data-generating distribution. In other words, what is optimal from one data-generating distribution may not be optimal for another.

Blier & Ollivier (2018) study the notion of MDL on deep neural networks. They considered a prequential plug-in code where the network weights are learned through stochastic gradient descent, which we will call the **prequential SGD plug-in code**. By looking at different neural networks for MNIST and CIFAR, they empirically found that a prequential SGD plug-in code produced much shorter codelengths than the two-part and variational code.

The variational code, being an approximation of the (universal) Bayes code was previously considered the state of the art for deep neural networks. The superiority of the prequential SGD plug-in code relative to the variational code would seem to suggest that it is also superior to the Bayes code. But this would be the wrong conclusion to draw. For highly complex neural networks such as the ones employed in Blier & Ollivier (2018), variational approximation is very poor.

In this work, we are motivated to take another look at the comparison between the prequential SGD plug in code and the Bayes code, but approximated through more powerful machinery. Recent work by Müller et al. (2024) showed that transformers can be trained to perform Bayesian inference. In particular, a transformer architecture called a Prior-Data Fitted Network (PFN) trained on prior-likelihood pairs are

[1]School of Mathematics and Statistics, University of Melbourne, Australia [2]Department of Econometrics and Business Statistics, Monash University, Australia. Correspondence to: Afiq Abdillah Effiezal Aswadi <aeffiezalasw@student.unimelb.edu.au>, Ria Jeffrey <alex.jeffrey@unimelb.edu.au>, Susan Wei <susan.wei@monash.edu>.

Proceedings of the 1st ICML Workshop on Foundation Models for Structured Data, Vancouver, Canada. 2025.

able to approximate a Bayes posterior predictive distribution (PPD) for some dataset given to the PFN at inference time.

We present a codelength induced by TabPFN (a PFN that is pretrained on synthetic data generated from a vast mixture of prior-likelihood pairs). The resulting codelength can be viewed as an approximation to a Bayes code corresponding to some prior-likelihood pair represented by the PFN weights. The codelengths induced by TabPFN are different from prequential plug-in codelengths in that there are no weights being learned – the model approximates a PPD for the data through in-context learning.

We perform experiments on OpenML (Vanschoren et al., 2014) datasets to compare the code induced by TabPFN to the prequential SGD plug-in code induced by training various MLPs. We find that the PFN code is consistently shorter than the prequential SGD plug-in code. The PFN is able to choose a model for the data with little training examples, and the choice of model generally does not vary over more training examples. This is unlike models trained through SGD, which would require many training examples to be appropriately trained on the data.

Setting Throughout the main paper we focus exclusively on classification tasks. We have data of the input-output type where input $x \in \mathbb{R}^d$ and output $y \in \{1, \dots, K\}$. We denote the data as $x_{1:n} = (x_1, \dots, x_n)$ and similarly for $y_{1:n}$. In Appendix A, we consider a toy regression example.

2. Prequential SGD-plug in code

In this section, we first briefly review the **predictive plug-in code**, also known as the "**prequential plug-in**" code, which were introduced independently by Rissanen (1984) and Dawid (1984). Then we will detail the particular modern construction employed in Blier & Ollivier (2018) in the context of deep learning.

The plug-in code relative to a parametric model $\{p_\theta(y|x)|\theta \in \Theta\}$ sequentially codes each outcome y_i conditional on x_i using an estimator based on all the data up to that point, $\hat{\theta}(x_{1:i-1}, y_{1:i-1})$. The associated codelength for input-output pair (x_i, y_i) is simply the log loss, $-\log p_{\hat{\theta}(x_{1:i-1}, y_{1:i-1})}(y_i|x_i)$. The total codelength we will denote

$$\mathcal{L}^{\text{preq}}(y_{1:n} \mid x_{1:n}) = \sum_{i=1}^{n} -\log p_{\hat{\theta}(x_{1:i-1}, y_{1:i-1})}(y_i \mid x_i). \tag{1}$$

Note that the code is conditional on $x_{1:n}$ as we are interested only in the conditional model. Traditionally $p_\theta(y|x)$ would have been a regular statistical model and $\hat{\theta}$ the maximum likelihood estimator, in which case the predictive plug-in code is also known as the **maximum likelihood plug-in code**.

Fitting this to purpose for modern deep learning, Blier & Ollivier (2018) offer two innovations on the classic maximum likelihood plug-in code which we now describe. If $p_\theta(y|x)$ is a probabilistic neural network, it is more natural to let $\hat{\theta}$ represent the network parameter trained via stochastic optimization, e.g., stochastic gradient descent (SGD). This is the first departure.

Next since it is burdensome to retrain the model at each observation, Blier & Ollivier (2018) suggests to split the observations into S buckets of observations $1 = t_0, \dots, t_S = n$, and retrain and compute the incremental codelength at each bucket. Inserting uniform encoding for the first bucket gives the **batched prequential SGD-plug in code**, computed sequentially:

$$\begin{aligned}\mathcal{L}^{\text{batched-SGD}}(y_{1:t_s} \mid x_{1:t_s}) &= t_1 \log K \\ &+ \sum_{i=1}^{s-1} -\log p_{\hat{\theta}_{t_i}}(y_{t_i+1:t_{i+1}} \mid x_{t_i+1:t_{i+1}})\end{aligned} \tag{2}$$

where $\hat{\theta}_{t_s}$ is a network parameter trained on $x_{1:t_s}$ through stochastic optimization. We define the final codelength as $L^{\text{batched-SGD}}(y_{1:n}|x_{1:n})$.

In Blier & Ollivier (2018), the **batched prequential SGD-plug in code** in (2) was found to produce shorter codelengths relative to variational codes corresponding to the same neural network. However this result is not particularly surprising given that we know from Graves (2011) that variational codes tend to be very far from the Bayes mixture code they are attempting to approximate.

A main motivation in our work is to leverage the new technique of learning Bayes mixture codes using the prior-data fitted network (PFN) (Müller et al., 2024), essentially a transformer trained on exchangeable distributions with an underlying prior and likelihood. We expect the resulting approximation to the Bayes mixture code to be far superior to variational codes, thus offering a more meaningful comparison to the batched prequential SGD plug-in code.

3. Bayes mixture code via PFNs

Let's begin with a brief review of prior-data fitted networks (PFNs). The exposition here follows closely from Müller et al. (2024). To reduce notational clutter, let us consider an unsupervised setting where we wish to produce short codes for observed data $y_{1:n}$. If we posit a prior-likelihood pair $p_\theta(y)$ and $\pi(\theta)$, then the associated Bayes mixture codelength is

$$\mathcal{L}^{\text{Bayes}}(y_{1:n}) = -\log \int \prod_{i=1}^{n} p_\theta(y_i)\pi(\theta)\mathrm{d}\theta. \tag{3}$$

The Bayes mixture code has many desirable qualities but it is generally intractable. We may employ MCMC sampling but that proves challenging when θ is high-dimensional. Furthermore if p_θ is a neural network model, it can be very discomfiting to specify a prior distribution on the neural network weights θ which have no physical meaning.

PFNs are quite ingenious in that they pose the learning of the Bayes posterior predictive density (PPD) as an optimisation problem. A PFN is trained on a large synthetic dataset $\{y^m_{1:n+1}\}_{m=1}^M$ where each $y^m_{1:n+1}$ is drawn from the joint distribution

$$p(y_{1:n+1}) := \int \prod_{i=1}^{n+1} p_\theta(y_i)\pi(\theta)\,\mathrm{d}\theta. \tag{4}$$

Essentially this is a two-step procedure, for each m, first drawn $\theta^m_0 \sim \pi(\theta)$ and then draw $y^m_{1:n+1}$ conditionally independent given θ^m_0, i.e., y^m_i are i.i.d. from $p_{\theta^m_0}(y)$.

Now let T_ϕ be a transformer sequence model with weights ϕ which outputs a predicted distribution over the classes. Consider the objective function[1]

$$\arg\min_\phi \sum_{m=1}^{M} -\log T_\phi(y^m_{n+1}|y^m_{1:n}). \tag{5}$$

Let $\hat{\phi}$ be the minimizer and we call the resulting transformer a **BayesPFN**. Then it is straightfoward to show that $T_{\hat{\phi}}$ is an estimate of the Bayes PPD corresponding to the likelihood-prior pair $p_\theta(y)$ and $\pi(\theta)$, that is

$$p(y_{n+1}|y_{1:n}) := \int p_\theta(y_{n+1})\pi(\theta|y_{1:n})\,\mathrm{d}\theta,$$

where $\pi(\theta|y_{1:n}) \propto \prod_{i=1}^n p_\theta(y_i)\pi(\theta)$ is the posterior distribution.

Since the marginal likelihood can be written in terms of the Bayes PPD, as a product of one-step ahead predictive densities,

$$\int \prod_{i=1}^{n} p_\theta(y_i)\pi(\theta)\mathrm{d}\theta = \prod_{i=1}^{n} p(y_i|y_{1:n})$$

as soon as we have estimates of the PPD, we can also estimate the Bayes mixture codelength for a new dataset. Specifically, let $y_{1:n}$ be drawn from $\int \prod_{i=1}^n p_\theta(y_i)\pi(\theta)\,\mathrm{d}\theta$, then we can use $T_{\hat{\phi}}$ to approximate the Bayes mixture codelength as

$$\mathcal{L}^{\text{BayesPFN}}(y_{1:n}) := \sum_{i=1}^{n} -\log T_{\hat{\phi}}(y_i|y_{1:i-1}). \tag{6}$$

[1]The actual PFN objective function is a modification of (5) where we repeatedly draw i from a list $\{1,\ldots,n\}$ which encourages the PFN to learn $p(y_i|y_{1:i-1}) := \int p_\theta(y_i)\pi(\theta|y_{1:i-1})\,\mathrm{d}\theta$ for $i = 1,\ldots,n$.

The conditional version of this is

$$\mathcal{L}^{\text{BayesPFN}}(y_{1:n}|x_{1:n}) := \sum_{i=1}^{n} -\log T_{\hat{\phi}}(y_i|x_i, y_{1:i-1}, x_{i-1}). \tag{7}$$

To approximate the Bayes mixture codelength for a new dataset, we prompt the trained PFN with our data, and the PFN chooses a PPD of the data through in-context learning. Note the PFN parameter ϕ is fixed- any PPD approximated by the PFN lies latent in the PFN weights.

We check for how well the PFN code approximates the Bayes code in Appendix A. In particular, we consider a theoretical setting where an asymptotic approximation of the Bayes code is available. We consider a two layer regression network with $\tanh$ activations and standard normal noise. We find that a PFN trained on a the underlying likelihood-prior has codelength which is a very good approximator to the theoretical Bayes mixture codelength.

4. TabPFN

Rather than train transformers from scratch on synthetic datasets sampled from (4) as described above, we instead use a pretrained PFN known as TabPFN (Hollmann et al., 2023; 2025). We are still approximating a Bayes mixture code, but the underlying likelihood-prior pair is quite complex. Being pretrained, the Bayes mixture code induced by TabPFN is significantly easier to compute because it just requires forward passes through the network.

The code induced by TabPFN is just as in (6) and (7) but the transformer is instead the TabPFN. Note neither BayesPFN nor TabPFN actually explicitly learn an explicit parameter θ for the data. In fact, any model approximated by PFN will lie latent in PFN's (fixed) weights, activated by the "prompt" (the data that is conditioned on) through in-context learning. This is different from traditional plug-in codes which learns the model parameter θ based on the dataset.

In the spirit of (2), we introduce a batched version of the TabPFN codelength. Recall we split the observations into S buckets of observations $1 = t_0,\ldots,t_S = n$. From Müller et al. (2024), we note that for some input $D = (x_{1:n}, y_{1:n}, x_{n+1:N})$, then for some $x_i \in x_{n+1:N}$ the attention heads of the PFN attends only to $(x_{1:n}, y_{1:n}, x_i)$. From this, we have the sequential **batched TabPFN codelength**:

$$\mathcal{L}^{\text{batched-TabPFN}}(y_{1:t_s} \mid x_{1:t_s}) = t_1 \log K - \sum_{j=1}^{s-1}\sum_{i=1}^{t_{j+1}-(t_j+1)} \log T_{\hat{\phi}}(y_i|x_i, y_{1:t_j}, x_{1:t_j}) \tag{8}$$

and we denote the full batched TabPFN code as $\mathcal{L}^{\text{batched-TabPFN}}(y_{1:n} \mid x_{1:n})$.

5. Experiments

We conclude with a set of experiments on real-world data. Our goal is to compare the batched prequential SGD-plug in code in (2) and the TabPFN-induced code in (8). Due to TabPFN's architectural constraints, we only consider tabular classification data for our empirical comparison. But of course in principle the concept of codelength generalises to non-tabular data as well. For instance, Blier & Ollivier (2018) considers CIFAR (Krizhevsky, 2012) and MNIST (LeCun et al., 2010).

Datasets are obtained from the OpenML (Vanschoren et al., 2014) suite, as in (Hollmann et al., 2023). Datasets with missing values are removed. We end up with 24 classification datasets. Further information on the datasets can be found in 1.

Codes The batched prequential SGD plug-in codelength in (2) is computed on MLPs of three different sizes. We also compute the batched-TabPFN code. Further implementation details can be found in Appendix B.

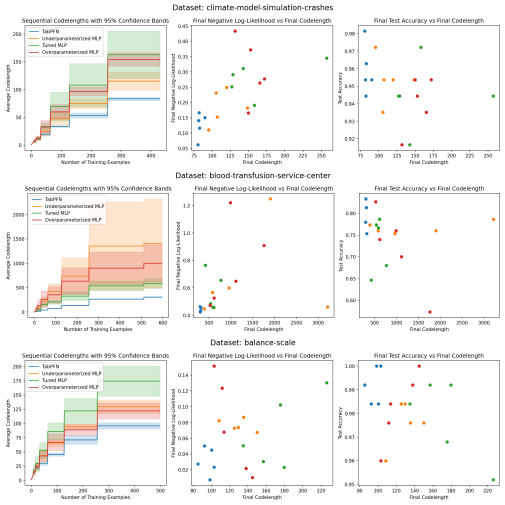


Figure 1. We study three MLP batched prequential SGD-plug in codes and the batched TabPFN code across three datasets. Comparison of sequential codelengths, averaged over 5 train-test splits with 95% confidence bands (left), negative log-likelihood (lower is better) vs final codelength for the five individual train-test splits (middle) and classification test accuracy vs. final codelength for the five individual train-test splits (right).

Evaluation For each dataset, we perform evaluations over 5 train-test splits. For a dataset with total number of samples N, we split into a train set of length n and a test set of length m. We denote the train set as $(x_{1:n}^{\text{train}}, y_{1:n}^{\text{train}})$ and the test set as $(x_{1:m}^{\text{test}}, y_{1:m}^{\text{test}})$. For stability reasons, we ensure that an observation of each class is contained in all train sets, and we also prepend an observation of each class to the train set to avoid the model assigning probability 0 to a class it has not seen. The training set is split into buckets $t_1 = \max(8, K), t_2 = 16, t_3 = 32, \ldots, t_S = n$.

We compute the negative log-likelihood (NLL) of the fully trained models. In the prequential SGD plug-in case, this is: $-\frac{1}{m}\sum_{i=1}^{m} \log p_{\hat{\theta}\left(x_{1:n}^{\text{train}}, y_{1:n}^{\text{train}}\right)}(y_i^{\text{test}}|x_i^{\text{test}})$, and in the TabPFN case: $-\frac{1}{m}\sum_{i=1}^{m} \log T_{\hat{\phi}}(y_i^{\text{test}}|x_i^{\text{test}}, x_{1:n}^{\text{train}}, y_{1:n}^{\text{train}})$.

We also compute the test accuracy of the fully trained models. Our results are shown in three types of plots, see Figure 1. In the first row, we show the average (over the train-test splits) sequential codelength against the number of training examples. In the second plot, we show the final NLL versus the final codelength as a scatter plot. In the third plot, we show classification test accuracy versus final codelength as a scatter plot.

5.1. Results

Due to space, we only show results for three of the 24 datasets in Figure 1. Numerical tables summarizing the different metrics for all 24 datasets can be found in Appendix B. Analogous plots to Figure 1 for the other 21 datasets can be found in Appendix C.

Across most of the datasets, we find that the final TabPFN codelength is consistently shorter than all three final batched prequential SGD plug-in codelength . We offer some conjectures on why TabPFN is so effective. TabPFN approximates a Bayesian posterior predictive density for the data through in-context learning. As TabPFN has been meta-trained on prior-likelihood pairs, the PPD approximated by TabPFN is contained in the (fixed) weights of the transformer, thus allowing prompting TabPFN to generalise faster than training a model from scratch. We also observe that TabPFN codelength varies less over different train-test splits than the batched prequential SGD plug-in codelength . This implies TabPFN choosing a model for the data with little training examples, and allows for a consistent model for the data throughout different train-test splits.

We also find that across most datasets, TabPFN manages to have a lower final NLL and higher test accuracy than the MLPs. This is aligned with our expectation.

We note that TabPFN does not uniformly outperform the MLPs. In certain datasets, for instance `cmc`, `car`, `mfeat-karhunen`, we find that the TabPFN NLL is not necessarily lower than the MLPs. We hypothesise that this comes from a fundamental limit of the TabPFN weights- as the TabPFN weights are fixed, there is a limit to the models TabPFN can represent, and therefore some models for data cannot be well represented by TabPFN, compared to a model explicitly trained on the data. We present the sequential NLL of the models for the 24 datasets in Appendix D. The findings here support our hypothesis: for most of

the 24 datasets considered, TabPFN "settles early", in that it apparently chooses a model for the dataset after just a few in-context examples; the model does not vary much over more in-context examples. In many instances, this leads to the MLP "catching up", and ending up with a NLL that is much closer, if not better than the TabPFN NLL. This highlights a potential advantage of TabPFN for small sample size settings.

References

Aoyagi, M. and Watanabe, S. Resolution of singularities and the generalization error with bayesian estimation for layered neural network. *Ieice Transactions - IEICE*, 10.

Barron, A., Rissanen, J., and Yu, B. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998. doi: 10.1109/18.720554.

Blier, L. and Ollivier, Y. The Description Length of Deep Learning Models, November 2018.

Dawid, A. P. Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984. ISSN 00359238, 23972327. URL http://www.jstor.org/stable/2981683.

Graves, A. Practical variational inference for neural networks. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K. (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

Grünwald, P. D., Myung, J. I., and Pitt, M. A. *Advances in Minimum Description Length: Theory and Applications*. The MIT Press, February 2005. ISBN 978-0-262-27446-3. doi: 10.7551/mitpress/1114.001.0001.

Hansen, M. H., , and Yu, B. Model Selection and the Principle of Minimum Description Length. *Journal of the American Statistical Association*, 96(454):746–774, June 2001. ISSN 0162-1459. doi: 10.1198/016214501753168398. URL https://doi.org/10.1198/016214501753168398. Publisher: ASA Website _eprint: https://doi.org/10.1198/016214501753168398.

Hollmann, N., Müller, S., Eggensperger, K., and Hutter, F. TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second, September 2023.

Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S. B., Schirrmeister, R. T., and Hutter, F. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, January 2025. ISSN 1476-4687. doi: 10.1038/s41586-024-08328-6.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

Krizhevsky, A. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Müller, S., Hollmann, N., Arango, S. P., Grabocka, J., and Hutter, F. Transformers Can Do Bayesian Inference, August 2024.

Myung, J. I., Navarro, D. J., and Pitt, M. A. Model selection by normalized maximum likelihood. *Journal of Mathematical Psychology*, 50(2):167–179, April 2006. ISSN 0022-2496. doi: 10.1016/j.jmp.2005.06.008. URL https://www.sciencedirect.com/science/article/pii/S0022249605000532.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12(null):2825–2830, November 2011. ISSN 1532-4435.

Rissanen, J. Modeling by shortest data description. *Automatica*, 14(5):465–471, September 1978. ISSN 0005-1098. doi: 10.1016/0005-1098(78)90005-5. URL https://www.sciencedirect.com/science/article/pii/0005109878900055.

Rissanen, J. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983. ISSN 00905364, 21688966. URL http://www.jstor.org/stable/2240558.

Rissanen, J. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636, 1984. doi: 10.1109/TIT.1984.1056936.

Vanschoren, J., Rijn, J. N. v., Bischl, B., and Torgo, L. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. ISSN 1931-0145, 1931-0153. doi: 10.1145/2641190.2641198. URL http://arxiv.org/abs/1407.7722. arXiv:1407.7722 [cs].

A. Sanity check on PFN approximation of the Bayes mixture code

If we use PFNs to approximate the Bayes mixture code corresponding to some likelihood-prior pair, $(p_\theta(y), \pi(\theta))$ as described in Section 3, a natural question is to ask how good the approximated code is. Since we rarely know the Bayes mixture code in complex models $p_\theta(y)$, we focus on a toy model that retains some of the essential features of a neural network but for which asymptotic results from singular learning theory are available to get a "ground-truth" for the Bayes mixture code-length.

The toy model in question is a two layer regression network with tanh activations and standard normal noise. For $x, y \in \mathbb{R}$, our model is:

$$p_\theta(y|x) = N(f(x,\theta), 1) \tag{9}$$

$$f(x,\theta) = \sum_{i=1}^{L} a_i \tanh(b_i x) \tag{10}$$

where L is the number of hidden nodes and $\theta = [a_1, ...a_L, b_1, ...b_L]$ with $N(\mu, 1)$ denoting the normal distribution with mean μ and unit variance.

Singular learning theory tells us, under certain conditions, that given a dataset generated from a fixed parameter vector θ_0, the Bayes mixture codelength can be asymptotically approximated as:

$$\mathcal{L}^{\text{Bayes}}(y_{1:n}|x_{1:n}) \simeq \left(\sum_{i=1}^{n} -\log p_{\theta_0}(y_i|x_i)\right) + \lambda(\theta_0)\log(n) - (m(\theta_0)-1)\log\log(n) + \text{const.}$$

where $\lambda(\theta_0)$ and $m(\theta_0)$ are coefficients related to the singularities of the parameter set of the model. We will refer to the first term on the right hand side of this equation, $(\sum_{i=1}^{n} -\log p_{\theta_0}(y_i|x_i))$ as the **empirical entropy**.

(Aoyagi & Watanabe) provides these coefficients for this model when we fix $\theta_0 = 0$, which is equivalent to sampling y iid from the standard normal distribution. In this case, the coefficients for our Bayes code approximation can be analytically derived as:

$$\lambda(0) = \frac{L + i^2 + i}{4i + 2}$$

$$m(0) = \begin{cases} 2, & i^2 = L \\ 1, & i^2 < L \end{cases}$$

where i is the largest integer such that $i^2 \leq L$.

BayesPFN To create the PFN model, we generate pretraining datasets by sampling θ from a standard normal prior distribution

$$\pi(\theta) = N(0, I)$$

and drawing x-values from the uniform distribution over the range $[-1, 1]$. That is,

$$x \sim U[-1, 1].$$

The y-values are sampled from the model given in Eq 9. Our PFN is trained on data $\{x^m_{1:n}, y^m_{1:n}\}$ for $m = 1, \ldots, M$ where $n = 1500$. The number of training datasets M was chosen per PFN to achieve convergence, but was typical between 80,000 and 6,000,000. For architecture details on the transformer used for the BayesPFNs, see Figure 3.

An important difference between the theory and the implementation is that the PFN model cannot model a truly continuous density function as its output. This is solved in (Müller et al., 2024) by using a Riemann distribution: discretising the output distribution into buckets and predicting the probability of the output distribution falling within each bucket. However, due to the nonlinarity of the log function, this discretisation will skew our estimation of the codelength away from the continuous case.

To account for this, we convert the continuous term in the empirical entropy into an identically bucketed discrete distribution by averaging the true data pdf over each bucket. We found that in practice this reduced the gap between the theoretical approximation and the calculated codelength.

Mathematically, this means our BayesPFN codelength is

$$\sum_i -\log T_{\hat{\phi}}(y_i \in [b_i, b_{i+1}] | x_{\leq i}, y_{<i})$$

and the empirical entropy is

$$\sum_i -\log(P(y_i \in [b_i, b_{i+1}] | x_i)$$

where $[b_i, b_{i+1}]$ is the bucket the i-th data point falls into and P is the measure under p_{θ_0} where $\theta_0 = 0$.

Thus let's define a normalized BayesPFN codelength:

$$\bar{\mathcal{L}}^{\text{BayesPFN}}(y_{1:n}|x_{1:n}) = \sum_i -\log T_{\hat{\phi}}(y_i \in [b_i, b_{i+1}] | x_{\leq i}, y_{<i}) - \sum_i -\log(P(y_i \in [b_i, b_{i+1}] | x_i) \tag{11}$$

Assessing the approximation quality of the Bayes PFN Let $\{y_{1:n}, x_{1:n}\}$, $n = 1500$ be a dataset generated according to Eq 9 with $\theta_0 = 0$. We call this an evaluation dataset.

To see whether $\bar{\mathcal{L}}^{\text{BayesPFN}}(y_{1:n}|x_{1:n})$ is a good approximation of the Bayes mixture code corresponding to $p_{\theta_0}, \pi(\theta)$, we compare it to

$$\lambda(\theta_0)\log(n) - (m(\theta_0) - 1)\log\log(n)$$

This will be referred to as the **theoretical approximation**.

Our experiments involved training 4 BayesPFN models on tanh priors with differing numbers of hidden nodes. The specific architecture of the BayesPFN is given in Figure 3. For each model, we then generated 100 evaluation datasets and computed the BayesPFN codelength curve on each dataset (as a function of dataset length). Finally, we subtracted the empirical entropy for the associated dataset from each curve and averaged across datasets to get the normalised BayesPFN codelength curves.

A.1. Results

Figure 2 plots the normalised BayesPFN codelength in experiments. These plots have iid evaluation data but differ in the datasets used to train the BayesPFN. In particular, each plot uses a different number of hidden nodes in the tanh network when generating data.

The plot shows the average codelength across 100 eval datasets at each prequential dataset length. The shared area shows a 95% confidence interval for the mean (i.e. +/- 1.96 standard error). The heading of each graph provides the number of hidden nodes along with the $\lambda(\theta_0)$ given by Aoyagi & Watanabe.

We perform regression to assess the quality of the approximation over a range of n's for $n > 100$. We chose not to fit both parameters are $\log(n)$ and $\log\log(n)$ are highly correlated on the data range. We perform the following regressions:

1. Regressing $\bar{\mathcal{L}}^{\text{BayesPFN}}(y_{1:n}|x_{1:n})$ against $\log n$ and an intercept.
2. When $m(\theta_0) \neq 1$, regressing $\bar{\mathcal{L}}^{\text{BayesPFN}}(y_{1:n}|x_{1:n}) + (m(\theta_0) - 1)\log\log n$ against $\log n$ and an intercept.

In both cases, when fitting we exclude a number of datapoints from the beginning as the approximation is known to only hold asymptotically for large n. The regression fit is somewhat sensitive to the threshold for n that is considered, but we found the results are relatively stable for $n > 10$. In this case, we chose to exclude the first $32 = 10^{1.5}$ datapoints.

Across the settings in Figure 2 we first see a strong logarithmic trend as predicted by the theory. We also see that BayesPFN is decent at recovering the true λ.

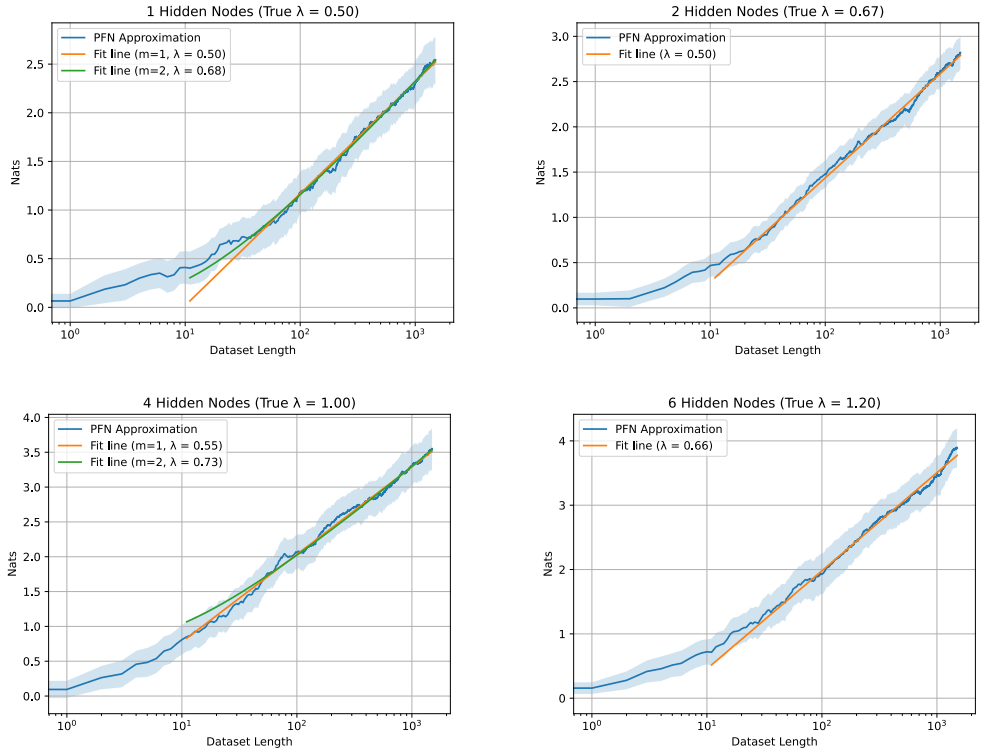


Figure 2. $\bar{\mathcal{L}}^{\text{BayesPFN}}(y_{1:n}|x_{1:n})$ as a function of n, mean plus or minus 1.96*standard error. Both fit lines are provided alongside estimated $\lambda(\theta_0)$. Each subplot shows a different PFN training dataset generated by different number of hidden nodes of tanh network.

```
    TransformerModel(
  (transformer_encoder): TransformerEncoderDiffInit(
    (layers): ModuleList(
      (0-3): 4 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256, bias=True)
        )
        (linear1): Linear(in_features=256, out_features=512, bias=True)
        (dropout): Dropout(p=0.0, inplace=False)
        (linear2): Linear(in_features=512, out_features=256, bias=True)
        (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.0, inplace=False)
        (dropout2): Dropout(p=0.0, inplace=False)
      )
    )
  )
  (encoder): Sequential(
    (0): Normalize()
    (1): Linear(in_features=1, out_features=256, bias=True)
  )
  (y_encoder): Linear(in_features=1, out_features=256, bias=True)
  (pos_encoder): NoPositionalEncoding()
  (decoder_dict): ModuleDict(
    (standard): Sequential(
      (0): Linear(in_features=256, out_features=512, bias=True)
      (1): GELU(approximate='none')
      (2): Linear(in_features=512, out_features=303, bias=True)
    )
  )
  (criterion): FullSupportBarDistribution()
)
```

Figure 3. Architecture of the model used for section A. This model has 2.4M total parameters.

B. Experimental details and additional results

We first describe the implementation details of the MLP codelengths. We use Scikit-learn (Pedregosa et al., 2011) to train the MLPs, with Adam (Kingma & Ba, 2017) as our stochastic optimizer for training the MLP.

The tuned MLP was hyperparameter tuned through grid search with the following parameter grid:

1. Hidden layer sizes $= [(32,), (64,), (32, 32), (64, 64), (32, 32, 32), (64, 64, 64), (128, 64, 32)]$
2. Learning rate = $[0.0001, 0.001, 0.01]$
3. L^2 regularisation term = $[0.00001, 0.0001, 0.001]$

through training a "tuned" multilayer perceptron (MLP), implementation details can be found in Appendix B. The hyperparameters for the MLP are tuned through grid search. We also compute the batched prequential SGD plug-in codelength on a smaller MLP and a larger MLP relative to the tuned MLP, with hyperparameters based on the tuned MLP. These three MLP codes are compared to the batched TabPFN code in (8).

The MLPs are trained through minimising the log loss, i.e., the cross entropy loss. At the beginning, it is the training data does not contain examples from a particular class. When computing the codelength, if the MLP assigns probability zero to the true class of some observation, the contribution to the codelength will become infinity (since $-\log(0) = \infty$). To avoid this, we added some small value $\epsilon = 1 \times 10^{-15}$ to the probability of the true class. This may explain some of the large jumps in the prequential plug-in codelengths seen in some figures in Appendix C. We note that TabPFN generally does not run into this issue and in general works well for low n. Perhaps this is due to the fact that it approximates a PPD, and so generally does not have overconfidence issues.

To avoid a model assigning probability zero to a class it hasn't seen during training, we prepend an observation of every class to the start of the training set.

To train the underparametrised MLP, we either remove the final hidden layer in the case where the tuned model has more than one hidden layer, otherwise we divide the number of nodes in the hidden layer by 2. To obtain the overparametrised MLP, we add an extra hidden layer to the MLP with the same number of nodes as the final layer of the tuned model. Note that the MLPs are retrained for each new batch. We do not perform partial fitting on observations from the new batch.

TabPFNv1 was used to compute the TabPFN codelength.

Dataset Details We use the same OpenML (Vanschoren et al., 2014) datasets from the OpenML-CC18 suite as in Hollmann et al. (2023). Dataset details are shown in Table 1. We note that the dataset details were also presented in Hollmann et al. (2023).

The OpenML-CC18 suite is specifically collated to not include simulated/artificial datasets. Datasets range from a sample size of 522 to 2000. Note the datasets were specifically chosen not to exceed this sample size limit, due to stability factors for TabPFN. The code for obtaining the datasets were based on the code from the original paper.

Further results We present further results in tables 2, 3, 4. The tables contain the respective final metric averaged over 5 train-test splits for all 24 datasets. We note that Hollmann et al. (2023) presents the AUC-OVO scores for each dataset.

Table 1. Summary of OpenML datasets used in experiments

	OpenML ID	Sample Size	Number of Features	Number of Classes
analcatdata_authorship	458	841	71	4
analcatdata_dmft	469	797	5	6
balance-scale	11	625	5	3
banknote-authentication	1462	1372	5	2
blood-transfusion-service-center	1464	748	5	2
car	40975	1728	7	4
climate-model-simulation-crashes	40994	540	19	2
cmc	23	1473	10	3
credit-g	31	1000	21	2
diabetes	37	768	9	2
ilpd	1480	583	11	2
kc2	1063	522	22	2
mfeat-fourier	14	2000	77	10
mfeat-karhunen	16	2000	65	10
mfeat-morphological	18	2000	7	10
mfeat-zernike	22	2000	48	10
pc1	1068	1109	22	2
pc3	1050	1563	38	2
pc4	1049	1458	38	2
qsar-biodeg	1494	1055	42	2
steel-plates-fault	40982	1941	28	7
tic-tac-toe	50	958	10	2
vehicle	54	846	19	4
wdbc	1510	569	31	2

Table 2. Average final codelength (Mean ± Std) over 5 different train-test splits.

	TabPFN	Underparameterized MLP	Tuned MLP	Overparameterized MLP
kc2	**171.24 ± 7.77**	1904.68 ± 72.95	1637.03 ± 310.74	2029.81 ± 617.73
climate-mode...	**83.29 ± 3.43**	115.17 ± 18.44	162.64 ± 48.64	154.25 ± 13.95
wdbc	**48.27 ± 4.64**	994.93 ± 627.92	152.92 ± 38.19	109.94 ± 8.48
ilpd	**254.11 ± 8.66**	824.32 ± 336.24	770.14 ± 400.50	730.51 ± 196.63
balance-scal...	**95.84 ± 6.31**	129.63 ± 13.60	174.55 ± 30.22	122.15 ± 16.24
blood-transf...	**304.08 ± 8.27**	1409.04 ± 1042.77	582.88 ± 114.34	999.80 ± 439.98
diabetes	**314.99 ± 9.21**	1140.89 ± 299.83	819.65 ± 197.18	974.18 ± 119.40
analcatdata-...	**65.18 ± 3.07**	155.08 ± 71.00	82.66 ± 23.94	95.39 ± 18.14
vehicle	**334.36 ± 12.16**	1171.38 ± 224.16	1055.94 ± 399.86	833.58 ± 117.09
tic-tac-toe	**313.94 ± 17.71**	596.18 ± 86.72	644.31 ± 128.16	823.47 ± 62.28
credit-g	**419.55 ± 5.72**	3238.78 ± 1823.81	1472.95 ± 1404.88	1456.67 ± 447.08
qsar-biodeg	**310.43 ± 11.20**	539.49 ± 126.48	522.69 ± 107.17	714.42 ± 51.96
pc1	**189.26 ± 5.60**	3217.83 ± 1898.35	1444.45 ± 581.37	1868.22 ± 599.55
banknote-aut...	**20.47 ± 1.80**	40.38 ± 5.26	43.09 ± 17.84	28.15 ± 11.60
pc4	**277.40 ± 6.13**	2672.05 ± 422.12	3156.71 ± 989.59	2873.96 ± 1293.90
cmc	**1127.63 ± 10.29**	1403.76 ± 258.64	1486.28 ± 222.51	1574.84 ± 371.55
pc3	**361.62 ± 16.35**	3539.02 ± 1257.38	3058.74 ± 514.67	3424.38 ± 1197.76
car	**242.12 ± 17.47**	329.58 ± 26.43	360.53 ± 57.36	443.20 ± 146.91
steel-plates...	**1219.17 ± 30.01**	23505.19 ± 1583.03	19091.50 ± 2633.55	23465.63 ± 1366.93
mfeat-zernik...	**870.49 ± 14.15**	5095.67 ± 1369.26	2842.34 ± 493.48	3340.28 ± 1135.40
mfeat-karhun...	599.30 ± 10.01	**490.92 ± 107.40**	626.31 ± 187.26	504.15 ± 14.05
mfeat-morpho...	**1215.47 ± 28.82**	20765.56 ± 1694.45	9244.82 ± 7835.24	10705.06 ± 3276.82
mfeat-fourie...	**1064.48 ± 32.69**	1506.16 ± 146.38	1425.96 ± 211.49	1731.72 ± 185.84

Table 3. Average Negative Log-likelihood of the final model (Mean ± Std) over 5 different train-test splits.

	TabPFN	Underparameterized MLP	Tuned MLP	Overparameterized MLP
kc2	**38.20 ± 2.22**	810.74 ± 50.32	430.16 ± 179.58	475.53 ± 248.09
climate-mode...	**13.75 ± 3.94**	20.02 ± 5.47	30.02 ± 5.73	32.70 ± 9.99
wdbc	**6.72 ± 3.88**	22.82 ± 3.96	23.92 ± 4.94	25.97 ± 3.61
ilpd	**58.88 ± 4.60**	64.89 ± 8.55	64.48 ± 5.60	68.62 ± 11.59
balance-scal...	**3.83 ± 1.93**	9.58 ± 0.87	8.42 ± 5.24	9.37 ± 6.92
blood-transf...	**66.38 ± 2.26**	99.70 ± 44.67	84.69 ± 18.70	113.31 ± 41.53
diabetes	**81.42 ± 6.06**	110.17 ± 16.92	100.18 ± 7.89	96.52 ± 4.60
analcatdata-...	**1.53 ± 0.95**	7.03 ± 4.86	4.80 ± 5.19	4.09 ± 4.31
vehicle	**50.57 ± 6.05**	161.86 ± 28.16	136.02 ± 16.85	115.31 ± 15.65
tic-tac-toe	**44.81 ± 3.53**	56.60 ± 22.12	73.70 ± 39.34	77.06 ± 34.03
credit-g	**98.48 ± 5.16**	812.88 ± 532.16	275.11 ± 299.76	222.64 ± 79.57
qsar-biodeg	**75.03 ± 12.27**	85.90 ± 17.46	89.11 ± 10.16	98.03 ± 8.68
pc1	**36.34 ± 2.13**	323.44 ± 63.73	353.82 ± 166.70	377.09 ± 201.18
banknote-aut...	**0.13 ± 0.07**	2.31 ± 0.80	2.50 ± 2.85	0.48 ± 0.14
pc4	**54.73 ± 5.25**	529.77 ± 118.43	380.41 ± 287.70	765.68 ± 109.57
cmc	**267.74 ± 5.26**	270.21 ± 4.98	269.76 ± 5.97	268.46 ± 6.22
pc3	**79.82 ± 5.85**	476.16 ± 109.85	378.89 ± 208.19	891.88 ± 756.57
car	19.51 ± 2.71	10.89 ± 4.46	**10.19 ± 3.50**	12.57 ± 9.80
steel-plates...	**213.17 ± 15.35**	5017.82 ± 313.20	4303.54 ± 2069.24	5320.40 ± 283.54
mfeat-zernik...	**141.02 ± 15.50**	423.68 ± 164.21	248.37 ± 59.40	312.44 ± 81.80
mfeat-karhun...	53.79 ± 8.70	**50.01 ± 15.92**	62.53 ± 25.83	52.45 ± 18.68
mfeat-morpho...	**254.25 ± 6.31**	3538.51 ± 1578.29	675.60 ± 191.55	1748.34 ± 755.62
mfeat-fourie...	**171.60 ± 16.09**	255.61 ± 65.48	221.58 ± 14.25	291.94 ± 114.54

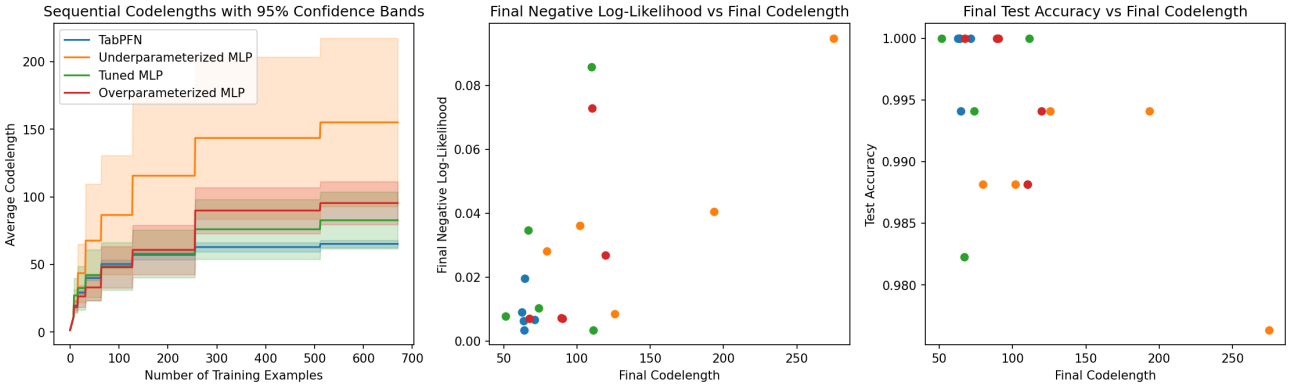
Table 4. Average classification test accuracy (Mean ± Std) over 5 different train-test splits.

	TabPFN	Underparameterized MLP	Tuned MLP	Overparameterized MLP
kc2	**0.85 ± 0.02**	0.22 ± 0.01	0.77 ± 0.06	0.72 ± 0.10
climate-mode...	**0.96 ± 0.01**	0.95 ± 0.01	0.94 ± 0.02	0.94 ± 0.01
wdbc	**0.98 ± 0.01**	0.93 ± 0.01	0.91 ± 0.03	0.92 ± 0.02
ilpd	**0.73 ± 0.02**	0.70 ± 0.04	0.70 ± 0.02	0.71 ± 0.01
balance-scal...	**0.99 ± 0.01**	0.98 ± 0.01	0.98 ± 0.02	0.98 ± 0.01
blood-transf...	**0.80 ± 0.03**	0.77 ± 0.01	0.73 ± 0.06	0.72 ± 0.08
diabetes	**0.74 ± 0.02**	0.67 ± 0.01	0.66 ± 0.02	0.66 ± 0.02
analcatdata-...	**1.00 ± 0.00**	0.99 ± 0.01	0.99 ± 0.01	1.00 ± 0.00
vehicle	**0.85 ± 0.02**	0.58 ± 0.08	0.66 ± 0.03	0.68 ± 0.04
tic-tac-toe	**0.93 ± 0.01**	0.90 ± 0.02	0.90 ± 0.02	0.90 ± 0.03
credit-g	**0.76 ± 0.02**	0.61 ± 0.14	0.64 ± 0.12	0.57 ± 0.11
qsar-biodeg	**0.87 ± 0.02**	0.85 ± 0.03	0.85 ± 0.02	0.86 ± 0.01
pc1	**0.93 ± 0.00**	0.89 ± 0.04	0.91 ± 0.03	0.87 ± 0.11
banknote-aut...	**1.00 ± 0.00**	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
pc4	**0.92 ± 0.01**	0.85 ± 0.03	0.80 ± 0.04	0.82 ± 0.11
cmc	**0.55 ± 0.02**	0.54 ± 0.01	0.55 ± 0.02	0.55 ± 0.01
pc3	**0.90 ± 0.01**	0.86 ± 0.03	0.89 ± 0.01	0.69 ± 0.17
car	0.98 ± 0.01	0.99 ± 0.01	**0.99 ± 0.00**	0.99 ± 0.01
steel-plates...	**0.78 ± 0.01**	0.44 ± 0.03	0.37 ± 0.08	0.40 ± 0.03
mfeat-zernik...	**0.83 ± 0.01**	0.80 ± 0.01	0.81 ± 0.01	0.79 ± 0.01
mfeat-karhun...	0.97 ± 0.01	**0.97 ± 0.01**	0.97 ± 0.01	0.97 ± 0.01
mfeat-morpho...	**0.74 ± 0.01**	0.25 ± 0.08	0.34 ± 0.19	0.18 ± 0.06
mfeat-fourie...	0.83 ± 0.01	**0.84 ± 0.01**	0.83 ± 0.01	0.82 ± 0.01

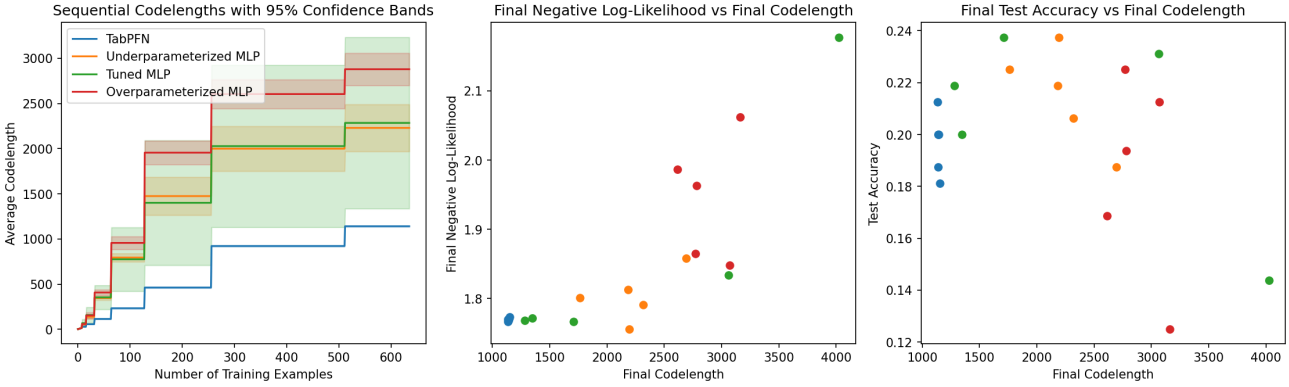
C. Additional results that complement Figure 1

Here we show the same result as in Figure 1 for all 24 OpenML datasets we considered.

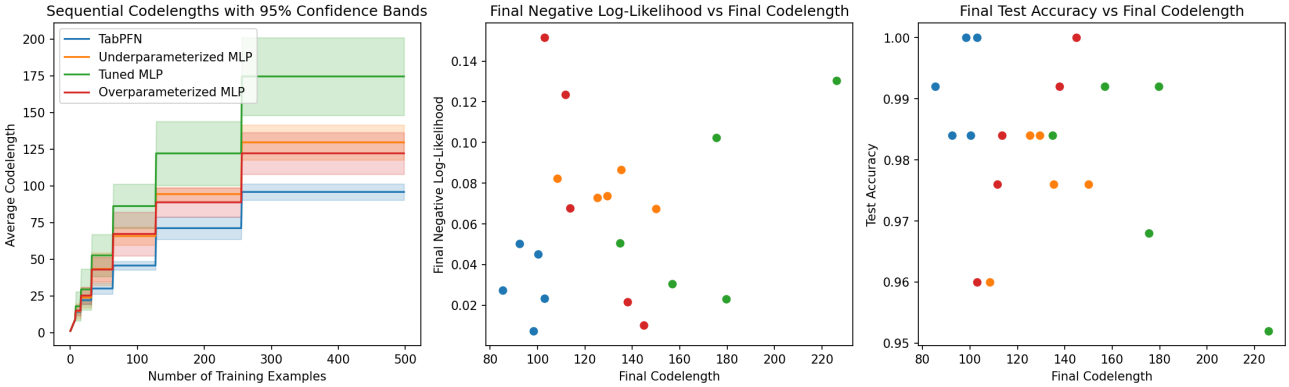
Dataset: analcatdata_authorship



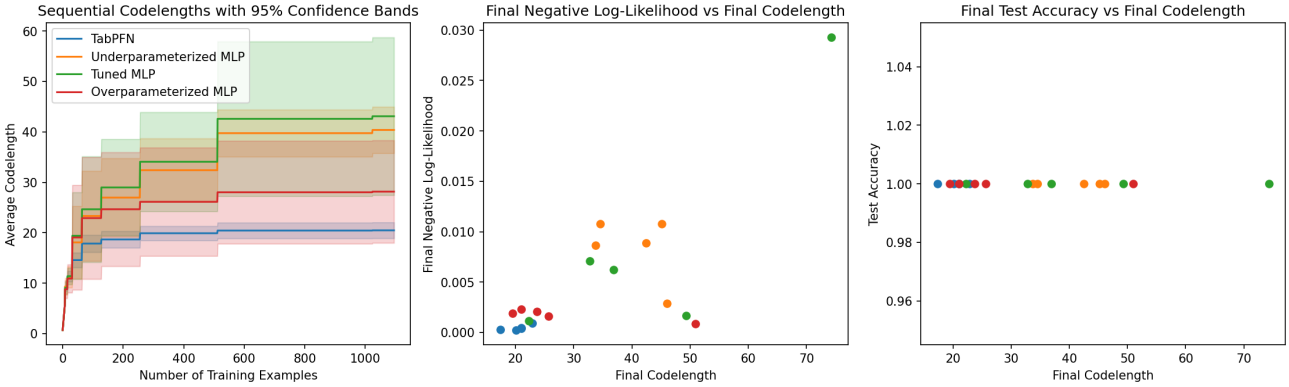
Dataset: analcatdata_dmft



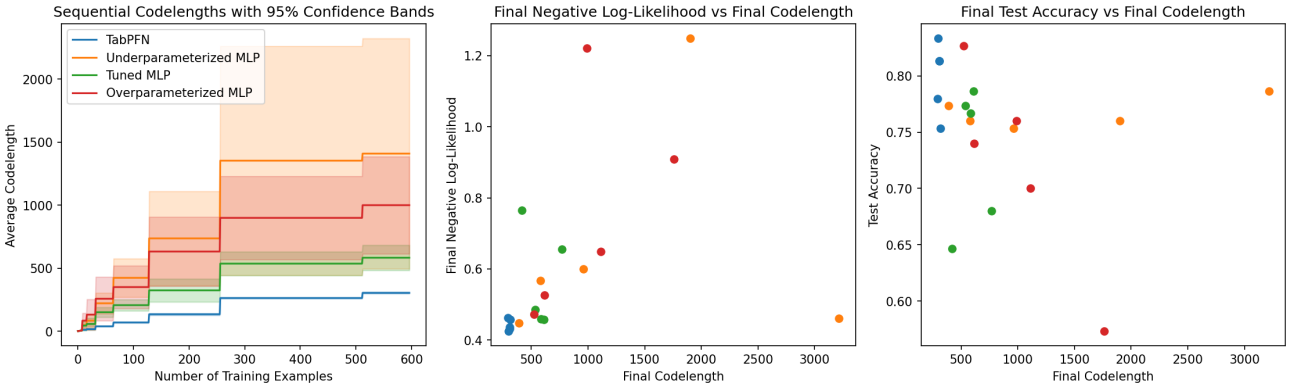
Dataset: balance-scale



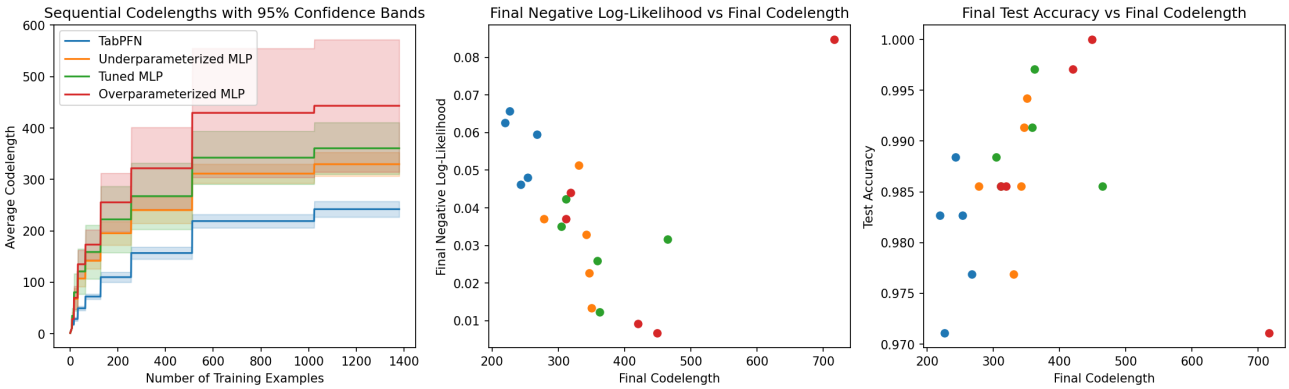
Dataset: banknote-authentication



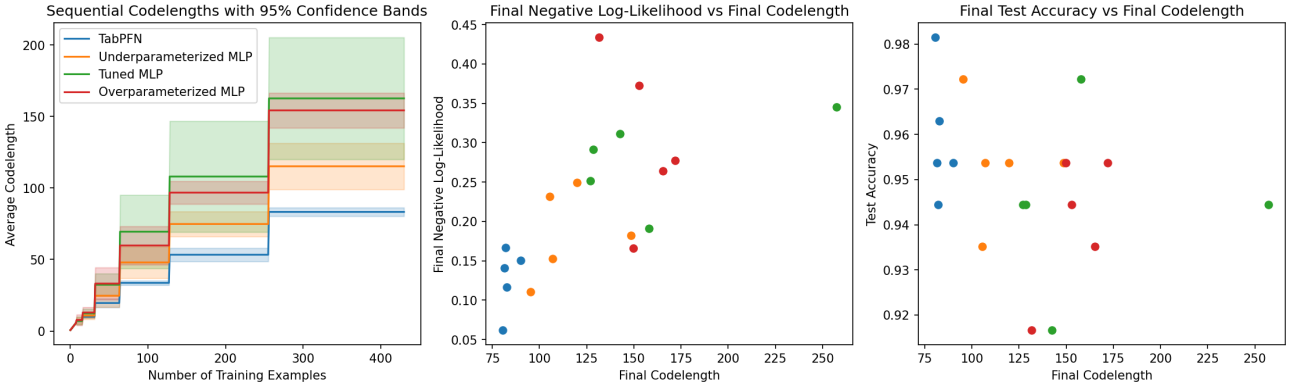
Dataset: blood-transfusion-service-center



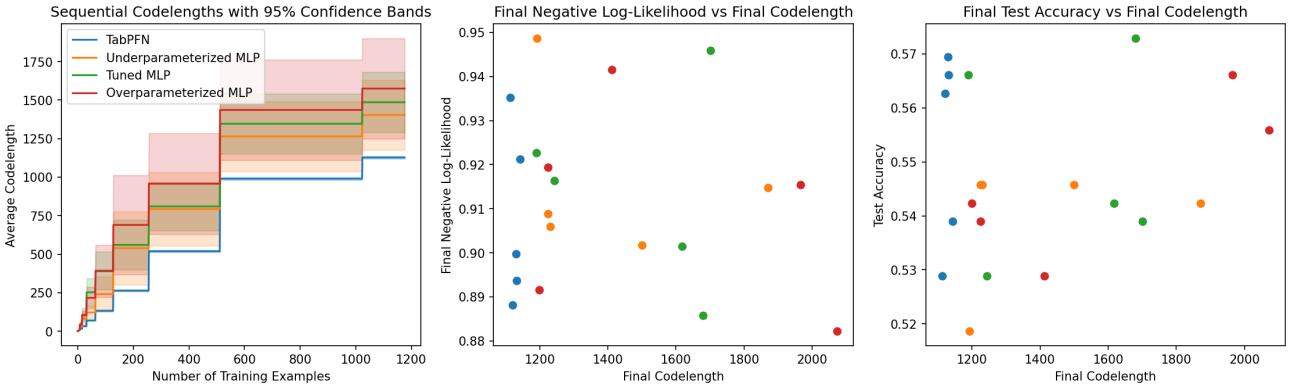
Dataset: car



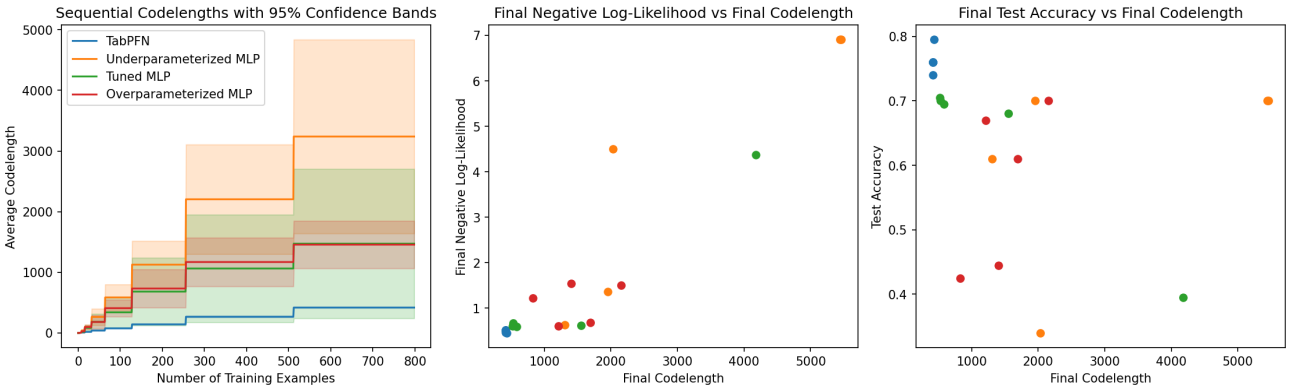
Dataset: climate-model-simulation-crashes



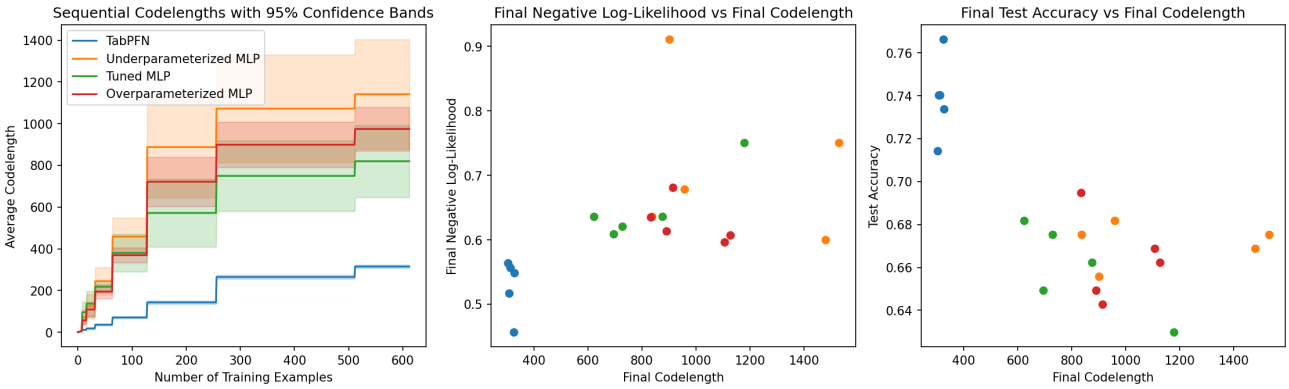
Dataset: cmc



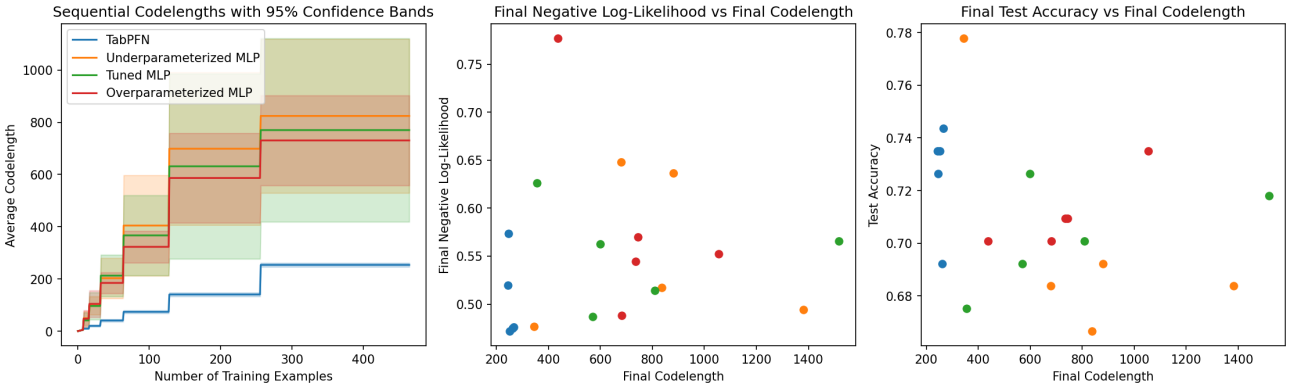
Dataset: credit-g



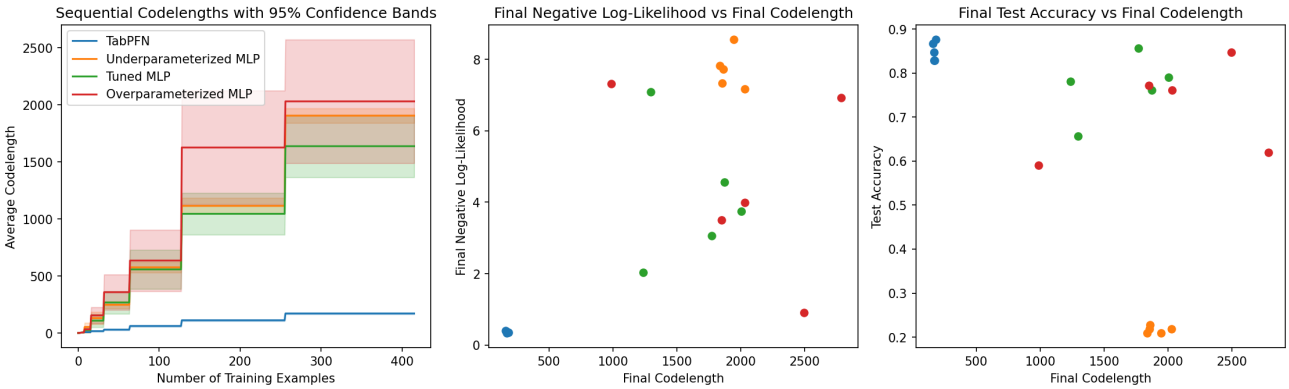
Dataset: diabetes



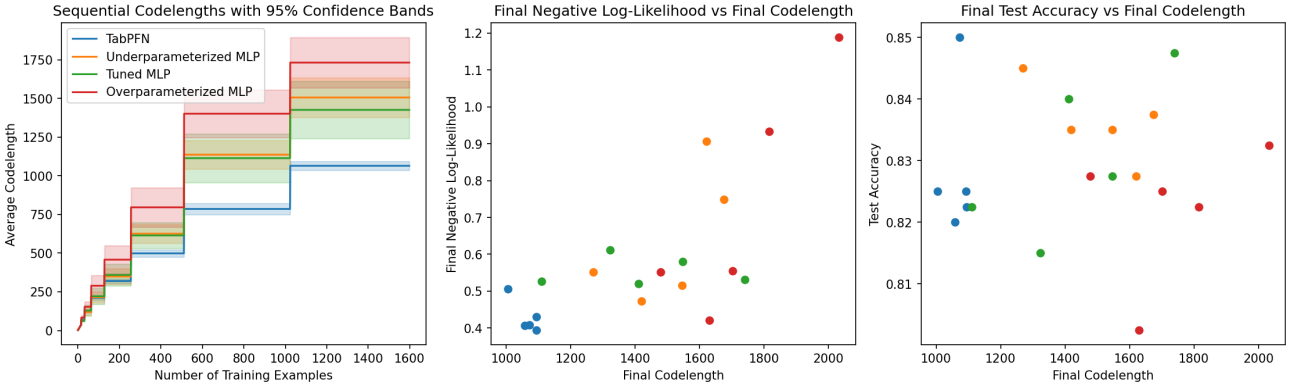
Dataset: ilpd



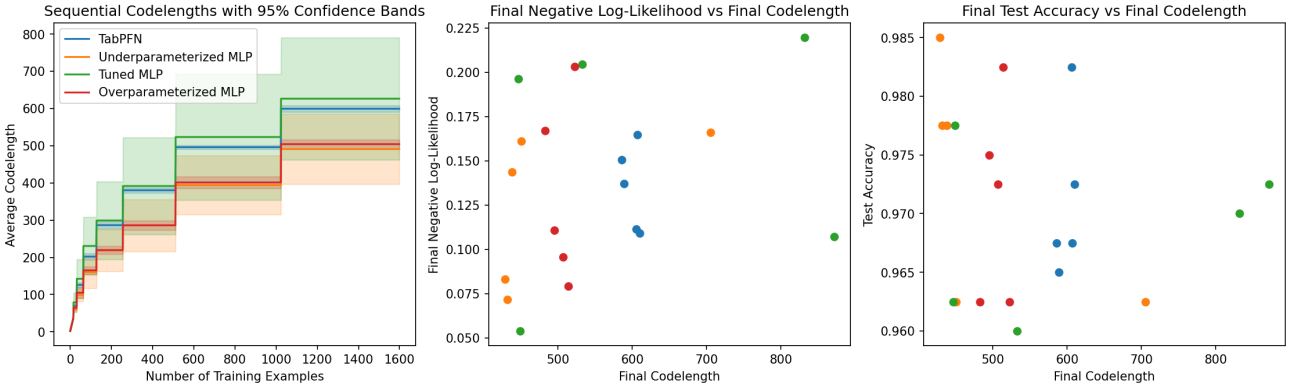
Dataset: kc2



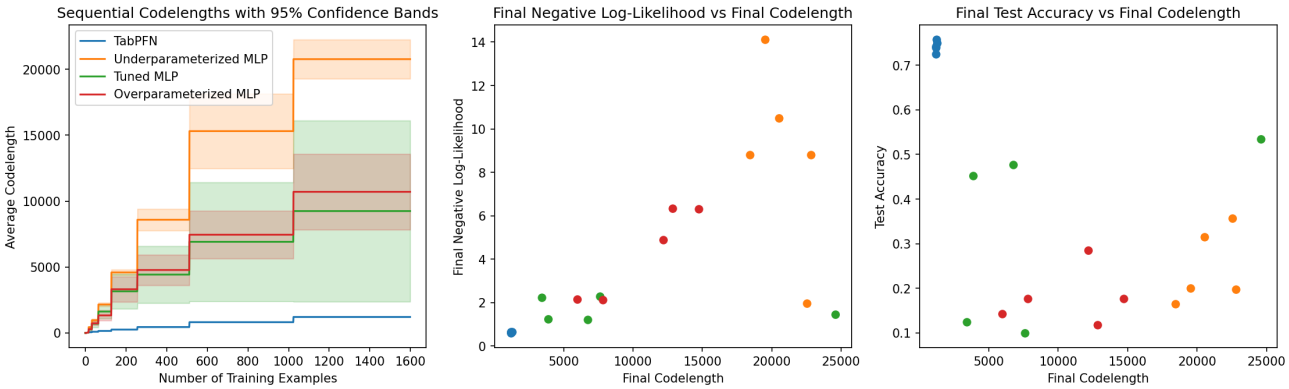
Dataset: mfeat-fourier



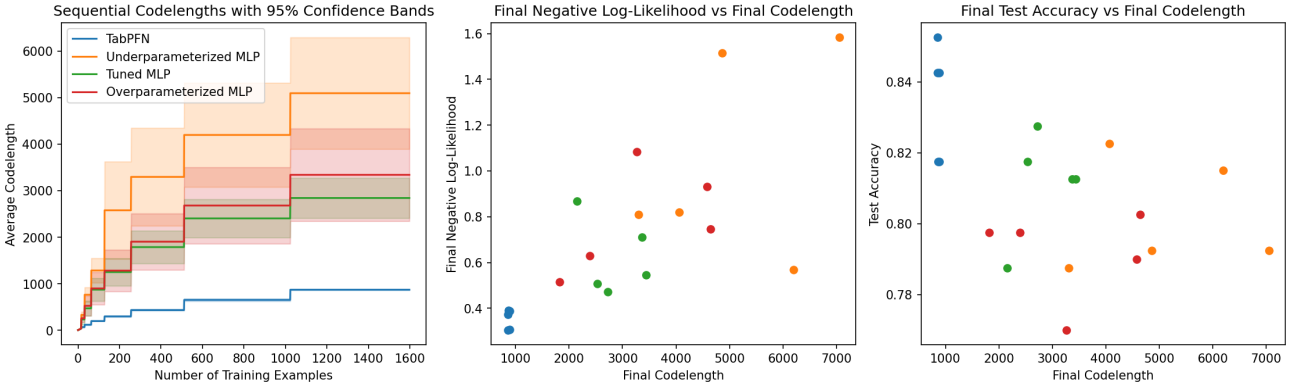
Dataset: mfeat-karhunen



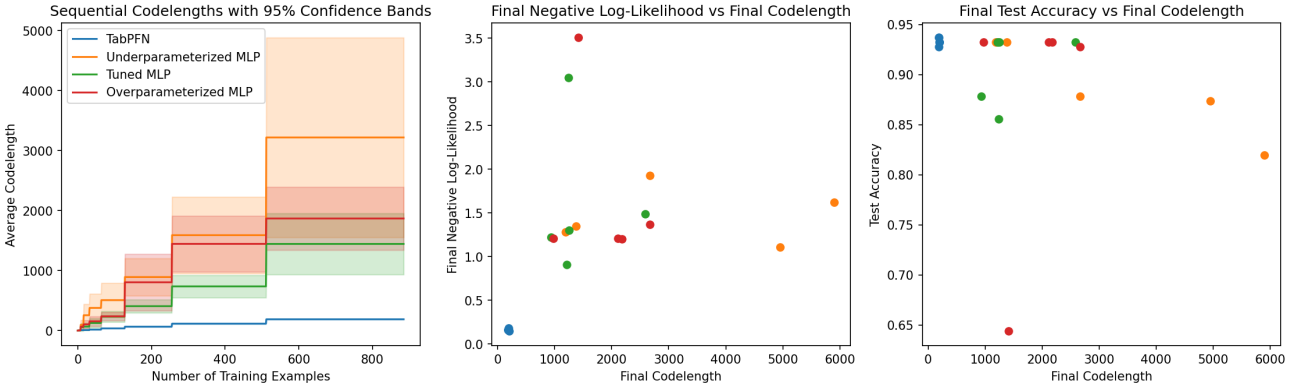
Dataset: mfeat-morphological



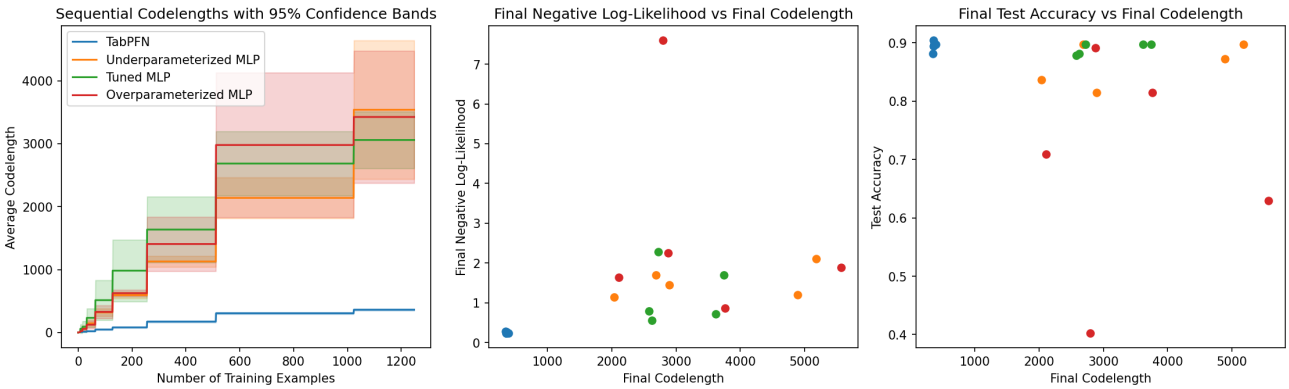
Dataset: mfeat-zernike



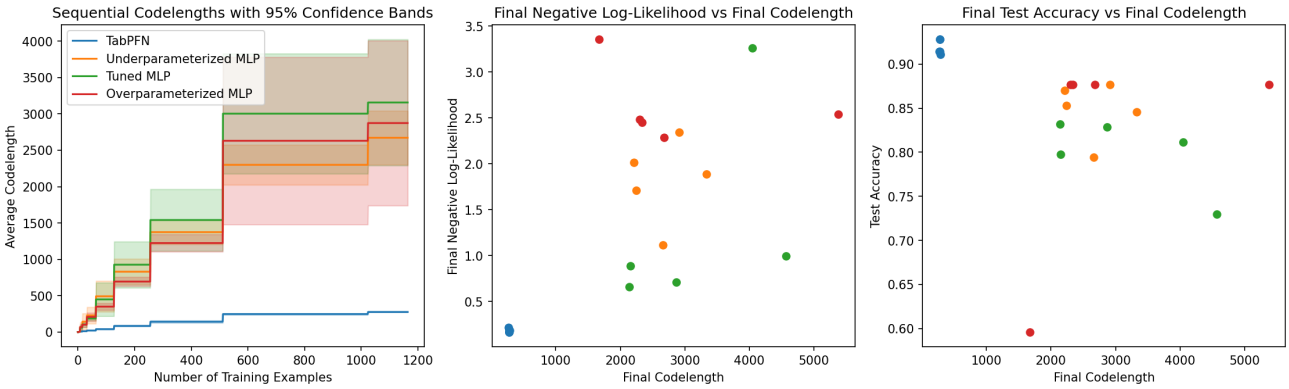
Dataset: pc1



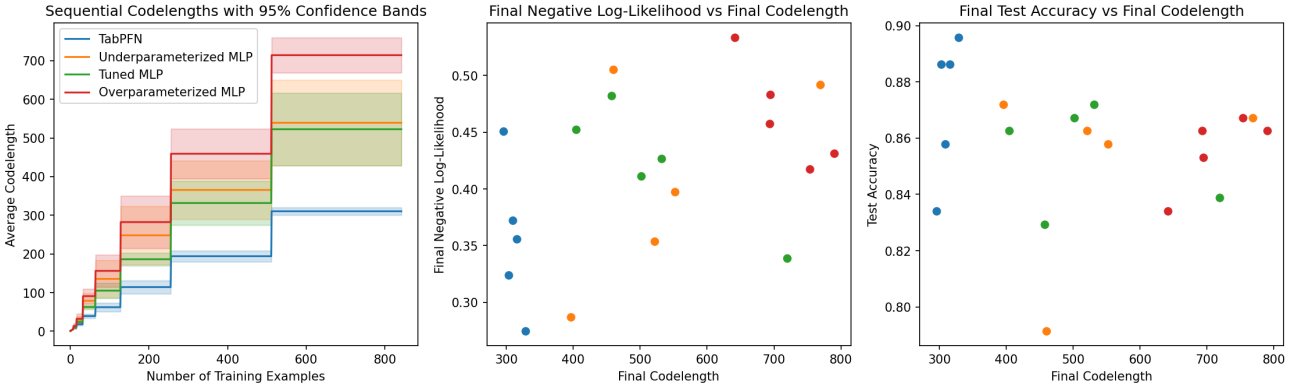
Dataset: pc3



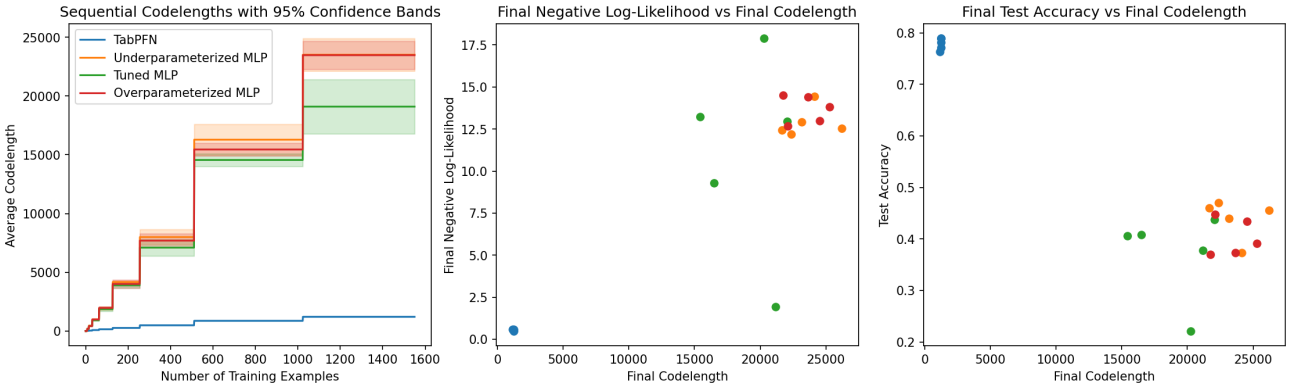
Dataset: pc4



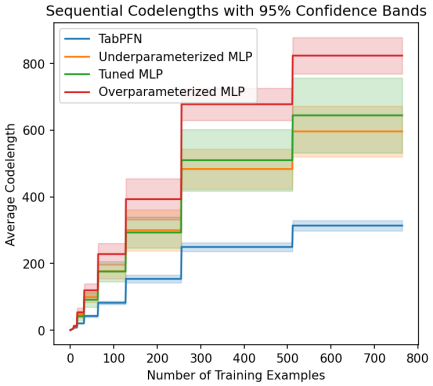
Dataset: qsar-biodeg

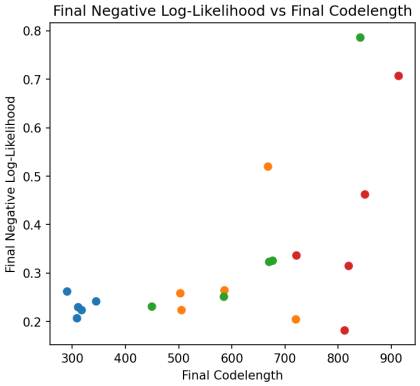


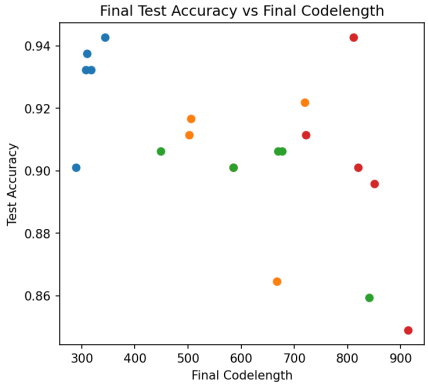
Dataset: steel-plates-fault



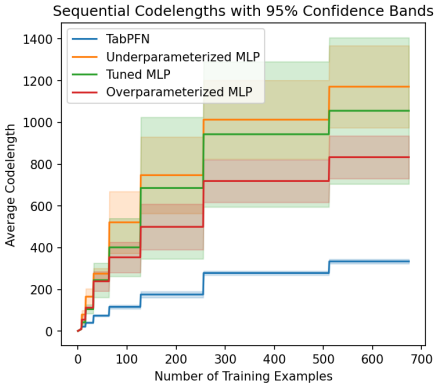
Dataset: tic-tac-toe

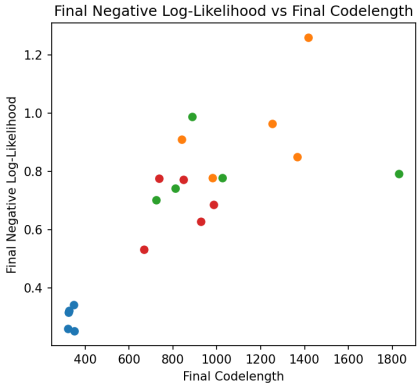


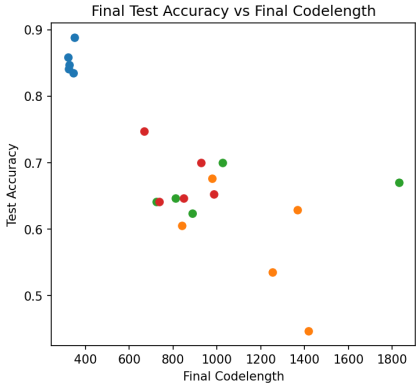




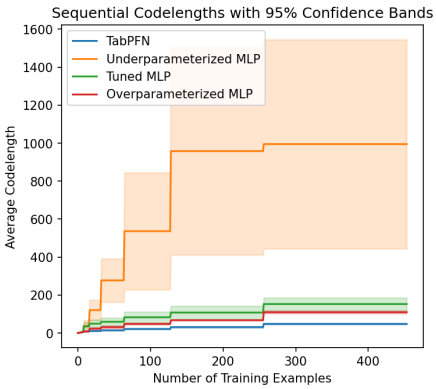
Dataset: vehicle

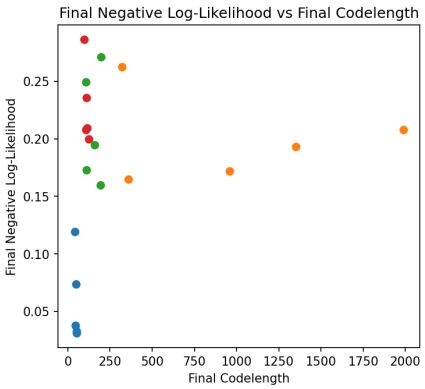


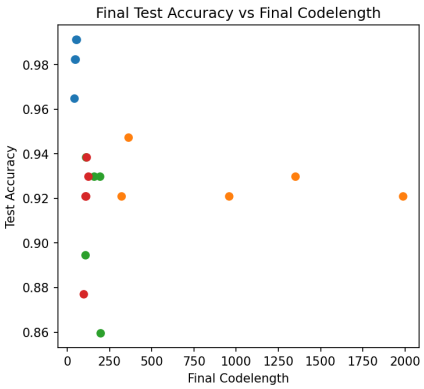




Dataset: wdbc







D. Tracking NLL as context length grows

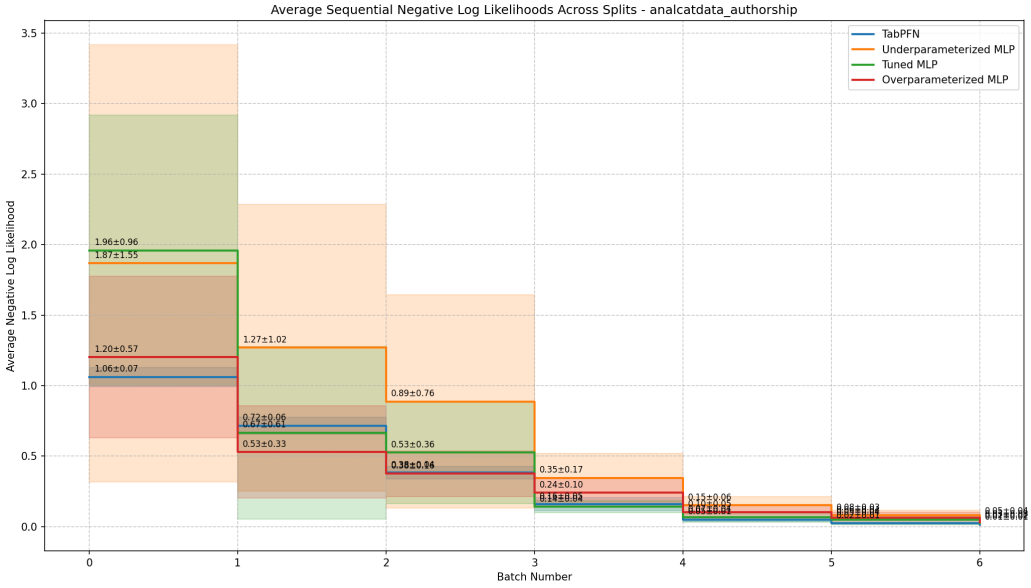
As we perform our training in buckets, it is interesting to look at the rate in which our models learn. A hypothesis is that TabPFN 'in-context learns' a PPD for the data early on, and doesn't vary its choice of model much over more training examples. To test this, we compute the NLL of the models after training on each bucket. After training on the new examples from the new bucket, we compute the NLL on the model in the prequential SGD plug-in case:

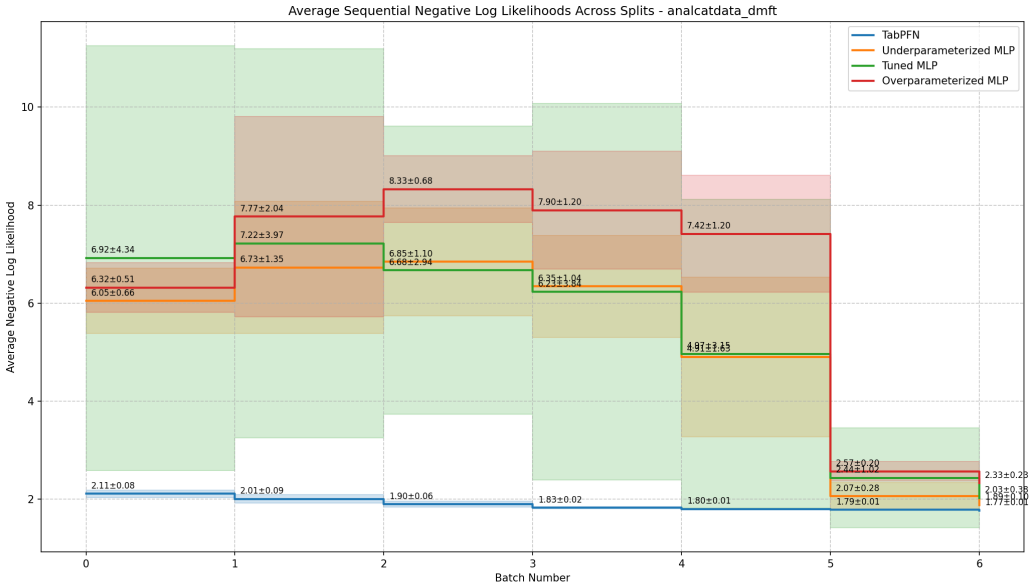
$$\text{NLL}_s = -\frac{1}{m}\sum_{i=1}^{m} \log p_{\hat{\theta}_{\left(x^{\text{train}}_{1:t_s}, y^{\text{train}}_{1:t_s}\right)}}(y_i^{\text{test}}|x_i^{\text{test}}), \quad s = 1, \ldots, S$$

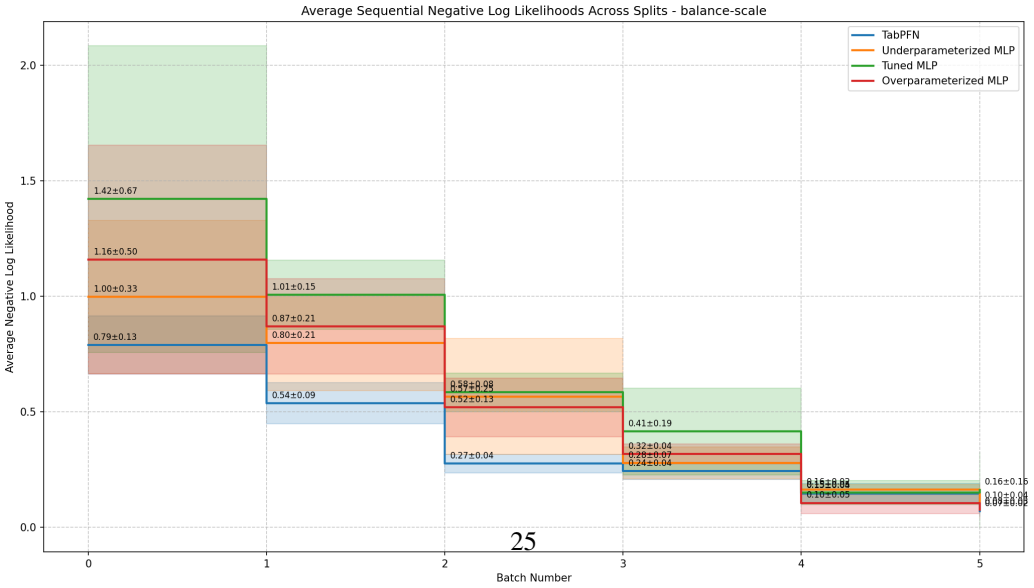
and the TabPFN case:

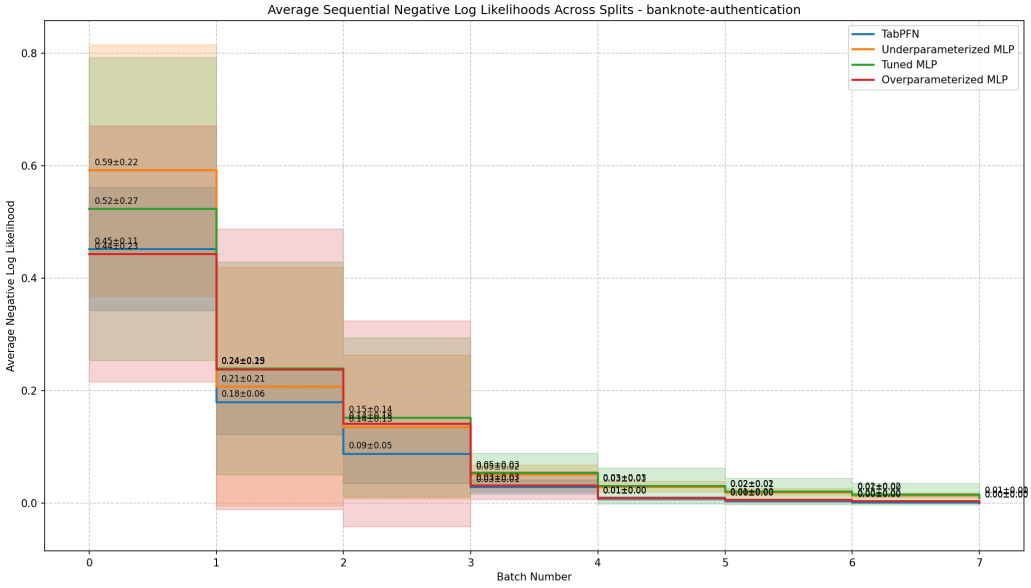
$$\text{NLL}_s = -\frac{1}{m}\sum_{i=1}^{m} \log T_{\hat{\phi}}(y_i^{\text{test}}|x_i^{\text{test}}, x^{\text{train}}_{1:t_s}, y^{\text{train}}_{1:t_s}), \quad s = 1, \ldots, S$$

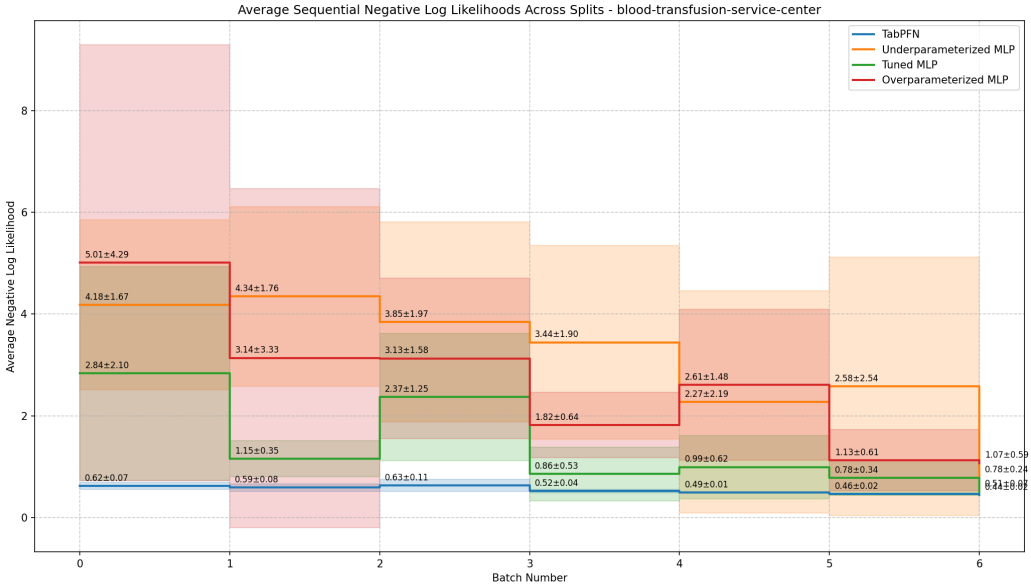
We plot the sequential NLL of TabPFN and the MLPs for each dataset. In many cases, we see that the sequential NLL stabilises very quickly for TabPFN, indicating TabPFN was able to choose a model (i.e a PPD) for the data with little in-context examples. This is unlike the MLPs, which require many more training examples to achieve a low NLL.

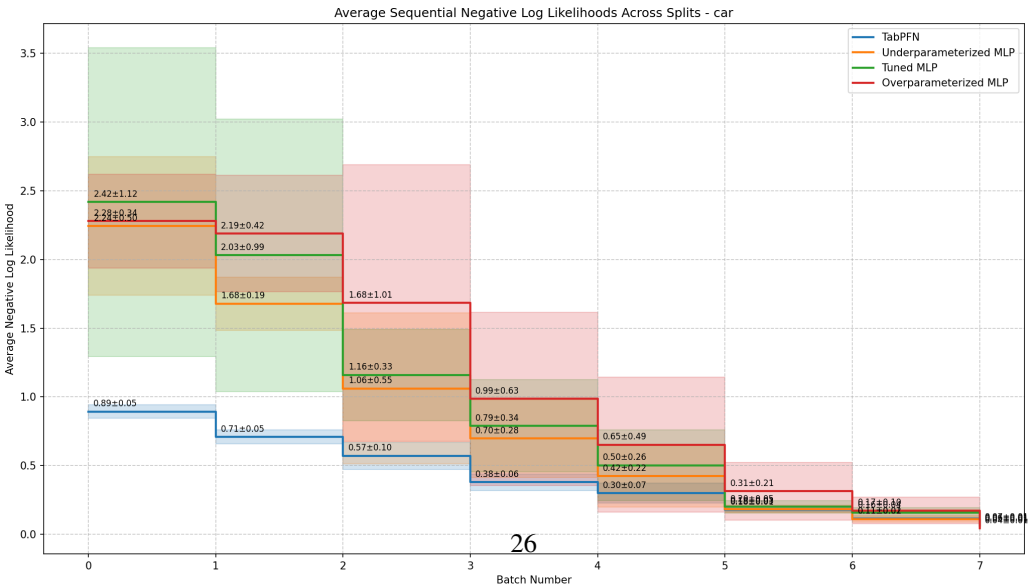

Average Sequential Negative Log Likelihoods Across Splits - analcatdata_authorship
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
1.96±0.96
1.87±1.55
1.20±0.57
1.06±0.07
1.27±1.02
0.72±0.06
0.67±0.61
0.53±0.33
0.89±0.76
0.53±0.36
0.35±0.17
0.24±0.10

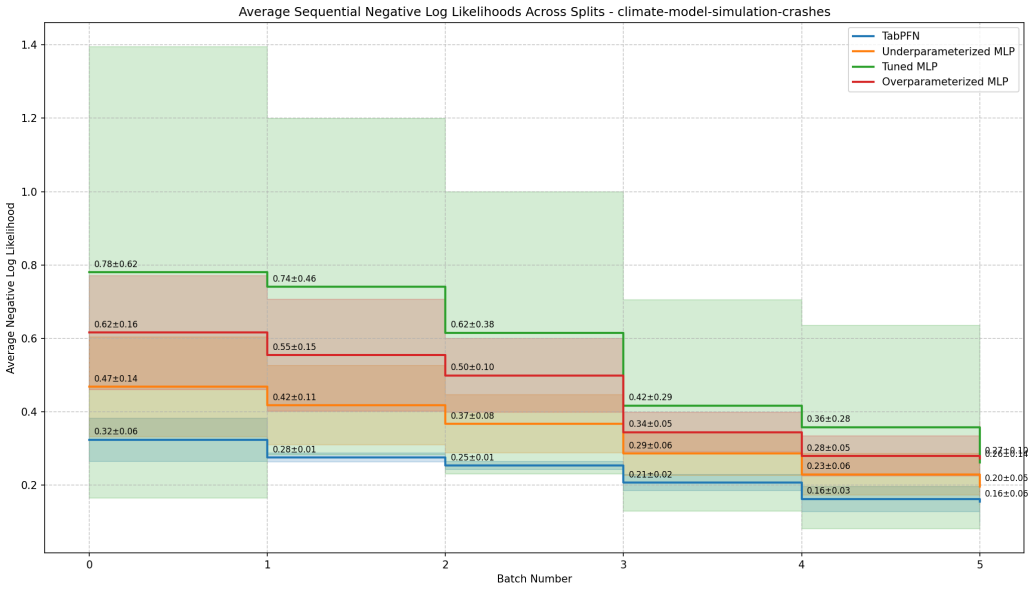

Average Sequential Negative Log Likelihoods Across Splits - analcatdata_dmft
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
6.92±4.34
6.32±0.51
6.05±0.66
7.77±2.04
7.22±3.97
6.73±1.35
8.33±0.68
6.85±1.10
6.68±2.94
7.90±1.20
7.42±1.20
2.11±0.08
2.01±0.09
1.90±0.06
1.83±0.02
1.80±0.01
2.07±0.28
1.79±0.01
2.33±0.23
2.03±0.35
1.77±0.01

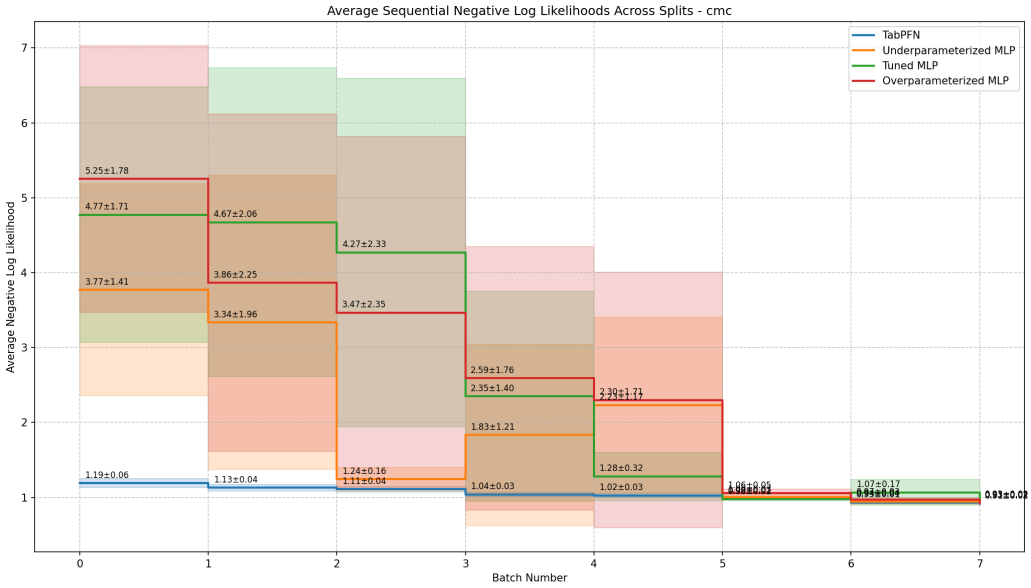

Average Sequential Negative Log Likelihoods Across Splits - balance-scale
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
1.42±0.67
1.16±0.50
1.00±0.33
0.79±0.13
1.01±0.15
0.87±0.21
0.80±0.21
0.54±0.09
0.52±0.13
0.27±0.04
0.41±0.19
0.32±0.04
0.24±0.04
0.16±0.16

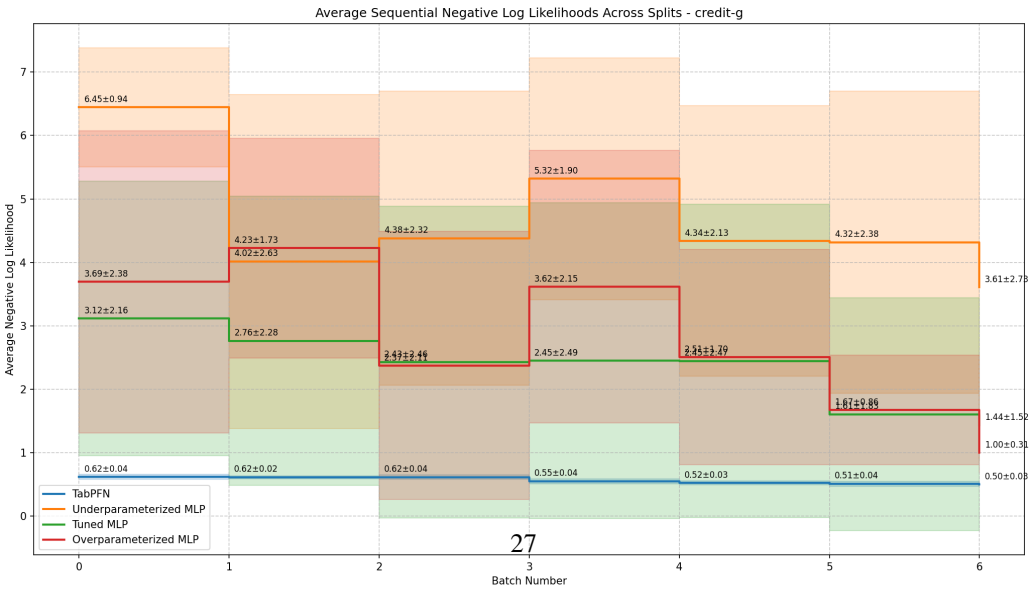

Average Sequential Negative Log Likelihoods Across Splits - banknote-authentication
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
0.59±0.22
0.52±0.27
0.21±0.21
0.18±0.06
0.09±0.05

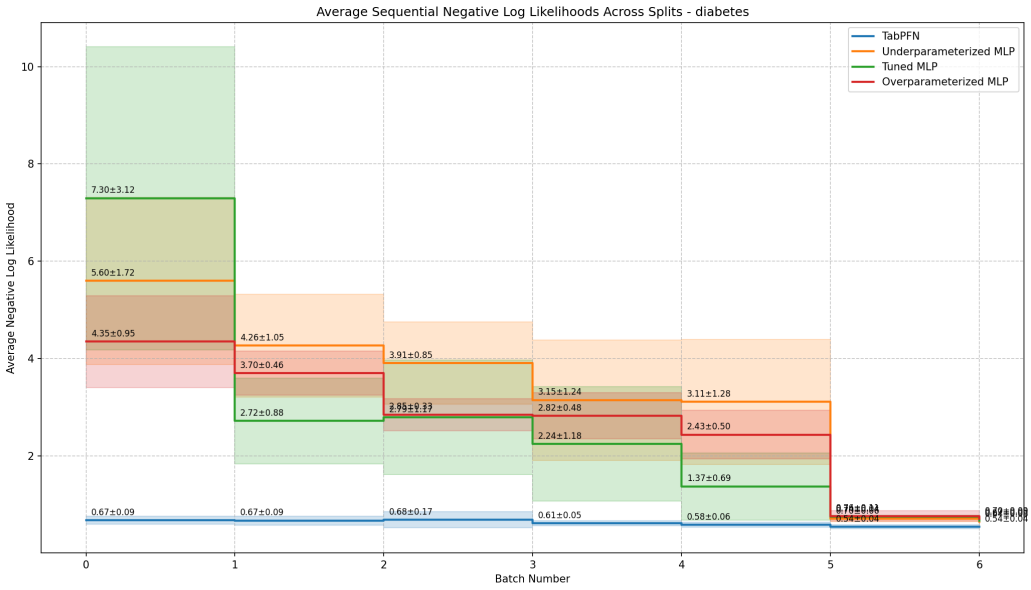

Average Sequential Negative Log Likelihoods Across Splits - blood-transfusion-service-center
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
5.01±4.29
4.18±1.67
4.34±1.76
3.14±3.33
3.85±1.97
3.13±1.58
3.44±1.90
2.84±2.10
1.15±0.35
2.37±1.25
1.82±0.64
2.61±1.48
2.27±2.19
2.58±2.54
1.13±0.61
0.62±0.07
0.59±0.08
0.63±0.11
0.86±0.53
0.52±0.04
0.99±0.62
0.49±0.01
0.78±0.34
0.46±0.02

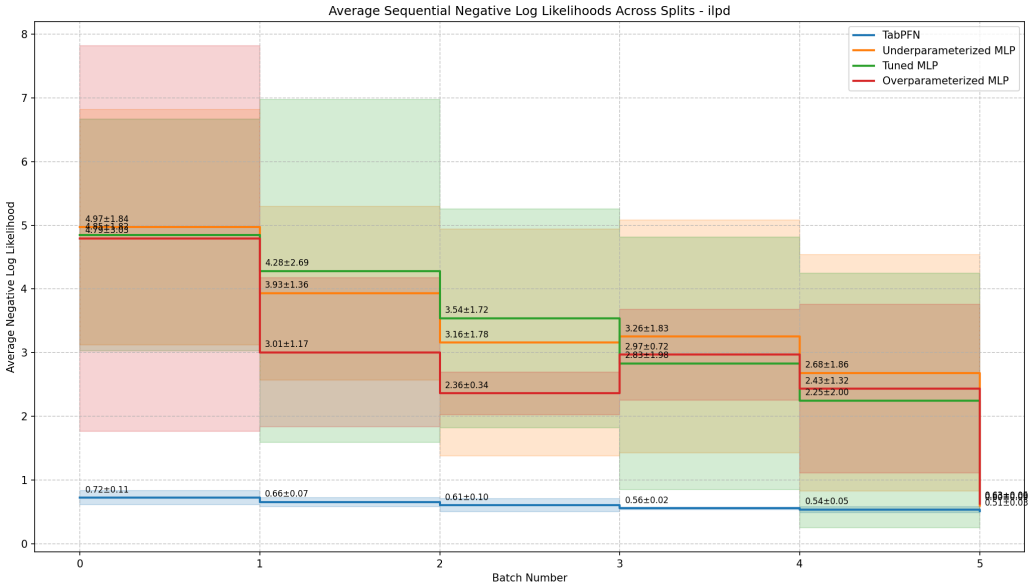

Average Sequential Negative Log Likelihoods Across Splits - car
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
2.42±1.12
2.19±0.42
2.03±0.99
1.68±0.19
1.68±1.01
1.16±0.33
1.06±0.55
0.89±0.05
0.71±0.05
0.99±0.63
0.79±0.34
0.70±0.28
0.57±0.10
0.65±0.49
0.50±0.26
0.42±0.22
0.38±0.06
0.30±0.07
0.31±0.21

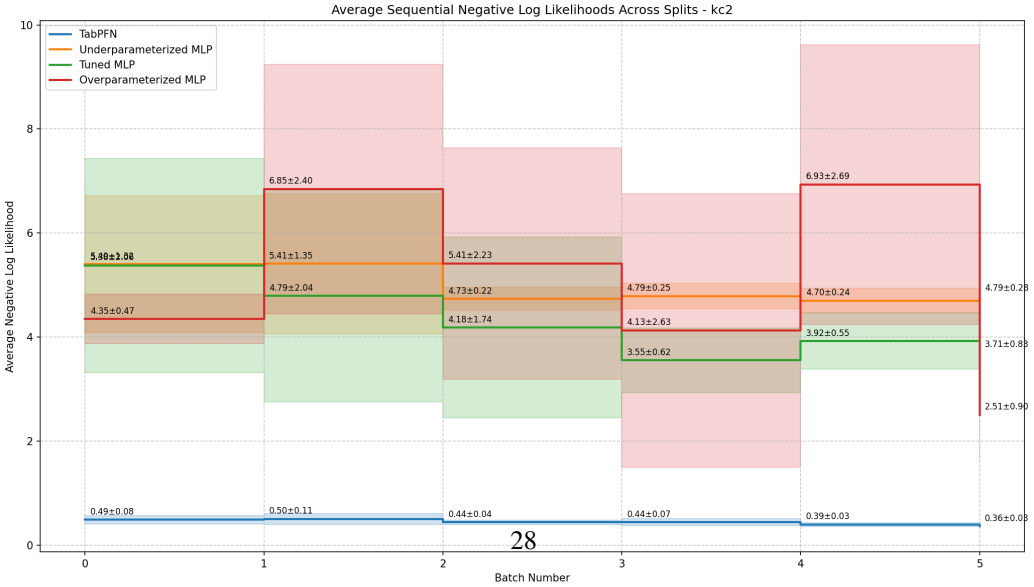

Average Sequential Negative Log Likelihoods Across Splits - climate-model-simulation-crashes
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
0.78±0.62
0.74±0.46
0.62±0.38
0.42±0.29
0.36±0.28
0.62±0.16
0.55±0.15
0.50±0.10
0.34±0.05
0.28±0.05
0.47±0.14
0.42±0.11
0.37±0.08
0.29±0.06
0.23±0.06
0.32±0.06
0.28±0.01
0.25±0.01
0.21±0.02
0.16±0.03
0.20±0.05
0.16±0.05

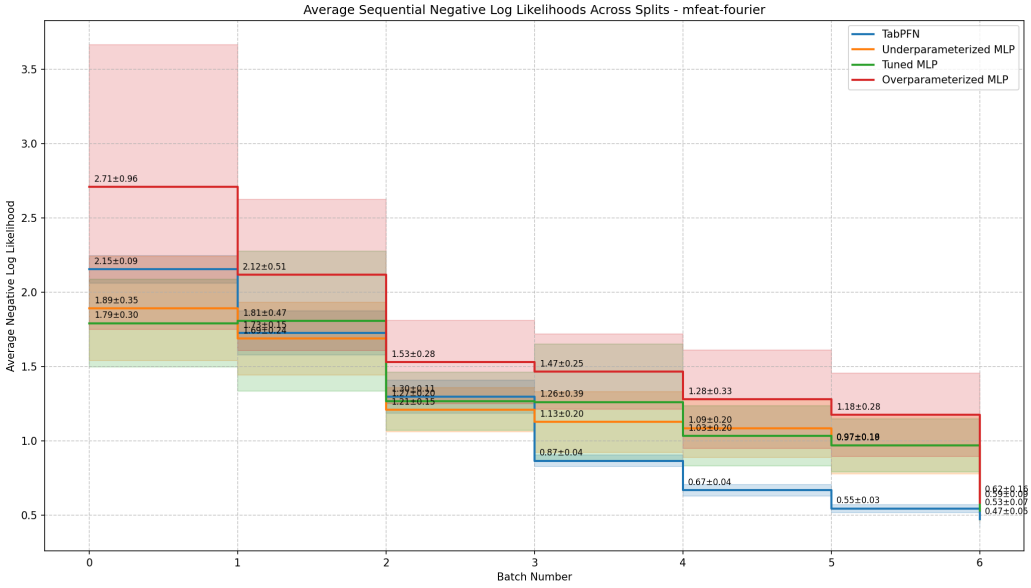

Average Sequential Negative Log Likelihoods Across Splits - cmc
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
5.25±1.78
4.77±1.71
4.67±2.06
4.27±2.33
3.77±1.41
3.86±2.25
3.47±2.35
3.34±1.96
2.59±1.76
2.35±1.40
1.83±1.21
1.19±0.06
1.13±0.04
1.24±0.16
1.11±0.04
1.04±0.03
1.28±0.32
1.02±0.03
1.07±0.17

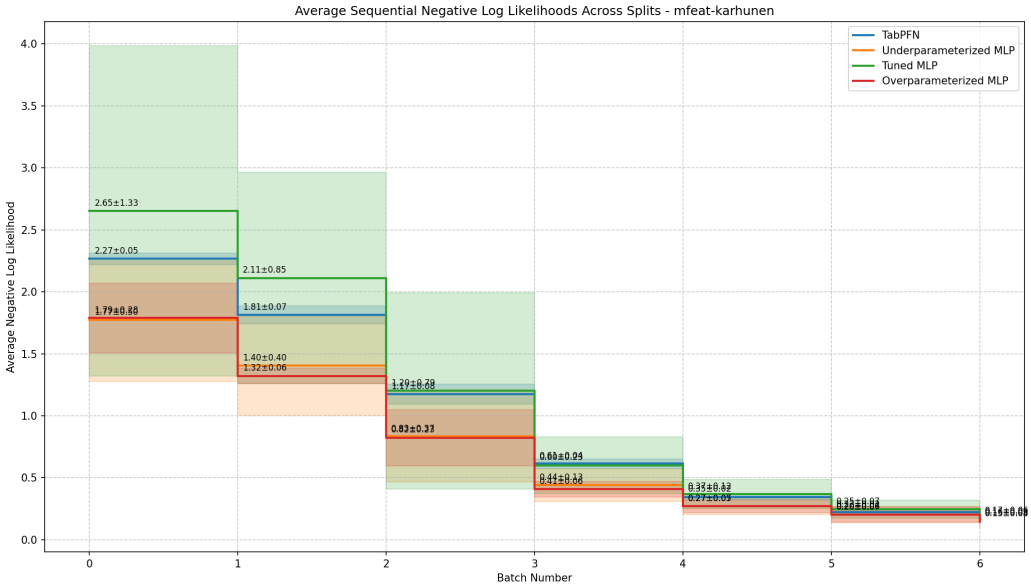

Average Sequential Negative Log Likelihoods Across Splits - credit-g
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
6.45±0.94
5.32±1.90
4.38±2.32
4.34±2.13
4.32±2.38
4.23±1.73
4.02±2.63
3.69±2.38
3.62±2.15
3.61±2.73
3.12±2.16
2.76±2.28
2.45±2.49
1.44±1.52
1.00±0.31
0.62±0.04
0.62±0.02
0.62±0.04
0.55±0.04
0.52±0.03
0.51±0.04
0.50±0.03

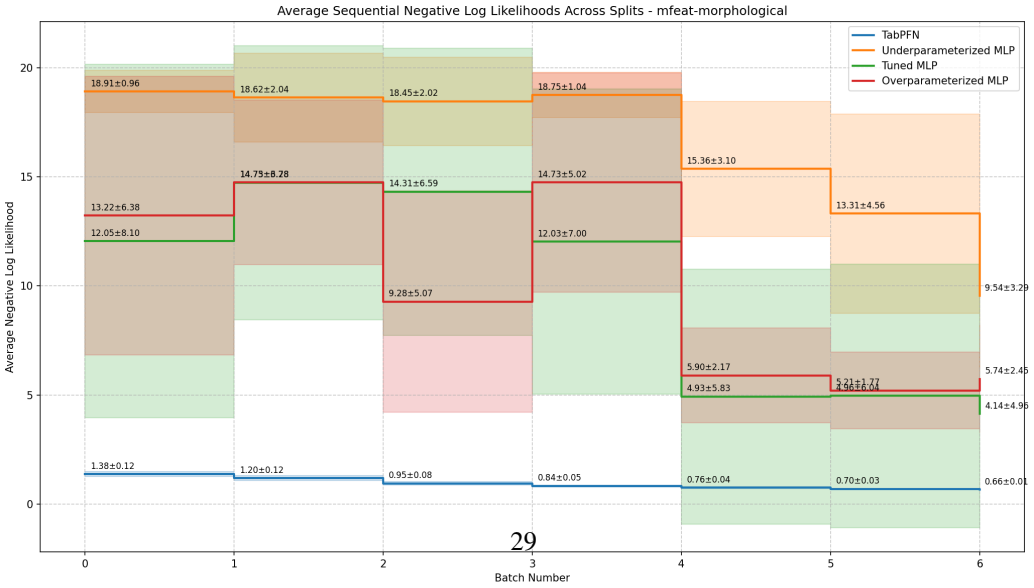

Average Sequential Negative Log Likelihoods Across Splits - diabetes
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
7.30±3.12
5.60±1.72
4.35±0.95
4.26±1.05
3.70±0.46
3.91±0.85
2.72±0.88
3.15±1.24
2.82±0.48
3.11±1.28
2.43±0.50
2.24±1.18
1.37±0.69
0.67±0.09
0.67±0.09
0.68±0.17
0.61±0.05
0.58±0.06

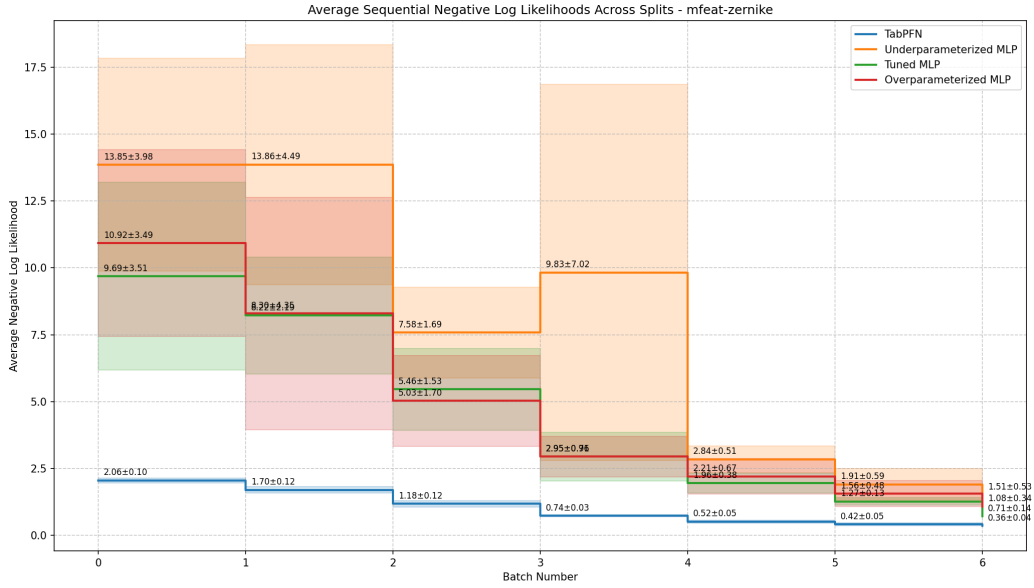

Average Sequential Negative Log Likelihoods Across Splits - ilpd
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
4.28±2.69
3.93±1.36
3.01±1.17
3.54±1.72
3.16±1.78
2.36±0.34
3.26±1.83
2.97±0.72
2.68±1.86
2.43±1.32
2.25±2.00
0.72±0.11
0.66±0.07
0.61±0.10
0.56±0.02
0.54±0.05

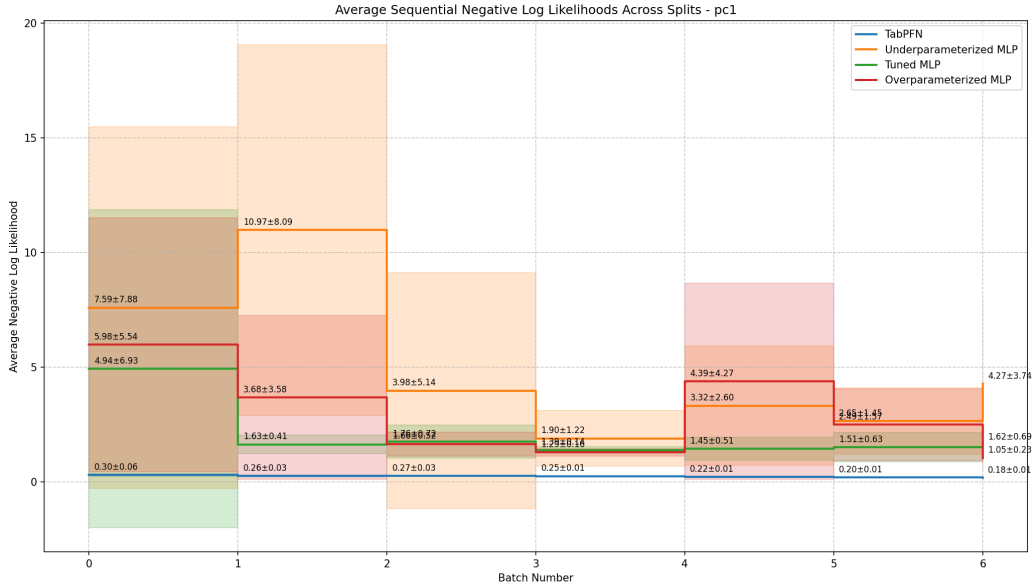

Average Sequential Negative Log Likelihoods Across Splits - kc2
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
6.85±2.40
5.41±1.35
4.79±2.04
4.35±0.47
5.41±2.23
4.73±0.22
4.18±1.74
4.79±0.25
4.13±2.63
3.55±0.62
6.93±2.69
4.70±0.24
3.92±0.55
4.79±0.28
3.71±0.83
2.51±0.90
0.49±0.08
0.50±0.11
0.44±0.04
0.44±0.07
0.39±0.03
0.36±0.03

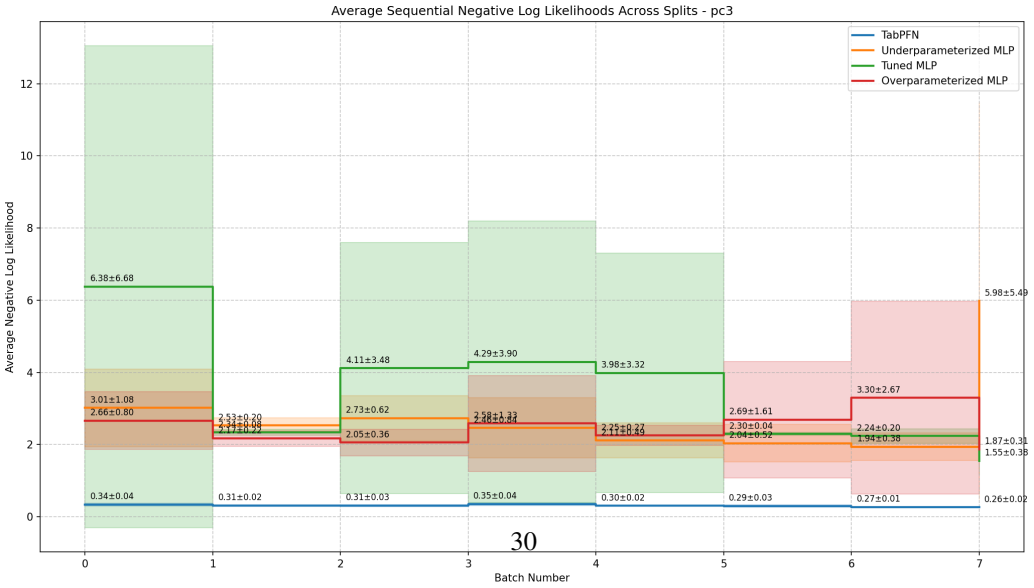

Average Sequential Negative Log Likelihoods Across Splits - mfeat-fourier
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
2.71±0.96
2.15±0.09
2.12±0.51
1.89±0.35
1.79±0.30
1.81±0.47
1.53±0.28
1.47±0.25
1.26±0.39
1.28±0.33
1.13±0.20
1.09±0.20
1.18±0.28
0.87±0.04
0.67±0.04
0.55±0.03
0.47±0.05

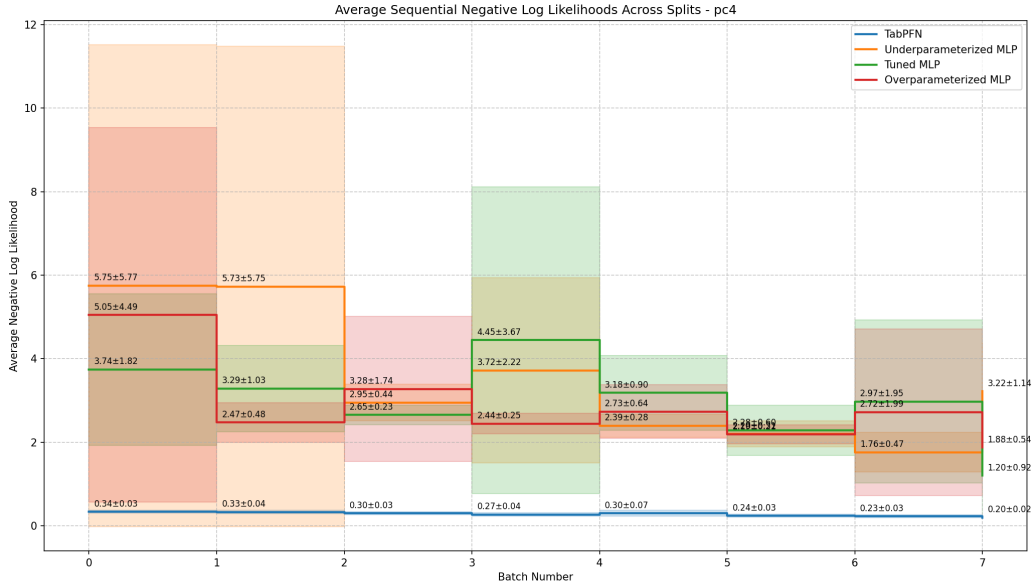

Average Sequential Negative Log Likelihoods Across Splits - mfeat-karhunen
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
2.65±1.33
2.27±0.05
2.11±0.85
1.81±0.07
1.40±0.40
1.32±0.06

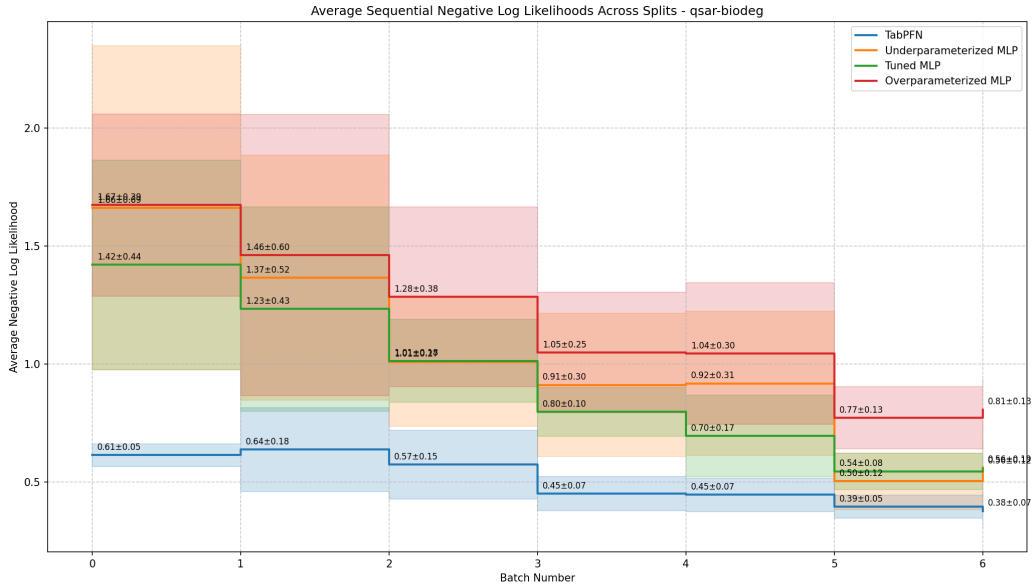

Average Sequential Negative Log Likelihoods Across Splits - mfeat-morphological
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
18.91±0.96
18.62±2.04
18.45±2.02
18.75±1.04
15.36±3.10
13.31±4.56
9.54±3.29
13.22±6.38
12.05±8.10
14.31±6.59
14.73±5.02
12.03±7.00
9.28±5.07
5.90±2.17
5.74±2.45
4.93±5.83
4.14±4.96
1.38±0.12
1.20±0.12
0.95±0.08
0.84±0.05
0.76±0.04
0.70±0.03
0.66±0.01

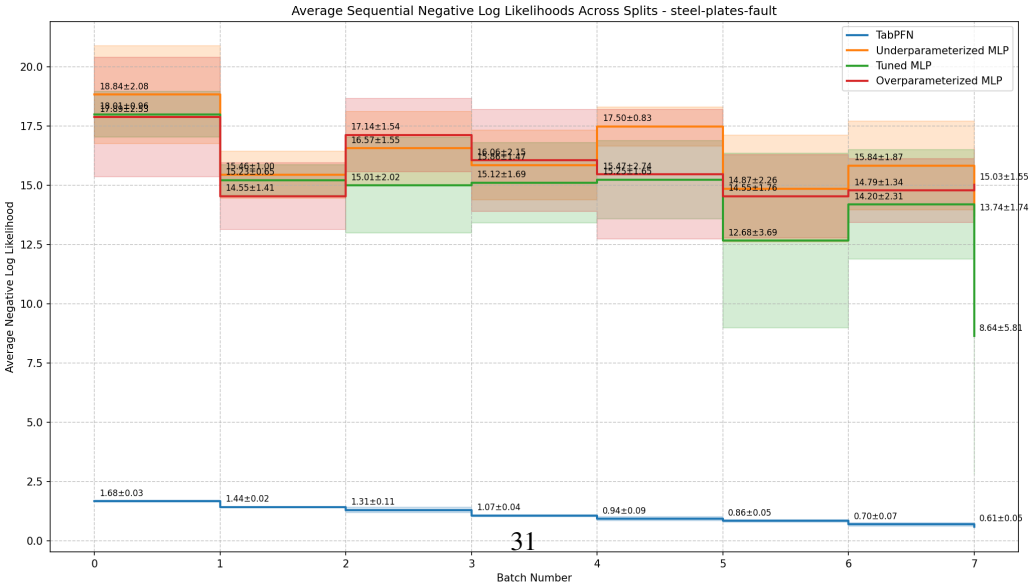

Average Sequential Negative Log Likelihoods Across Splits - mfeat-zernike
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
13.85±3.98
13.86±4.49
10.92±3.49
9.69±3.51
7.58±1.69
9.83±7.02
5.46±1.53
5.03±1.70
2.84±0.51
2.06±0.10
1.70±0.12
1.18±0.12
0.74±0.03
0.52±0.05
0.42±0.05

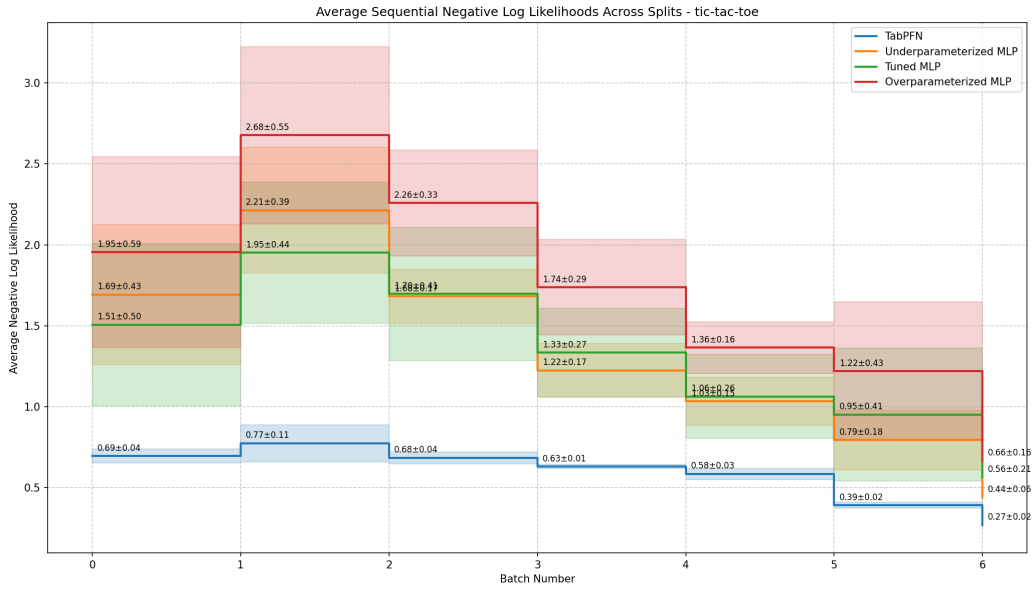

Average Sequential Negative Log Likelihoods Across Splits - pc1
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
10.97±8.09
7.59±7.88
5.98±5.54
4.94±6.93
3.98±5.14
3.68±3.58
4.39±4.27
3.32±2.60
4.27±3.74
1.90±1.22
1.63±0.41
1.45±0.51
1.51±0.63
1.62±0.69
0.30±0.06
0.26±0.03
0.27±0.03
0.25±0.01
0.22±0.01
0.20±0.01
0.18±0.01

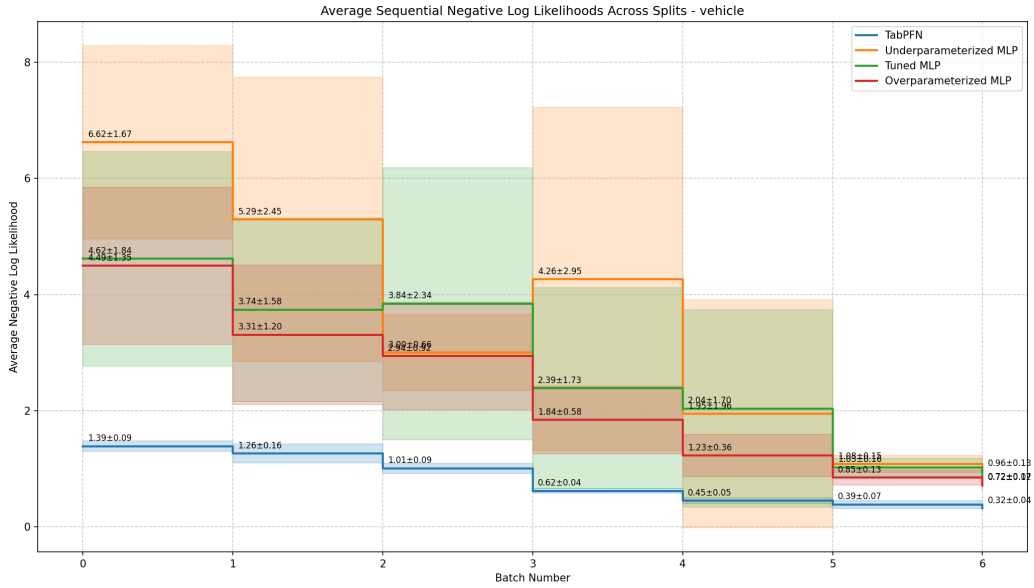

Average Sequential Negative Log Likelihoods Across Splits - pc3
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
6.38±6.68
5.98±5.49
4.11±3.48
4.29±3.90
3.98±3.32
3.30±2.67
3.01±1.08
2.66±0.80
2.73±0.62
2.69±1.61
2.24±0.20
2.05±0.36
1.87±0.31
0.34±0.04
0.31±0.02
0.31±0.03
0.35±0.04
0.30±0.02
0.29±0.03
0.27±0.01
0.26±0.02

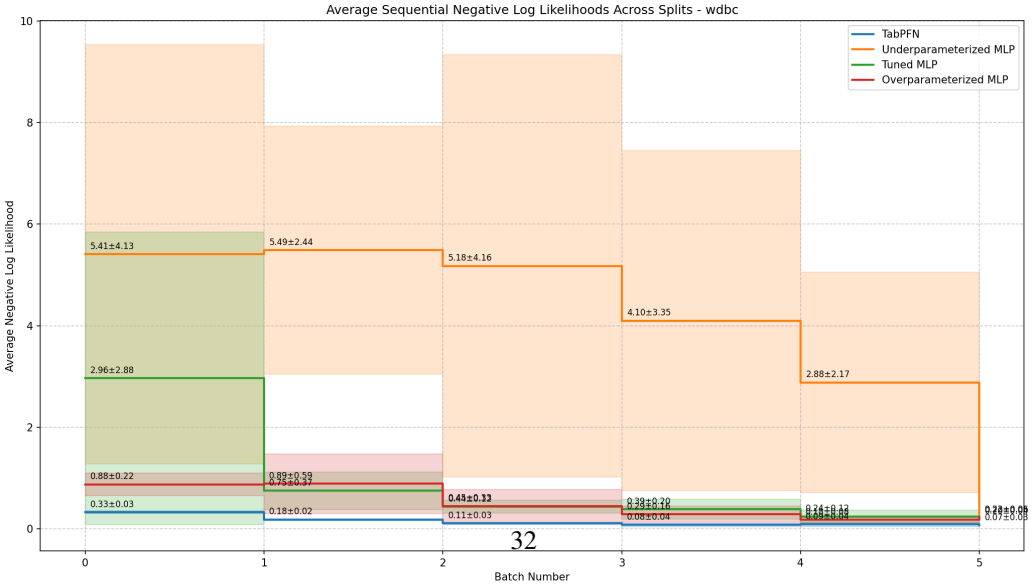

Average Sequential Negative Log Likelihoods Across Splits - pc4
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
5.75±5.77
5.73±5.75
5.05±4.49
3.74±1.82
3.29±1.03
3.28±1.74
2.95±0.44
2.65±0.23
2.47±0.48
4.45±3.67
3.72±2.22
2.44±0.25
3.18±0.90
2.73±0.64
2.39±0.28
2.97±1.95
2.72±1.99
1.76±0.47
3.22±1.14
1.88±0.54
1.20±0.92
0.34±0.03
0.33±0.04
0.30±0.03
0.27±0.04
0.30±0.07
0.24±0.03
0.23±0.03
0.20±0.02


Average Sequential Negative Log Likelihoods Across Splits - qsar-biodeg
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
1.46±0.60
1.42±0.44
1.37±0.52
1.23±0.43
1.28±0.38
1.05±0.25
1.04±0.30
0.91±0.30
0.92±0.31
0.80±0.10
0.70±0.17
0.77±0.13
0.81±0.13
0.61±0.05
0.64±0.18
0.57±0.15
0.45±0.07
0.45±0.07
0.54±0.08
0.50±0.12
0.39±0.05
0.38±0.07


Average Sequential Negative Log Likelihoods Across Splits - steel-plates-fault
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
18.84±2.08
17.14±1.54
16.57±1.55
17.50±0.83
15.84±1.87
15.03±1.55
14.55±1.41
15.01±2.02
15.12±1.69
14.79±1.34
14.20±2.31
13.74±1.74
12.68±3.69
8.64±5.81
1.68±0.03
1.44±0.02
1.31±0.11
1.07±0.04
0.94±0.09
0.86±0.05
0.70±0.07
0.61±0.05


Average Sequential Negative Log Likelihoods Across Splits - tic-tac-toe
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
1.95±0.59
1.69±0.43
1.51±0.50
2.68±0.55
2.21±0.39
1.95±0.44
2.26±0.33
1.74±0.29
1.33±0.27
1.22±0.17
1.36±0.16
1.22±0.43
0.95±0.41
0.79±0.18
0.69±0.04
0.77±0.11
0.68±0.04
0.63±0.01
0.58±0.03
0.39±0.02
0.66±0.15
0.56±0.21
0.44±0.05
0.27±0.02


Average Sequential Negative Log Likelihoods Across Splits - vehicle
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
6.62±1.67
5.29±2.45
3.74±1.58
3.31±1.20
3.84±2.34
4.26±2.95
2.39±1.73
1.84±0.58
2.04±1.70
1.23±0.36
1.39±0.09
1.26±0.16
1.01±0.09
0.62±0.04
0.45±0.05
0.39±0.07
0.32±0.04


Average Sequential Negative Log Likelihoods Across Splits - wdbc
TabPFN
Underparameterized MLP
Tuned MLP
Overparameterized MLP
Average Negative Log Likelihood
Batch Number
5.41±4.13
5.49±2.44
5.18±4.16
4.10±3.35
2.88±2.17
2.96±2.88
0.88±0.22
0.33±0.03
0.18±0.02
0.11±0.03
0.08±0.04
0.07±0.08