In-Context Function Estimation based on Transformer Models

Christian Schultze [1]

1. Introduction

Accurate modeling of a wide variety of functions enables the imputation of missing values in time series data and the creation of continuous data for further processing. We assume that for a given set of observations, there exists an underlying function that describes the data. Our goal is to model this underlying function such that we can predict all function values and therefore impute missing values. Empirical real-world data often consist of discrete measurements that need to be imputed to apply continuous mathematical tools.

One example from the field of astronomy are luminosity values from Cepheid variable stars. Cepheid stars change their luminosity periodically, as visible in Figure 1. Crucially, as discovered by the astronomer Henrietta Leavitt, the frequency of those periods directly correlates to their absolute luminosity. Knowing the absolute luminosity of a star is very useful, as it allows astronomers to calculate the distance to that star.

Before the twentieth century it was not possible to measure distances of more than a few hundred lightyears. This was important for the debate of whether the Andromeda galaxy, formerly known as the Andromeda nebula, is part of our own galaxy or rather a galaxy of its own. By discovering cepheid variable stars in the Andromeda galaxy and using Leavitts Law, the distance could be calculated and the Andromeda galaxy was recognized as such. (Müller, 2019)

Estimating the underlying function given the provided data in Figure 1 would allow removal of the present noise and apply further processing, such as calculating the extrema of that function and computing the frequency based on that. This would then enable us to calculate the distance to the star from which the data have been collected.

We use synthetic time-series data to train a model that interpolates the underlying function and imputes missing values. We sample our data from a multivariate normal distribution and train a transformer model to estimate unseen values given a certain number of context points. We follow the general approach from Seifner et al. (2025), while using transformers instead of LSTMs and estimating the actual function, instead of derivatives. We compare our results with a Gaussian Process baseline and discuss how the prior distribution and number of context points influences the performance. Finally, we use our transformer model to impute missing values for Cepheid star luminosity values and show that the model trained on synthetic data is applicable to real world data.

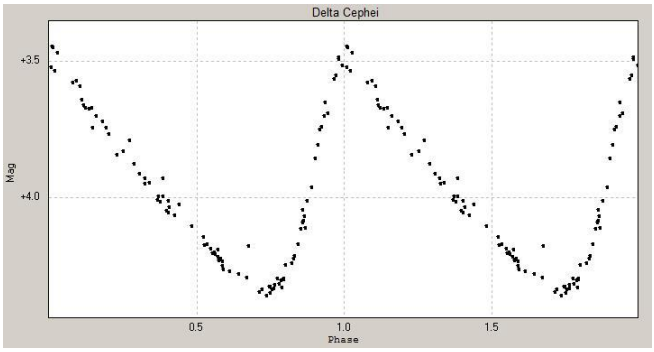

Figure 1. Lightcurve data from the variable star Delta Cephei, over two periods. https://de.wikipedia.org/wiki/Datei:Delta_Cephei_lightcurve.jpg

The entire code base, models and datasets are available online on github[1] and sciebo[2]

2. Preliminaries

In-context function estimation is the task of estimating the underlying function given a set of context points that have been sampled from that function or an empirical process. Using the estimated function, it is possible to impute missing data points. (Seifner et al., 2025)

This can be done using an underlying model assumption like with linear regression. A polynomial of a certain order is fitted to the given data, determining which parameters fit best to the data. For linear regression, it is assumed that the underlying function can be modeled by a polynomial. The

[1]Department of Computer Science, University of Bonn, Germany. Correspondence to: Christian Schultze <s6cnscul@uni-bonn.de>.

[1]https://github.com/ChristianSchultze/in-context-function-estimation.git

[2]https://github.com/ChristianSchultze/in-context-function-estimation.git

order of this polynomial must be determined in advance. Due to the best-fit approach, linear regression is not able to handle uncertainties that might result from missing data.

Gaussian Processes (GP) provide a way of estimating missing values of a function $f(x)$ that takes uncertainties into account. GPs are distributions over functions and we assume a prior distribution of possible underlying functions and update that belief based on the given context points. In practice we are dealing with discrete values, so GPs are defined by a mean value μ and a covariance matrix. The covariance matrix defines how strongly two values $f(x), f(x')$ vary from each other. Put differently, it defines how much they depend on each other. As we are dealing with 1-d functions, we expect functions values to depend on each other more strongly, the closer the input values are. One way of modelling this behaviour is to use the radial basis function defined in Equation 1. Each value of the covariance matrix is defined by this equation. Note that σ^2 is the variance of functions and l is the scaling parameter for all covariance values. Therefore, the lower the value of l, the more are the functions in the prior distribution expected to vary over time. (Garnelo et al., 2018; Williams & Rasmussen, 2006)

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{1}$$

Additionally to defining a prior distribution from which we can draw random functions, we want to update the prior based on context points and get the posterior distribution. This is done by conditioning the prior on the context points. This means removing functions from the prior distribution that do not agree with the context points. For this to be applicable to real data, we need to incorporate noise into the prior distribution. We assume additive Gaussian noise with variance σ_n^2 and add it to the diagonal of the covariance matrix. (Williams & Rasmussen, 2006)

Figure 2 shows some results of fitting Gaussian Processes on given context points while knowing the prior distribution from which the true underlying function has been sampled and the standard deviation of the added Gaussian noise. This posterior distribution defines a mean and standard deviation value for each point in the grid. In the leftmost plot, it is visible that missing data from about $x = 0.6$ to $x = 1$ leads to high uncertainty. As shown in the rightmost plot, we observe spikes and large uncertainty between context points. Specifically, the predicted mean goes to zero (the mean of the prior distribution) between context points. We speculate, that this is due to the low RBF-Scale, which leads to the GP assuming little or no correlation between points that we still consider to be close together. This leads to the posterior distribution reverting to the prior distributions mean of 0 and variance of 1.

For real-world scenarios, we cannot assume to know the prior distribution beforehand. Rather, we want to work with unseen functions directly without any prior knowledge, which inevitably introduces bias. We need to have a model that estimates the underlying function given only some context points and can impute missing values based on that. This in-context zero-shot function estimation can be split up into two subtasks. First, process the context points so that we get a representation of the underlying function and then predict the value of that function at a certain point.

Lu et al. (2021) train two neural networks for their DeepONet, that tackles these subtasks. The Branch net learns an operator $G : u \to G(u)$ with u being the sequence of context points and the result $G(u)$ is supposed to represent the underlying function. Using this representation and given an arbitrary point y, the Trunk net predicts the value $G(u)(y)$. (Lu et al., 2021)

Neural networks are not well fit to handle sequences of varying length, as we expect with the sequence of context points having a different length depending on the specific use case. Seifner et al. (2025) use a bi-directional LSTM that acts like the Branch net from DeepONets and allows processing sequences of varying length.

Adapting their approach, we use a transformer to model sequences of varying length. Transformer rely on the attention mechanism, introduced in Vaswani et al. (2017). As the name suggests, the attention mechanism enables models to focus their attention on inputs that are of particular importance. For this, three versions of the input called query, key and value are used. All three versions have their own set of weights for an initial linear projection. The query and key are matrix multiplied, effectively calculating the inner product between each element of the sequence. This can be interpreted as an similarity metric. The result of that matrix multiplication is scaled and normalized by the softmax function, resulting in a weight matrix of the same size as the original input. This weight matrix is applied to the values version of the input. So in essence elements of the input sequence that are similar to many other elements will be weighted higher than other elements. Crucially, the model can learn to adjust what exactly determines the similarity of two elements in the sequence, resulting in the learnable self-attention mechanism. This self attention is then performed multiple times independently in a certain number of attention heads, enabling the model to focus its attention on different things at the same time. (Vaswani et al., 2017)

3. Synthetic Data Generation

We sample functions from a multivariate normal distribution with an RBF kernel and a randomly chosen length scale. The functions are defined on a grid of 128 points in the interval

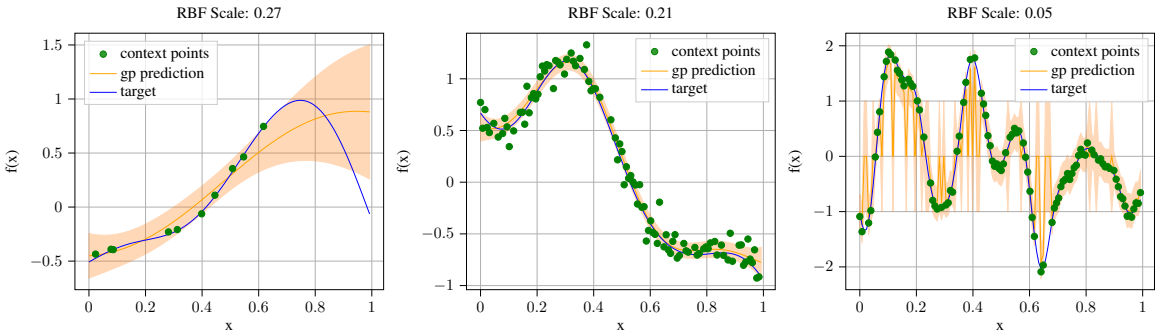

Figure 2. Gaussian Process function estimation compared to the target function for four example data frames with varying number of context points (5, 50, 50). Functions have been sampled from a multivariate normal distribution with an RBF kernel, the RBF-Scale is given on the top of each plot.

Table 1. Number of functions and context point subset size for training and test dataset. The number of context points for the training datasets vary, while for test datasets they are fixed.

	Train	Test			
Functions	100K	1K			
Subset size	5-50	5	10	50	100

Table 2. Beta distribution parameters and resulting mean for the sampled RBF-Scale values for both dataset versions.

	Version-A	Version-B
Alpha	2	2
Beta	10	5
Scale Mean	0.15	0.3

$[0, 1]$. From these 128 points, a subset of 5 to 50 points is chosen randomly to be the set of context points. This subset is the input data from which the underlying function must be estimated. Our train datasets consist of 100.000 functions and test-datasets consist of 1000 functions. The test datasets have fixed subset sizes (Table 1). We generate two versions of all datasets. In both versions the RBF-Scale values for each sampled function are drawn from a beta-distribution. Version-A uses parameters $\alpha = 2, \beta = 10$ and Version-B uses $\alpha = 2, \beta = 5$ (Table 2). In total, we generate two training datasets and eight test datasets. Figure 3 shows the distribution from which the RBF-Scale values are drawn in Version-A and Version-B. The lower the RBF-Scale, the stronger the sampled functions vary in a given time frame. We can see this exemplary in Figure 2, where the most right function with RBF-Scale of 0.05 varies much more than the left most function with RBF-Scale 0.27. Therefore, Version-A atasets contain much more strong varying functions and we consider it more difficult than Version-B.

The size n of the context point subsets is sampled uniformly

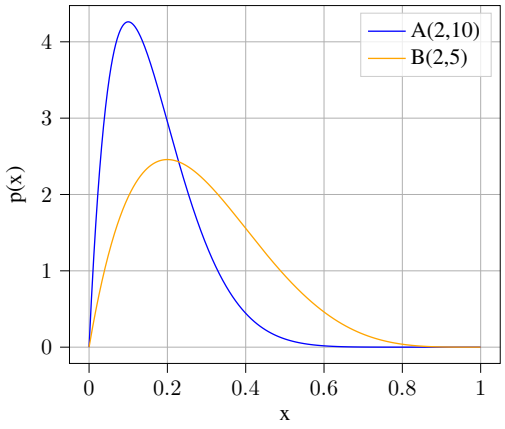

Figure 3. Beta distributions for Version-A and -B from which the RBF scale is sampled. We consider the datasets for Version-A to be more challenging, however it has to be considered, that there is a large overlap between the distributions.

$n \sim \mathcal{U}(5, 50)$ and the resulting distribution is used for both train datasets. For the test datasets, n is fixed. The subsets are sampled uniformly from all possible subsets of size n, with the original grid always being 128 points. To be more in line with real world observations we add noise to the context points. We draw the noise from $\mathcal{N}(0, \sigma)$ with a standard deviation $sigma$, which is itself sampled from $\mathcal{N}(0, 0.1)$.

4. Model

Our in-context function estimation model consists of an encoder and a decoder with two outputs. The encoder and the first decoder component can be interpreted as the Branch and Trunk nets from DeepONets Lu et al. (2021). Like Seifner et al. (2025) we predict the mean and logarithmic variance of a Gaussian distribution for each data point. Figure 4 shows our model architecture. We use the Attention is all you need

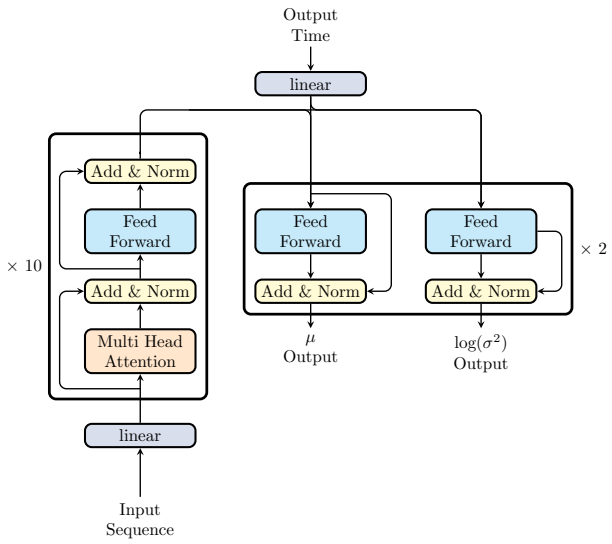

Figure 4. In-context function estimation model with an encoder on the left and a feed forward decoder on the right with two outputs. The encoder consists of a linear layer followed by 10 transformer layers and the decoder consists of two feed forward layers. The Output Time input at the top represents the time values at which the decoder should predict the mean and log variance. The encoder layers are identical to Vaswani et al. (2017)

transformer encoder (Vaswani et al. (2017)) with 10 layers to process the input sequence. The transformer encoder has a fixed number of channels and expects the input sequence to be in the same dimensional shape. Including time indices and function values, our input sequence has two channels. The linear projection layer in the encoder projects the sequence so that it has the desired number of channels. Linear projections for a sequence with shared weights are equivalent to 1d-convolutions, like Vaswani et al. (2017) we use 1d-convolutions for the linear projection in the encoder. The encoder outputs a sequence of the same length as the input sequence. To be processed by the feed forward networks of the decoder, we need a fixed size vector as output. To fix the location of our desired output vector, we append a learnable token as the first element of the sequence. Therefore, we are ignoring all but the first element of the output sequence and this first element is forwarded to the decoder. This vector of fixed size is supposed to store all relevant information from the context points to predict all function values given the desired time index.

The decoder receives the hidden vector from the encoder, representing the input data and a single time index. This is the desired time to which the model is supposed to predict the function value. Like in the encoder, the time index is projected to higher dimensions. Afterwards, it is concatenated with the hidden vector. In the decoder, we are not dealing with sequences, so the linear projection is done by a feed-forward network. The same is true for the feedforward layers within both decoder components. The two decoder components predict the mean and log variance for a specific point in time. When processing data during training and prediction, the encoder first processes the input sequence of known context points. Then decoder is then used to predict all points on a grid. We use a grid of 128 points on an interval of $[0, 1]$. This means that we run the decoder 128 times with the corresponding output time indices. The encoder result is only calculated once.

We normalize values and time indices as part of the model implementation. Like Seifner et al. (2025) we employ min-max normalization for the input sequence (time indices and values). During training, the target sequence is normalized as well and the loss is computed on the normalized data. This has the advantage that all functions in one batch are weighed equally independent of the scale at which they are operating. Equation 2 shows the general formula for normalizing data. During training, the min and max values for normalizing both the input and target values are determined from the input data. This has the advantage that all elements of one batch are weighed equally, independent of the scale of the data. When predicting results, we denormalize the mean and standard deviation values. Denormalizing predicted mean values in Equation 3 differs from denormalizing the standard deviation in Equation 4, as only its scale must be denormalized. Note that Equation 4 includes the predicted log variance for completeness.

$$x_i \leftarrow \frac{x_i - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{2}$$

$$\mu \leftarrow \mu(x_{\text{max}} - x_{\text{min}}) + x_{\text{min}} \tag{3}$$

$$\sigma = \sqrt{e^{\log(\sigma^2)}} x_{\text{max}} - x_{\text{min}} \tag{4}$$

5. Experiments

We train our model using the pytorch Lightning framework (Falcon, 2019). This has the advantage, that we do not need to implement the optimization process, logging and model saving ourself. We load hyper parameters from a YAML configuration file. This allows us to easily adjust hyper parameters for our experiments.

We train in batches with 256 padded sequences. These sequences contain indices and values. For input data, we use a padding mask to exclude padding values from normalization. Our target sequences are of a fixed length of 128 elements and therefore, no padding is needed. This means, padding values have not to be considered for loss calculation.

Our training objective is the Gaussian negative log-likelyhood loss (Equation 5), processing the outputs of both

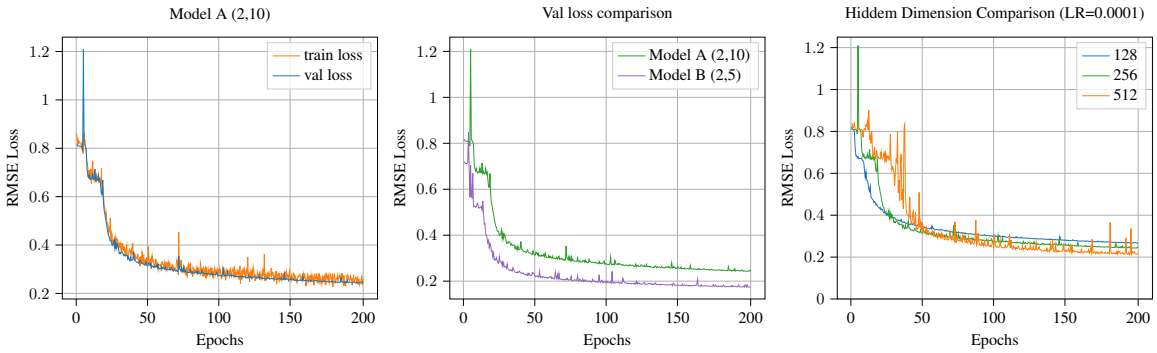

Figure 5. Convergence behaviour plots using the RMSE Loss regarding the mean prediction. Left: Comparison between train and validation loss for model A. Middle: Validation Loss comparison between model A and B. Right: Validation loss comparison between models with increasing hidden dimension.

Table 3. Training hyperparameters and model configuration.

Training		Model	
epochs	200	dimension	256
learning rate	1e-04	layers	10
weight decay	1e-04	heads	2
batch size	256		

decoders. This loss function enables the model to take into account the uncertainty that results from the gaps between the input points. Note that we are predicting μ and $\log(\sigma^2)$ with the decoder. This means that the model results replace the mean and log variance in the loss equation.

$$\mathcal{L} = \frac{1}{2}\left(\log(2\pi) + \log(\sigma^2) + \frac{((\text{target} - \mu)^2)}{2e^{\log(\sigma^2)}}\right) \quad (5)$$

We use the AdamW (Loshchilov et al. (2017)) optimizer without scheduler and validate twice per epoch. We save the best model regarding the validation GNNL loss, with the fixed validation dataset consisting of 10% of all data. If not otherwise stated, Table 3 lists the used hyperparameters. We train on a local GPU cluster, using one Nvidia GA102GL GPU, 50GB RAM and four cores of an Intel Xeon Silver 4309Y CPU. Training for 200 epochs takes about 3 hours, with each epoch completed in less than a minute.

We observe convergence for training models on both versions of the dataset, while model B shows a significantly lower validation loss than model A (Figure 5). This is because dataset B is significantly less challenging. Furthermore, we observe that increasing the number of hidden dimensions leads to a slight increase in performance, as well as a less stable training.

We use the posterior distribution of a Gaussian Process as the baseline. The GP has access to the true prior distribution from which the data have been sampled and the standard deviation with which the additive noise has been sampled. This baseline struggles with predicting values for functions, that have been sampled with a low RBF-Scale value. Table 4 shows that the Loss increases significantly increases when the RBF-Scale is below 0.1. Furthermore, the mean and standard deviation of the predicted σ values increases significantly. As explained in section 2, we speculate that this is the result of the model assuming no or little correlation between not so distant points and therefore reverting to the prior mean and variance values.

Table 4. Comparison between GP results on functions sampled with high and low RBF-Scale. For this experiment we generated another test dataset for Version-A, sampling the number of context points randomly between 5 and 50. From the 1000 generated functions, 294 have a RBF-Score of less than 0.1. We compare the RMSE Loss, as well as the standard deviation σ for each prediction. We compute the max and standard deviation value for all σ values in each prediction and report the average values over all predictions.

Scale	RMSE Loss	mean σ	std σ
≥ 0.1	0.53	0.51	0.21
< 0.1	0.86	0.80	0.36

6. Results

For evaluation, we use eight test datasets (four for each version) with a fixed number of context points (Table 1). We use the posterior distribution of a Gaussian Process as the baseline. All models are evaluated on all 1.000 functions in each dataset. As shown in Table 5, both models outperform the GP baseline. One reason for this are the uncertainty spikes visible in Figure 7. This happens for RBF-Scales below 0.1.

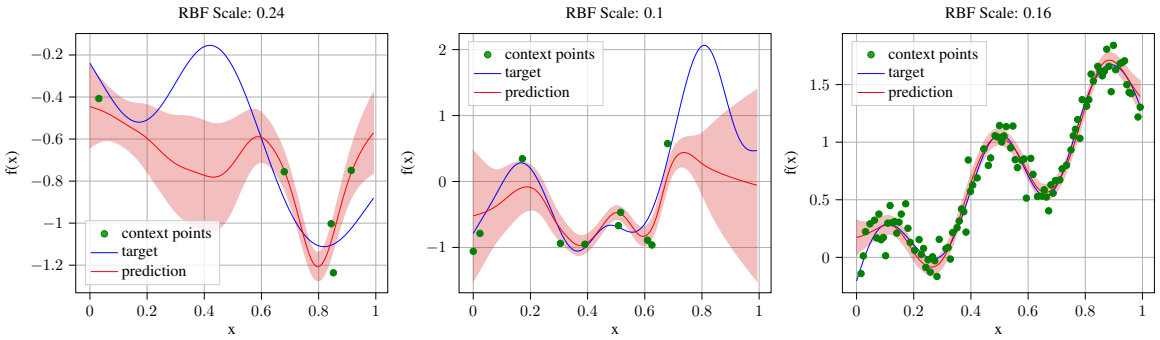

Figure 6. Model A prediction plots with increasing number of context points (5,10,50).

Table 5. Evaluation results on all eight test dataset with RMSE Loss for the mean prediction compared to the target values. We compare our GP baseline to both Model-A and -B, while both are tested with all test datasets and therefore evaluated on both dataset versions.

Dataset	Context Points	GP	Model-A	Model-B
A (2,10)	5	0.91	0.72	0.71
	10	0.78	0.50	0.49
	50	0.51	0.26	0.29
	100	0.32	0.24	0.28
B (2,5)	5	0.84	0.55	0.55
	10	0.63	0.37	0.36
	50	0.28	0.14	0.16
	100	0.16	0.13	0.16

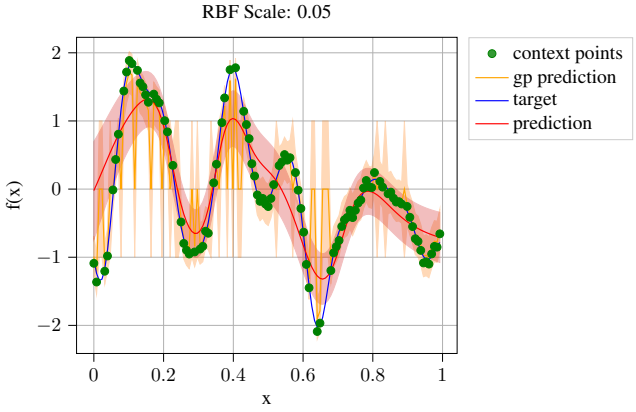

Figure 7. Comparison plots between model A and the GP baseline.

For all models, we observe increasing performance with an increasing number of context points. This is clearly visible in Figure 6, with the left and middle plots showing a lot of uncertainty due to the low number of context points. The right plot shows the prediction much closer to the target despite a large amount of noise. Furthermore, evaluations on the Version-B datasets show better performance than evaluations on Version-A datasets. This is expected as Dataset-A is more challenging. Model-B showing similar performance to Model-A on Dataset-A, despite being trained on the less challenging Dataset-B, might be sign of good generalization capabilities. However, as shown in Figure 3, the RBF-Scale distributions show a large overlap. For this reason, Model-B has seen sufficient examples of challenging functions during training and the similar performance cannot be attributed to generalization.

To apply our model to real-world data and seriously test its generalization capabilities, we process Cepheid star luminosity data. Figure 8 shows prediction results for Model-A. Due to missing ground truth data, we cannot evaluate the performance numerically. We observe a comparatively high uncertainty despite a high number of context points, espe-

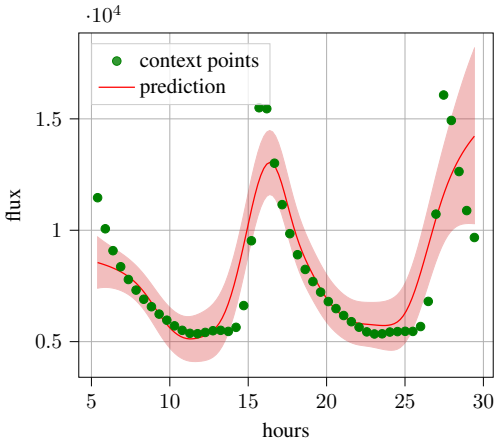

Figure 8. Model A Prediction result on luminosity values (unit flux) for a cepheid variable star over 24 hours. Data from Benkő et al. (Benkő et al., 2014)

cially during the low luminosity phases. Otherwise, these results show good generalization from training on synthetic data and applying the same model on real world observations.

7. Conclusion

This project demonstrates how to train a transformer-based model for in-context function estimation with synthetic 1d-time series data and apply this model successfully to real-world data. We show that our approach outperforms the Gaussian Process fitted on the same data, despite the GP knowing the true prior distribution. For both GPs and our model, data with strongly varying values or large noise levels pose a challenge. It should be further examined whether this is due insufficient information caused by a low resolution grid, or resulting from a less than optimal training process.

References

Benkő, J. M., Plachy, E., Szabó, R., Molnár, L., and Kolláth, Z. Long-timescale behavior of the blazhko effect from rectified kepler data. *The Astrophysical Journal Supplement Series*, 213(2):31, July 2014. ISSN 1538-4365. doi: 10.1088/0067-0049/213/2/31. URL http://dx.doi.org/10.1088/0067-0049/213/2/31.

Falcon, W. Pytorch lightning. https://github.com/Lightning-AI/lightning, 2019. Accessed: 2025-02-17.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

Loshchilov, I., Hutter, F., et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5(5): 5, 2017.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://dx.doi.org/10.1038/s42256-021-00302-5.

Müller, J. M. G. . J. Können wir die welt verstehen? pp. 125–131. S. Fischer, 1 edition, 2019.

Seifner, P., Cvejoski, K., Körner, A., and Sánchez, R. J. Zero-shot imputation with foundation inference models for dynamical systems, 2025. URL https://arxiv.org/abs/2402.07594.

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

Williams, C. K. and Rasmussen, C. E. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.