Amortized Inference for Gaussian Process Hyperparameters of Structured Kernels (Supplementary Material)

Matthias Bitzer[1] **Mona Meister**[1] **Christoph Zimmer**[1]

[1]Bosch Center for Artificial Intelligence, Renningen, Germany

In the following sections, we give further information about our method. We start by giving more experimental details. Then we illustrate the application of our method on an extended set of simulated datasets via plots of the predictive distributions. Finally, we give the proofs for the invariances and equivariances of our amortization network.

A EXPERIMENTAL DETAILS

Architecture - Dataset-Encoder. The dataset-encoder consists of four transformer blocks each consisting of a stack of Transformer-Encoder sublayers [Vaswani et al., 2017] where each sublayer has a multi-head-self-attention layer and an element-wise MLP layer as the two trainable parts. We don't use dropout and positional encodings in our architecture as it would disable equivariances [Lee et al., 2019].

Table 1: Dataset-Encoder Configuration.

Block	Num. of Layers	Embedding Dim.	Hidden Dim. in MLP
Transformer no. 1 (used in step 3)	4	256	512
Transformer no. 2 (used in step 5)	4	256	512
Transformer no. 3 (used in step 7)	4	512	512
Transformer no. 4 (used in step 9)	4	512	512

Architecture - Kernel-Encoder-Decoder. The kernel-encoder-decoder consists of two stacks of **Kernel-Encoder-Block** layers, which are also specified via an embedding dimension and the hidden dimension of its MLP layer. Furthermore, the kernel-encoder-decoder also contains one transformer block.

Table 2: Kernel-Encoder-Decoder Configuration.

Block	Num. of Layers	Embedding Dim.	Hidden Dim. in MLP
Kernel-Encoder-Block stack 1 (used in step 1)	3	512	1024
Transformer block (used in step 2)	4	512	1024
Kernel-Encoder-Block stack 2 (used in step 4)	3	512	1024

Architecture - Output layer. Each base-symbol specific **MLP** layer has one hidden layer with dimension $d_h = 200$. The **MLP** layer for the noise variance prediction consists of two hidden layers with dimension $d_{h_1} = 200$ and $d_{h_2} = 100$.

Sampling distribution. We use as base symbols $\mathrm{SE}, \mathrm{LIN}$ and PER and its two gram multiplications like, e.g. $\mathrm{SE} \times \mathrm{LIN}$. We simulate datasets of sizes between $n = 10$ and $n = 250$ (drawn uniformly) and input dimensions between $d = 1$ and $d = 8$. Here, we prefer smaller dimensions via sampling d from a geometric distribution with $p = 0.25$ and clip them to

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

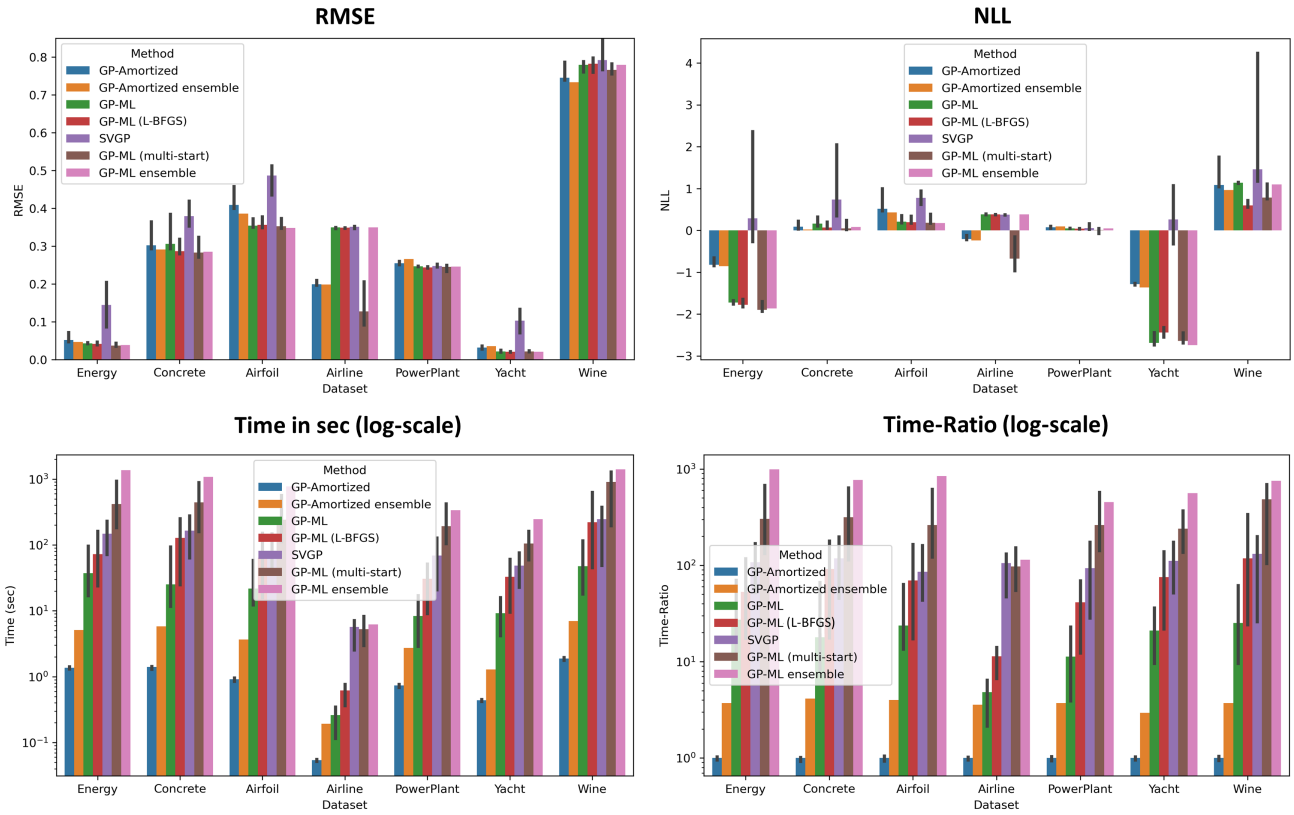

Figure 1: In the top-left plot the RMSE scores of all considered methods are shown evaluated on the respective test set. Several kernels $\{\mathcal{S}^{(1)}, \dots, \mathcal{S}^{(m)}\}$ are evaluated where each bar shows the median RMSE value and the error-bars show the 20th and 80th percentiles of the RMSE scores of the different kernels. In the top-right plot, the NLL scores are shown and in the bottom-right plot the corresponding ratios of inference times to our method are shown in log-scale. In the bottom-left plot, the absolute inference times in seconds are shown for the corresponding method.

$d \leq 8$. We sample the number of addends in each subexpression $\mathcal{S}_i$ also from a geometric distribution with $p = 0.6$. Each dataset is generated via sampling a kernel structure, sampling the input dataset uniformly from $[0, 1]^d$, then sampling the kernel parameters and noise variance from their prior. Finally, we sample the output of the dataset from the respective GP at the respective input locations. We use the following base kernel parameterizations,

$$k_{\text{SE}}(x, x') = \sigma_k^2 \exp\left(-\frac{1}{2}\frac{(x - x')^2}{l^2}\right), \quad k_{\text{PER}}(x, x') = \sigma_k^2 \exp\left(-\frac{1}{2}\frac{\sin^2(\pi|x - x'|/p)}{l^2}\right), \quad k_{\text{LIN}}(x, x') = \sigma_k^2 x\, y + \sigma_c^2$$

with priors, $\sigma_k^2 \sim \text{Gamma}(2.0, 3.0)$, $\sigma_c^2 \sim \text{Gamma}(2.0, 3.0)$, $p \sim \text{Gamma}(2.0, 3.0)$ and $l \sim \text{Gamma}(2.0, 5.0)$. For the noise variance we use the prior $\sigma^2 \sim \text{Exp}(\frac{1}{0.15^2})$.

Training - Phase 1 and 2. In the first phase, we train the amortization network via a batch-size of $B = 128$ for a total of 9 million datasets, where in each batch we simulate the datasets on-the-fly. Thus, each dataset in the training phase only appears once. We use RAdam as optimizer with a constant learning rate of $\text{lr} = 2 \times 10^{-5}$. In the second phase the extended loss is configured with parameters $\alpha = 10$ and $\beta = 1$. Besides that, we use the same configuration as in the first phase. We train in this phase on 200 thousand datasets.

Datasets. All datasets are publically available (Airline can be accessed for example via `https://kaggle.com`. The other datasets are UCI datasets from `https://archive.ics.uci.edu/ml/datasets.php`). We use for all datasets $n_{\text{train}} = 500$ except the smaller datasets Airline and Yacht for which we use 100 and 250 training datapoints. The training datapoints are drawn uniformly from the complete dataset. We use the rest of the dataset as test-set, clipped to a maximum of $n_{\text{test}} = 400$.

Type-2-ML GP settings. In our main evaluation, we consider three competitor approaches for full GP's with Type-2-ML hyperparameter inference: Firstly, GP-ML and GP-ML (multi-start) where we do Type-2-ML optimization via Adam with one and with multiple restarts. Here, we use the following setting. We use the Adam optimizer with a learning rate of 0.1 for 150 steps to optimize the marginal likelihood, where we early-stop the optimization once the marginal-likelihood has converged. We found that this setting provides a good trade-off between accuracy and inference time. For the single-run version we use the same initial parameter values for all datasets, where we set all initial parameter values of the kernels to one, e.g. $\sigma_k = 1.0$ and the initial likelihood noise to $\sigma = 0.2$. For the multi-start version, we do 10 restarts, where in each restart the initial parameter values are sampled from the respective prior. The methods are based on GPyTorch [Gardner et al., 2018]. Furthermore, we consider optimization of the marginal likelihood via L-BFGS, to which we refer to as GP-ML (L-BFGS). Here, we use GPflow [Matthews et al., 2017] as implementation and use the default optimization parameters of the library.

Sparse-Variational GP settings. For the Sparse-Variational GP (SVGP) we used $I = 0.5n$ number of inducing points, where n is the number of training points in the dataset. Furthermore, we use Adam to optimize the ELBO with respect to the inducing point locations and GP parameters. Here, we trained for 500 iterations (full-batch) with a learning rate of $lr = 0.03$, with early stopping once the ELBO converged.

Evaluation set of kernels. In the main evaluation and for fast-ensembling we utilize a set of 24 kernel structures $\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$. Here, we use in each input dimension the same substructure, thus $\mathcal{S}_i^{(l)} = \mathcal{S}_j^{(l)}$ for $j \neq i$ and all $l = 1, \ldots, m$ and generate the substructures in the following way. We include the set of base-symbols $\{\mathrm{SE}, \mathrm{PER}, \mathrm{LIN}, \mathrm{SE} \times \mathrm{PER}, \mathrm{SE} \times \mathrm{LIN}, \mathrm{LIN} \times \mathrm{PER}\}$. For example $\mathcal{S}^{(1)}$ with $\mathcal{S}_i^{(1)} = \mathrm{SE}_i, i = 1, \ldots, d$ and $\mathcal{S}^{(2)}$ with $\mathcal{S}_i^{(2)} = \mathrm{PER}_i, i = 1, \ldots, d$, are part of the set of 24 kernel structures. Furthermore, we sample six pairs, e.g. $\mathcal{S}_i = \mathrm{SE}_i + \mathrm{LIN}_i, i = 1, \ldots, d$, six triples and six quadruples from the set of base symbols each without replacement to generate the 24 kernel structures. In this way, we apply the same set of kernel (sub-)structures on each dataset - only the number of input dimension changes.

AHGP-SE-ARD settings. For the AHGP-SE-ARD model, we replicated the architecture of Rehn [2022], which uses the dataset-encoder provided by Liu et al. [2020] and a custom head to deal with the RBF kernel. We equip the dataset-encoder with 14 *LocalTransformer* blocks and 14 *GlobalTransformer* blocks (see Liu et al. [2020]) each specified with an embedding size of 512. Furthermore, we use for the lengthscale and variance MLP's (see Rehn [2022]) two hidden layers with 1024 and 512 nodes. We train the network on the same data distribution as our method for 9 million datasets, where we used RAdam as optimizer with a learning rate of $\mathrm{lr} = 2 \times 10^{-5}$ and a batch size of $L = 32$.

Fast-ensembling details. For our fast-ensembling approach, we use a set of kernel structures $\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$ and define the resulting predictive distribution of our ensemble model for a dataset $\mathcal{D}$ with input $\mathbf{X}$ and output $\mathbf{y}$ via

$$p(y^*|x^*, \mathcal{D}) := \frac{1}{m} \sum_{l=1}^{m} w_l \, p\left(y^* \middle| x^*, \mathcal{D}, \phi_{\mathcal{S}^{(l)}} = g_\psi(\mathcal{D}, \mathcal{S}^{(l)}) \right),$$

where $p\left(y^* \middle| x^*, \mathcal{D}, \phi_{\mathcal{S}^{(l)}} = g_\psi(\mathcal{D}, \mathcal{S}^{(l)}) \right)$ is the predictive distribution of the GP with kernel structure $\mathcal{S}^{(l)}$ and predicted parameters $\phi_{\mathcal{S}^{(l)}} = g_\psi(\mathcal{D}, \mathcal{S}^{(l)})$ and where $w_l = \frac{\tilde{w}_l}{\sum_{l=1}^{m} \tilde{w}_l}$ with $\tilde{w}_l := p(\mathbf{y}|\mathbf{X}, \phi_{\mathcal{S}^{(l)}} = g_\psi(\mathcal{D}, \mathcal{S}^{(l)}))$ is the (predicted) marginal likelihood value for kernel structure $\mathcal{S}^{(l)}$.

Comparison of all considered models. In Figure 1 the evaluation plots are shown for all considered methods. Here, we also show NLL scores. The biggest difference in NLL compared to the full-GP Type-2-ML methods is present for datasets where the predictions are already very precise (measured via RMSE) like on Energy or Yacht aka, where the NLL scores are already very small. It is worth noting that we outperform SVGP in terms of NLL on all datasets except Powerplant.

B FURTHER EXPERIMENTS

Qualitative Analysis of Predictive Distributions. We do a qualitative analysis of the predictive distributions for 1D datasets for different settings. In the plots (see for example Figure 2) datasets $\mathcal{D}^*$ are sampled from a GP with ground truth kernel $\mathcal{S}_{gt}$. We compare the predictive distributions induced by our predicted GP parameters $\phi^* = g_\psi(\mathcal{D}^*, \mathcal{S})$ to the predictive distributions induced by the learned GP parameters ϕ_{ML} via Type-2-ML and against the predictive distributions with the ground truth hyperparameters ϕ_{gt}.

The first setting (Figure 2) considers a *correctly specified kernel structure*, where the input kernel structure $\mathcal{S}$ is the same as the ground truth kernel structure $\mathcal{S}_{gt}$. In Figure 2, the predictive distributions of the three methods can be seen for four different kernel structures. We see that the noise range is captured well by our method, as well as the in-distribution data-fit.

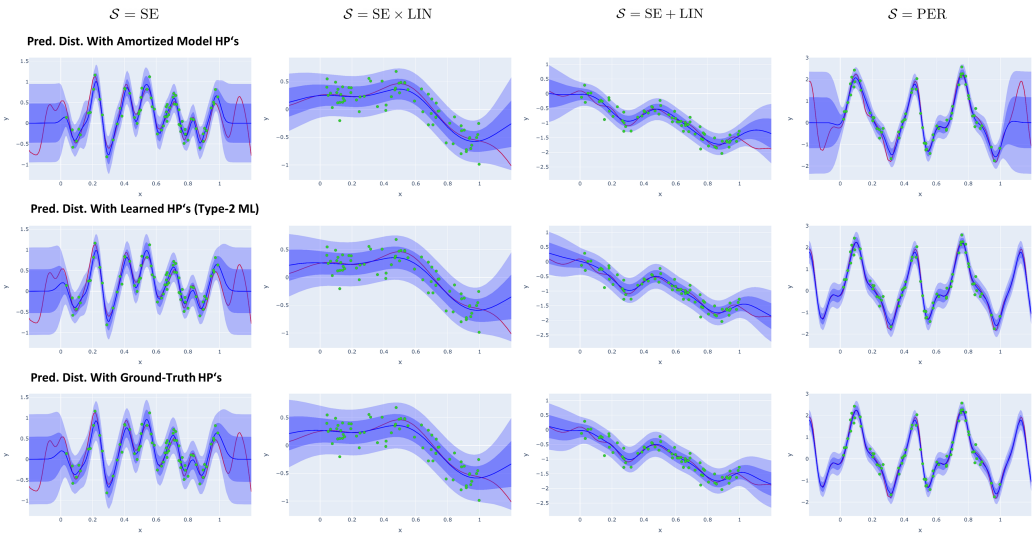

Figure 2: We show predictive distributions with kernel parameters of our method (top), Type-2-ML (middle) and with ground truth hyperparameters (bottom), where in each column a different input kernel structure $\mathcal{S}$ is used and $\mathcal{S} = \mathcal{S}_{gt}$.

In the extrapolation regime left and right of the data, the confidence intervals of our method appear reasonable for SE, SE $\times$ LIN and SE + LIN. However, the method makes a conservative prediction for the PER kernel via ignoring the periodicity in the data, whereas the type-2 ML and ground truth kernel extrapolate less conservative, which in this case leads to accurate predictions.

For the second setting (Figure 3) a *misspecified kernel structure* is used, where the input kernel structure $\mathcal{S}$ is not the same as the ground truth kernel structure $\mathcal{S}_{gt}$. We see for example for the dataset coming from the linear kernel (second column from left) that the predicted lengthscale of the specified SE kernel appears to be sufficiently large. When the linear kernel is applied to a dataset coming from an SE kernel (first column from left) we see that the noise variance prediction is higher

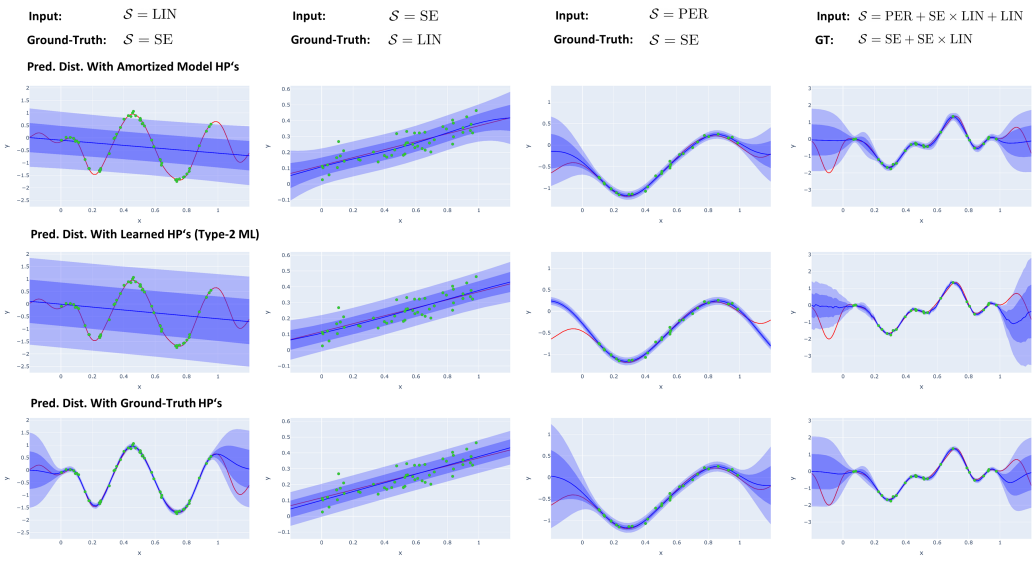

Figure 3: We show the predictive distributions with kernel hyperparameters predicted via our method (top), learned via Type-2-ML (middle) and with ground truth hyperparameters (bottom), where in each column a different input kernel $\mathcal{S}$ and a different ground-truth kernel $\mathcal{S}_{gt}$ is used and $\mathcal{S} \neq \mathcal{S}_{gt}$. In red, the underlying ground-truth function is shown.

than the ground-truth noise (which was $\sigma = 0.05$), which is desirable for the linear kernel as the GP needs to explain the data with noise in this case. Arguably, the most interesting case is, when a periodic kernel PER is specified as input and the input data is coming from an SE kernel (third column from left). Here, we see that the predictive-distribution of the Type-2-ML estimate extrapolates aggressively and overfits to the data (we made sure to find a very high marginal likelihood value here via repeating the optimization 30 times). In this case, the conservative prediction of our method for the periodic kernel is beneficial, as it leads to reasonable prediction intervals. As the last kernel pair, we use two more complex kernels as ground truth and input kernel. Here, we observe that our method can also deal with more complex kernels and leads to a reasonable predictive distribution.

In the last setting (Figure 4), we consider *smaller datasets* with $n = 10$. Here, we again use correctly specified kernels. We observe for all kernel structures reasonable prediction intervals and out-of-data predictions. We observe for example for

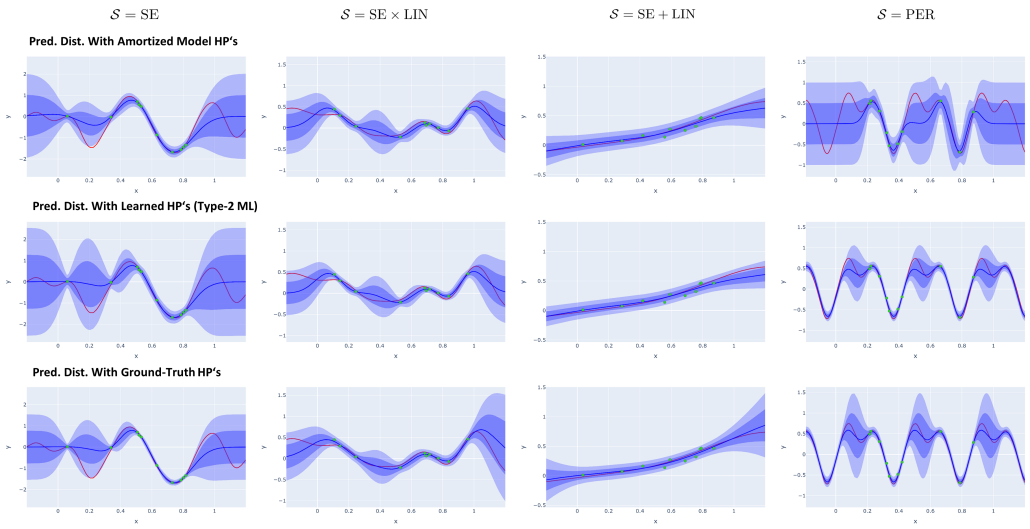

Figure 4: We show predictive distributions with kernel hyperparameters of our method (top), Type-2-ML (middle) and with ground truth hyperparameters (bottom), where in each column a different input kernel structure $\mathcal{S}$ is used and $\mathcal{S} = \mathcal{S}_{gt}$. In red we show the underlying ground-truth function. Here, we consider small datasets with $n = 10$.

the $\mathrm{SE} + \mathrm{LIN}$ that the method recognizes that the data is very linear already with ten datapoints. Notably, we see again a conservative prediction for the periodic kernel and a less conservative, but in this case accurate extrapolation for the Type-2-ML estimates.

Simulation analysis for smaller datasets on inference time. In Figure 5 we show prediction results on simulated data, with varying sizes of the training set $\mathcal{D}_{train}$ that is used for one-shot prediction of the kernel parameters. For each boxplot,

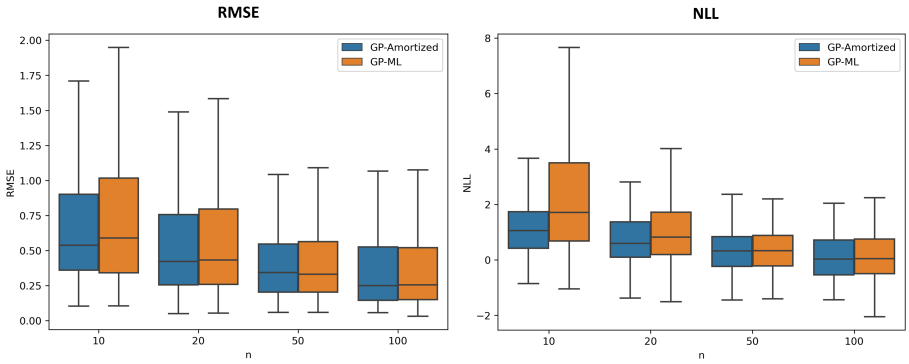

Figure 5: We show RMSE and NLL boxplots on simulated data (splitted into train/test) with $d = 2$ and varying n_{train}. Each sample is a dataset kernel pair $(\mathcal{D}, \mathcal{S})$ drawn from our training sampling distribution (except that d and n are fixed). We compare our method (GP-Amortized) against Type-2-ML (GP-ML).

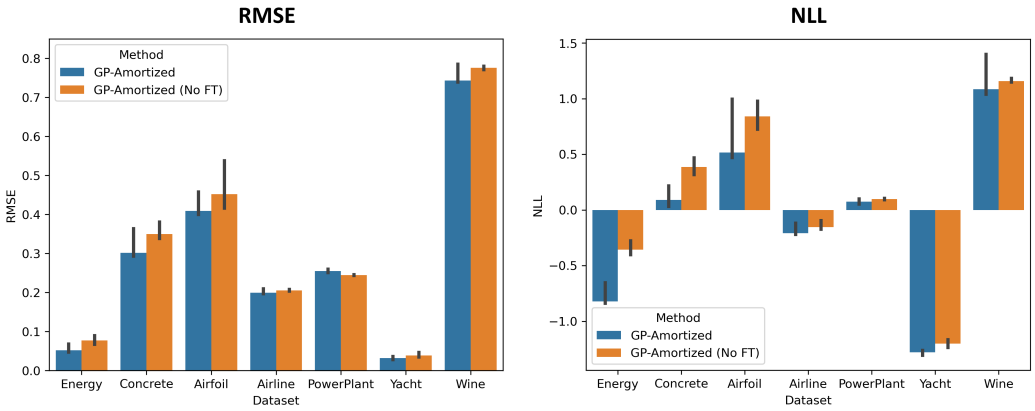

Figure 6: RMSE and NLL over kernel structures (error bars) and datasets for our method with noise-variance fine-tuning (GP-Amortized) and without fine-tuning [GP-Amortized (No FT)].

we simulate 200 dataset-kernel pairs $(\mathcal{D}, \mathcal{S})$ from our sampling distribution, where we fix $d = 2$ and vary the number of training datapoints $n_{train} = 10, 20, 50, 100$ in order to investigate the impact that the number of training inputs has on the performance difference of our method (GP-Amortized) against Type-2-ML (GP-ML). Similar to Liu et al. [2020], we find that in particular for smaller datasets amortization provides more robust kernel parameters, as can be seen for example in terms of improved NLL scores on smaller datasets. Thus, amortizing kernel parameter inference might provide regularization against overfitting of the kernel parameters and thus might be in particular useful in the small-data regime.

Effect of noise variance fine-tuning. We investigate the effect of noise variance fine-tuning on the real-world performance in Figure 6 and observe that on almost all datasets the fine-tuning has a beneficial effect on (median) RMSE and NLL scores. We think that this is the case because the marginal-likelihood landscape is multi-modal with respect to the noise value (high-noise vs low-noise explanation of the data) and the noise variance fine-tuning renders the loss landscape more well-behaved, similar to putting a prior on the GP parameters when doing maximum-a-posteriori (MAP) estimation.

Comparison to MAP estimation. Using simulated data from a hierarchical GP model with priors on the kernel hyperparameters encodes the prior to a certain extent into the amortization neural network, via implicitly encoding the range of parameter values in the datasets that are used in the training of the amortization network. Thus, one might compare our method against MAP estimation of the kernel parameters using the same priors. In Figure 7 we show a comparison to MAP estimation of the GP hyperparameters with standard ML estimation and with our method. We observe that MAP estimation provides a small benefit compared to marginal-likelihood maximization in terms of RMSE and NLL and also in terms of computation time - likely because of a more well-behaved loss landscape, where L-BFGS converges with fewer iterations. However, we note that our method is still orders of magnitudes faster.

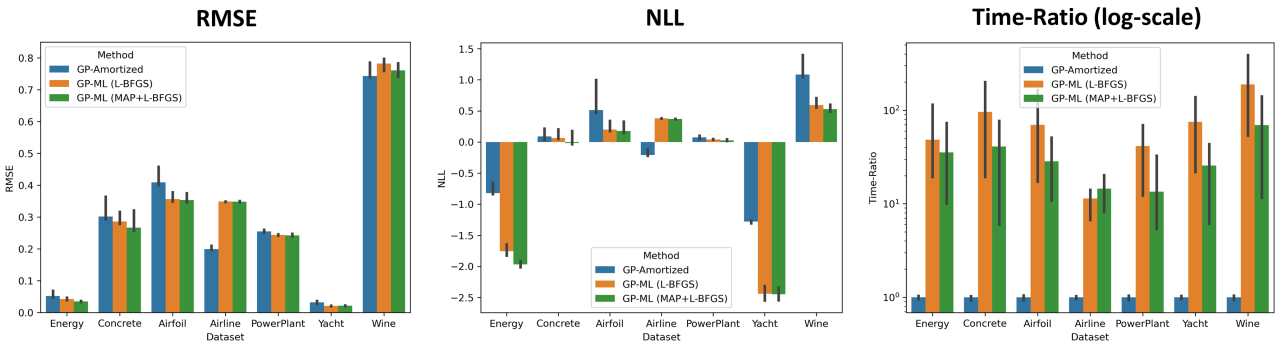

Figure 7: RMSE, NLL and time ratios over kernel structures (error bars) and datasets for our method, a GP trained with L-BFGS and a GP with the same priors on the kernel parameters as used for training data generation of the amortization neural network (here also L-BFGS was used).

C THEORY

Before starting to prove the invariance/equivariances in our network, we provide a high-level example on which invariances/equivariances are present in our architecture and why they are important. Our network predicts for a given dataset $\mathcal{D}$ and a given kernel structure $\mathcal{S}$ the kernel parameters and likelihood noise $(\hat{\theta}_{\mathcal{S}}, \hat{\sigma}^2) = g_\psi(\mathcal{D}, \mathcal{S})$. Firstly, we note that a reshuffling of the order of the dataset elements inside $\mathcal{D}$ should not change the output of g_ψ, since a reordering of the dataset also does not change the Type-2-ML maximization in the standard GP optimization. Therefore, it should hold for a shuffled dataset $\mathcal{D}_\pi$ that

$$(\hat{\theta}_{\mathcal{S}}, \hat{\sigma}^2) = g_\psi(\mathcal{D}, \mathcal{S}) = g_\psi(\mathcal{D}_\pi, \mathcal{S}) = (\hat{\theta}_{\mathcal{S},\pi}, \hat{\sigma}^2_\pi)$$

Furthermore, we consider kernel structures that are multiplications over dimensions and additions of base kernels inside dimensions. For example, on two dimensions we might consider

$$(\mathrm{SE}_1 + \mathrm{PER}_1) \times \mathrm{SE}_2.$$

As addition is commutative it should not make a difference for the values of the predicted parameters if we change the order of the addition for example to

$$(\mathrm{PER}_1 + \mathrm{SE}_1) \times \mathrm{SE}_2.$$

This property is captured in our architecture as the predicted parameters of the base symbols are equivariant to such a permutation, thus only the index changes after the permutation (now SE_1 is the second entry in the addition) but not the value of the predicted parameter ($\hat{\theta}_{\mathrm{SE}_1}$ is unaffected from the permutation).

A similar equivariance is captured in case the dimension index is reshuffled. A permutation of the input dimension effects both the dataset, that now has reshuffled dimensions indices, as well as the expression, that now also has reindexed dimensions. For example, swapping the two dimensions in our previous example leads to

$$\mathrm{SE}_1 \times (\mathrm{SE}_2 + \mathrm{PER}_2).$$

Such a permutation should not change the previously predicted parameter value $\hat{\theta}_{\mathrm{SE}_2}$ of the previous symbol SE_2 that is now, after reindexing of the dimension, the predicted parameter $\hat{\theta}_{\mathrm{SE}_1}$ of symbol SE_1. Here, our architecture is again equivariant meaning that a permutation like that only changes the index of the symbol but not the value of the predicted parameter. We show these properties formally in Theorem 4.

Next, we give the proofs for the claimed invariances of our amortization network. First, we state the formal definition of an invariant and an equivariant function f and prove some basic properties.

Definition 1 (Permutation Invariance) *Let $\mathcal{G}$ be the permutation group and $f : \mathbb{X}^N \to \mathbb{Y}$ a function with $N \in \mathbb{N}$. The function f is called permutation invariant iff*

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_N) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(N)}) = f(\pi \cdot \mathbf{x}), \quad \forall \pi \in \mathcal{G}, \mathbf{x} \in \mathbb{X}^N.$$

Definition 2 (Permutation Equivariant) *Let $\mathcal{G}$ be the permutation group and $f : \mathbb{X}^N \to \mathbb{Y}^N$ a function with $N \in \mathbb{N}$. The function f is called permutation equivariant iff*

$$\pi \cdot f(\mathbf{x}) = f(\pi \cdot \mathbf{x}), \quad \forall \pi \in \mathcal{G}, \mathbf{x} \in \mathbb{X}^N.$$

We first show that a function f that maps each of its sequence elements $x_i \in \mathbb{X}$ element-wise via a function g is permutation equivariant.

Lemma 1 *Let $f : \mathbb{X}^N \to \mathbb{Y}^N$ be a function with $N \in \mathbb{N}$ and let $f(x_1, \ldots, x_N) = [g(x_1), \ldots, g(x_N)] \in \mathbb{Y}^N$ for some $g : \mathbb{X} \to \mathbb{Y}$. Then f is permutation equivariant.*

Proof: Let $N \in \mathbb{N}$ and $\mathbf{x} = [x_1, \ldots, x_N] \in \mathbb{X}^N$ and let $\pi \in \mathcal{G}$. Let $y_j := g(x_j)$. Then

$$\begin{aligned} f(\pi \cdot \mathbf{x}) &= f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(N)}) = [g(x_{\pi(1)}), g(x_{\pi(2)}), \ldots, g(x_{\pi(N)})] \\ &= [y_{\pi(1)}, y_{\pi(2)}, \ldots, y_{\pi(N)}] = \pi \cdot [y_1, \ldots, y_N] = \pi \cdot f(\mathbf{x}). \end{aligned}$$

■

Next, we show that chaining a permutation equivariant function with a permutation invariant function leads to a permutation invariant function.

Lemma 2 *Let $f : \mathbb{X}^N \to \mathbb{Y}^N$ be a permutation equivariant function with $N \in \mathbb{N}$ and let $g : \mathbb{Y}^N \to \mathbb{V}$ be permutation invariant. Then $h : \mathbb{X}^N \to \mathbb{V}$ with $h = g \circ f$ is permutation invariant.*

Proof: Let $\mathbf{x} \in \mathbb{X}^N$ and $\pi \in \mathcal{G}$. Then

$$h(\pi \cdot \mathbf{x}) = g(f(\pi \cdot \mathbf{x})) = g(\pi \cdot f(\mathbf{x})) = g(f(\mathbf{x})) = h(\mathbf{x}).$$

■

Furthermore, the chaining of two permutation equivariant functions is permutation equivariant.

Lemma 3 *Let $f : \mathbb{X}^N \to \mathbb{Y}^N$ be a permutation equivariant function with $N \in \mathbb{N}$ and let $g : \mathbb{Y}^N \to \mathbb{V}^N$ be permutation equivariant. Then $h : \mathbb{X}^N \to \mathbb{V}^N$ with $h = g \circ f$ is permutation equivariant.*

Proof: Let $\mathbf{x} \in \mathbb{X}^N$ and $\pi \in \mathcal{G}$. Then

$$h(\pi \cdot \mathbf{x}) = g(f(\pi \cdot \mathbf{x})) = g(\pi \cdot f(\mathbf{x})) = \pi \cdot g(f(\mathbf{x})) = \pi \cdot h(\mathbf{x}).$$

■

Furthermore, the chaining of a permutation invariant function with any function is permutation invariant.

Lemma 4 *Let $f : \mathbb{X}^N \to \mathbb{Y}$ be a permutation invariant function with $N \in \mathbb{N}$ and let $g : \mathbb{Y} \to \mathbb{V}$. Then $h : \mathbb{X}^N \to \mathbb{V}$ with $h = g \circ f$ is permutation invariant.*

Proof: Let $\mathbf{x} \in \mathbb{X}^N$ and $\pi \in \mathcal{G}$. Then

$$h(\pi \cdot \mathbf{x}) = g(f(\pi \cdot \mathbf{x})) = g(f(\mathbf{x})) = h(\mathbf{x}).$$

■

Furthermore, we make use of the following mappings that are used in our architecture:

1. **Transformer-Encoder without positional encoding**: We use the encoder part of the transformer architecture without positional-encoding and without dropout [Vaswani et al., 2017, Lee et al., 2019] and denote the mapping as $\mathbf{Transformer} : \mathbb{X}^n \to \mathbb{X}^n$ with $\mathbb{X} \subset \mathbb{R}^h$. This block uses a stack of multi-head-self-attention layer [Vaswani et al., 2017] denoted as mapping $\mathbf{MHSA} : \mathbb{X}^n \to \mathbb{X}^n$. Both, the **MHSA** (without dropout) and $\mathbf{Transformer}$ (without positional encoding and dropout) are permutation-equivariant as shown in Lee et al. [2019] Property 1 and Section 3.4.
2. **Mean aggregation:** The mean aggregation $\mathbf{MeanAGG} : \mathbb{X}^N \to \mathbb{X}$ with $\mathbb{X} \subset \mathbb{R}^h$ is permutation invariant.

C.1 MAIN INVARIANCES AND EQUIVARIANCES

Next, we will prove the mentioned invariances for the several parts of the amortization network. We denote the dataset with $\mathcal{D} = \{(x_j, y_j) \in \mathbb{R}^{d+1} | j = 1, \ldots, n\} = \{(x_j^{(1)}, \ldots, x_j^{(d)}, y_j) \in \mathbb{R}^{d+1} | j = 1, \ldots, n\}$. A permutation π of the dataset elements is then a shuffling of $[(x_1, y_1), \ldots, (x_n, y_n)]$ to $[(x_{\pi(1)}, y_{\pi(1)}), \ldots, (x_{\pi(n)}, y_{\pi(n)})]$. We denote the reindexed dataset with $\mathcal{D}_\pi$. A permutation $\hat{\pi}$ of the dimensions is a reshuffling of $\mathcal{D} = \{(x_j^{(1)}, \ldots, x_j^{(d)}, y_j) \in \mathbb{R}^{d+1} | j = 1, \ldots, n\}$ to $\{(x_j^{(\hat{\pi}(1))}, \ldots, x_j^{(\hat{\pi}(d))}, y_j) \in \mathbb{R}^{d+1} | j = 1, \ldots, n\}$. We denote the resulting dataset with $\mathcal{D}_{\hat{\pi}}$.

Theorem 1 *Let $\mathcal{D}$ be a dataset with $n \in \mathbb{N}$ elements and $d \in \mathbb{N}$ input-dimensions. The dataset-encoder g_D as a mapping from the dataset $\mathcal{D}$ to embedding vectors $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ (described through the steps 1.-9.) is permutation-invariant to a shuffling of the dataset elements (in the sense that $\mathbf{h}_i$ is invariant for all $i = 1, \ldots, d$) and permutation-equivariant to a shuffling of the dimensions.*

Proof: We go along the computation steps 1.-9. and keep track of the invariances/equivariances of the respective quantities. We note that applying multiple equivariant transformations in a row, keeps the equivariance according to Lemma 3. For each quantity we add the respective computation step index as right-most lower index to each quantity. We also do that with the weights of the respective learnable function.

1. Here, the dataset $\mathcal{D}$ is transformed into dimension-wise sequences $\mathcal{H}^{(i)} = [(x_j^{(i)}, y_j)]_{j=1}^n$ where we denote $h_{j,1}^{(i)} := (x_j^{(i)}, y_j)$.
   Dataset shuffling: When we shuffle the dataset elements via permutation π this translates to shuffeled sequences $[(x_{\pi(j)}^{(i)}, y_{\pi(j)})]_{j=1}^n$ applied to all dimensions $i = 1, \ldots, d$. Thus, the sequences $[h_{j,1}^{(i)}]_{j=1}^n$ are equivariant to a dataset shuffling.
   Dimension shuffling: When we shuffle the dimension via a permutation $\hat{\pi}$ this translates to the shuffling of the sequence of sequences $[\mathcal{H}^{(1)}, \ldots, \mathcal{H}^{(d)}]$ to $[\mathcal{H}^{(\hat{\pi}(1))}, \ldots, \mathcal{H}^{(\hat{\pi}(d))}]$ by definition of $\mathcal{H}^{(i)}$. The sequence of sequences itself $[\mathcal{H}_1^{(1)}, \ldots, \mathcal{H}_1^{(d)}]$ is thus equivariant, to a shuffling of the input dimensions.

2. Each element $h_{j,1}^{(i)}$ in $\mathcal{H}_1^{(i)}$ is mapped element-wise via the same linear layer $\mathbf{Linear}_{W_2} : \mathbb{R}^2 \to \mathbb{R}^h$ with weights W_2 to $h_{j,2}^{(i)}$:

$$h_{j,2}^{(i)} = \mathbf{Linear}_{W_2}\left(h_{j,1}^{(i)}\right).$$

   Dataset shuffling: From Lemma 1 with $g = \mathbf{Linear}_{W_2}$ it follows that the mapped sequence $\mathcal{H}_2^{(i)} = [h_{1,2}^{(i)}, \ldots, h_{n,2}^{(i)}]$ is equivariant to a dataset shuffling for all $i = 1, \ldots, d$.
   Dimension shuffling: As the same mapping is applied for all dimensions $i = 1, \ldots, d$ via

$$\mathcal{H}_2^{(i)} = \mathbf{ElemLinear}_{W_2}(\mathcal{H}_1^{(i)}) := [\mathbf{Linear}_{W_2}(h_{1,1}^{(i)}), \ldots, \mathbf{Linear}_{W_2}(h_{n,1}^{(i)})],$$

   the sequence of transformed sequences $[\mathcal{H}_2^{(1)}, \ldots, \mathcal{H}_2^{(d)}] = [\mathbf{ElemLinear}_{W_2}(\mathcal{H}_1^{(1)}), \ldots, \mathbf{ElemLinear}_{W_2}(\mathcal{H}_1^{(d)})]$ is equivariant to dimension shuffling with Lemma 1 and $g = \mathbf{ElemLinear}_{W_2}$.

3. Here, the $\mathbf{Transformer}_{W_3}$ block with weights W_3 is applied and results in the transformed sequence $\mathcal{H}_3^{(i)} := [h_{1,3}^{(i)}, \ldots, h_{n,3}^{(i)}] = \mathbf{Transformer}_{W_3}([h_{1,2}^{(i)}, \ldots, h_{n,2}^{(i)}])$.
   Dataset shuffling: As the $\mathbf{Transformer}_{W_3}$ block is permutation equivariant, the transformed sequence $\mathcal{H}_3^{(i)} = [h_{1,3}^{(i)}, \ldots, h_{n,3}^{(i)}] = \mathbf{Transformer}_{W_3}([h_{1,2}^{(i)}, \ldots, h_{n,2}^{(i)}])$ is equivariant to dataset shuffling for all $i = 1, \ldots, d$.
   Dimension shuffling: As we apply the same (weight-shared) transformer block $\mathbf{Transformer}_{W_3}$ to all dimensions $i = 1, \ldots, d$, and applying Lemma 1 with $g = \mathbf{Transformer}_{W_3}$, the transformed sequence of sequences

$$[\mathcal{H}_3^{(1)}, \ldots, \mathcal{H}_3^{(d)}] = [\mathbf{Transformer}_{W_3}(\mathcal{H}_2^{(1)}), \ldots, \mathbf{Transformer}_{W_3}(\mathcal{H}_2^{(d)})]$$

   is equivariant to input dimension shuffling.

4. Here, we first construct the combined (over $i = 1, \ldots, d$) sequences $\mathcal{W}_{j,4} := [h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}]$ and apply mean aggregation on them via

$$h_{j,4} := \mathbf{MeanAGG}(\mathcal{W}_{j,4}) = \mathbf{MeanAGG}([h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}])$$

   to form the sequence $[h_{1,4}, \ldots, h_{n,4}]$.
   Dataset shuffling: As $[h_{1,3}^{(i)}, \ldots, h_{n,3}^{(i)}]$ is equivariant to dataset shuffling for all $i = 1, \ldots, d$, also the sequence $[\mathcal{W}_{1,4}, \ldots, \mathcal{W}_{n,4}]$ of the combined (over $i = 1, \ldots, d$) elements $\mathcal{W}_{j,4} := [h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}]$ is equivariant to dataset shuffling. As we apply the same function to each $\mathcal{W}_{j,4}$, via $h_{j,4} := \mathbf{MeanAGG}(\mathcal{W}_{j,4})$, also the sequence $[h_{1,4}, \ldots, h_{n,4}]$ is equivariant to dataset shuffling via Lemma 1.
   Dimension shuffling: As $[\mathcal{H}_3^{(1)}, \ldots, \mathcal{H}_3^{(d)}]$ is equivariant to dimension shuffling each sequence $[h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}]$ is equivariant to dimension shuffling for all $j = 1, \ldots, n$. Applying the mean aggregation $h_{j,4} = \mathbf{MeanAGG}([h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}])$ onto $[h_{j,3}^{(1)}, \ldots, h_{j,3}^{(d)}]$ renders via Lemma 2 the quantity $h_{j,4}$ invariant to dimension shuffling and thus also the complete sequence $[h_{1,4}, \ldots, h_{n,4}]$.

5. Here, we apply the transformer block $\textbf{Transformer}_{W_5}$ with weights W_5 on $[h_{1,4}, \ldots, h_{n,4}]$ to form the sequence

$$[h_{1,5}, \ldots, h_{n,5}] = \textbf{Transformer}_{W_5}([h_{1,4}, \ldots, h_{n,4}]).$$

Dataset shuffling: As $[h_{1,4}, \ldots, h_{n,4}]$ is equivariant to dataset shuffling and the transformer block $\textbf{Transformer}_{W_5}$ with weights W_5 keeps equivariance also the transformed sequence $[h_{1,5}, \ldots, h_{n,5}]$ is equivariant to dataset shuffling.
Dimension shuffling: The previous sequence $[h_{1,4}, \ldots, h_{n,4}]$ is invariant to dimension shuffling, and the invariance of $[h_{1,5}, \ldots, h_{n,5}]$ to dimension shuffling is preserved via Lemma 4.

6. Here, we concatenate for each $i = 1, \ldots, d$ the sequences $\mathcal{H}_3^{(i)} = [h_{1,3}^{(i)}, \ldots, h_{n,3}^{(i)}]$ from step 3 with $[h_{1,5}, \ldots, h_{n,5}]$ via

$$h_{j,6}^{(i)} := \textbf{Concat}(h_{j,5}, h_{j,3}^{(i)})$$

to form the combined sequences $\mathcal{H}_6^{(i)} := [h_{1,6}^{(i)}, \ldots, h_{n,6}^{(i)}]$ with $i = 1, \ldots, d$.

Dataset shuffling: As the sequences $\mathcal{H}_3^{(i)} = [h_{1,3}^{(i)}, \ldots, h_{n,3}^{(i)}]$ from step 3. are equivariant to a shuffling of the dataset elements for all $i = 1, \ldots, d$ as well as the sequence $[h_{1,5}, \ldots, h_{n,5}]$ from step 5 also the concatenation of the elements of the two sequences is permutation equivariant. Thus, the transformed sequence $\mathcal{H}_6^{(i)} := [h_{1,6}^{(i)}, \ldots, h_{n,6}^{(i)}]$ is permutation equivariant to dataset shuffling for all $i = 1, \ldots, d$.
Dimension shuffling: The sequence of transformed sequences $[\mathcal{H}_6^{(1)}, \ldots, \mathcal{H}_6^{(d)}]$ is permutation equivariant to a shuffling of the dimensions as we applied the same function to all sequences, via

$$\mathcal{H}_6^{(i)} = \textbf{ElemConcat}([h_{j,5}]_{j=1}^n, \mathcal{H}_3^{(i)}) = [\textbf{Concat}(h_{1,5}, h_{1,3}^{(i)}), \ldots, \textbf{Concat}(h_{n,5}, h_{n,3}^{(i)})],$$

where $[h_{j,5}]_{j=1}^n$ is invariant to the shuffling of the dimensions (see step 5).

7. Here, we apply the $\textbf{Transformer}_{W_7}$ block with weights W_7 to get the transformed sequence

$$[h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}] = \textbf{Transformer}_{W_7}([h_{1,6}^{(i)}, \ldots, h_{n,6}^{(i)}]).$$

Dataset shuffling: The $\textbf{Transformer}_{W_7}$ block with weights W_7 is equivariant, therefore is the transformed sequence $[h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}]$ equivariant to dataset shuffling for all $i = 1, \ldots, d$.
Dimension shuffling: As we use the same transformer for all dimensions $i = 1, \ldots, d$, the transformed sequence of sequences

$$[\mathcal{H}_7^{(1)}, \ldots, \mathcal{H}_7^{(d)}] = [\textbf{Transformer}_{W_7}(\mathcal{H}_6^{(1)}), \ldots, \textbf{Transformer}_{W_7}(\mathcal{H}_6^{(d)})]$$

keeps its equivariance to the shuffling of the input dimensions.

8. Here, we apply the mean aggregation

$$\mathbf{h}_{i,8} = \textbf{MeanAGG}([h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}])$$

on each sequence $\mathcal{H}_7^{(i)} = [h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}]$ to form the compressed sequence $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$

Dataset shuffling: As the mean aggregation is invariant to shuffling and the sequences $\mathcal{H}_7^{(i)} = [h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}]$ are equivariant to a shuffling of the dataset elements, with Lemma 2, it follows that the vector $\mathbf{h}_{i,8} = \textbf{MeanAGG}([h_{1,7}^{(i)}, \ldots, h_{n,7}^{(i)}])$ is invariant to the shuffling of the dataset elements and thus also the complete sequence $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$.

Dimension shuffling: As we apply the same function (mean aggregation) to all sequences $\mathcal{H}_7^{(i)}, i = 1, \ldots, d$ and the sequence of sequences $[\mathcal{H}_7^{(1)}, \ldots, \mathcal{H}_7^{(d)}]$ is permutation equivariant to dimension shuffling, also the output embeddings sequence $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$ is permutation equivariant to dimension shuffling.

9. In the last step we apply a $\textbf{Transformer}_{W_9}$ with weights W_9 to $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$ to get the sequence

$$[\mathbf{h}_{1,9}, \ldots, \mathbf{h}_{d,9}] = \textbf{Transformer}_{W_9}([\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]).$$

Dataset shuffling: As $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$ was invariant to to dataset shuffling before the transformation, it is still after the transformation, via Lemma 4.
Dimension shuffling: As the transformer block $\textbf{Transformer}_{W_9}$ with weights W_9 is equivariant and the input sequence $[\mathbf{h}_{1,8}, \ldots, \mathbf{h}_{d,8}]$ is equivariant, also the sequence $[\mathbf{h}_{1,9}, \ldots, \mathbf{h}_{d,9}]$ is equivariant to a shuffling of the input dimensions.
■

Next, we will prove invariances of the kernel-encoder-decoder block $g_k(\mathbf{h}_\mathcal{D}, \mathcal{V}_\mathcal{S})$ that maps a sequence of sequences $\mathcal{V}_\mathcal{S} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$ of base kernel representations, with $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ and the sequence of dataset embeddings $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ to a transformed sequence of sequences $[\tilde{\mathcal{V}}_1, \ldots, \tilde{\mathcal{V}}_d]$ of learned base kernel representations. In particular, when we refer to a permutation $\hat{\pi}$ of the input dimensions of the datset we implicity assume that the input sequence $[\mathcal{V}_1, \ldots, \mathcal{V}_d]$ is shuffled in the same way, thus to $[\mathcal{V}_{\hat{\pi}(1)}, \ldots, \mathcal{V}_{\hat{\pi}(d)}]$. Furthermore, as one sequence $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ describes an addition of kernels we also consider the permutation equivariance of these subsequences in the sense that for any dimension $i \in \{1, \ldots, n\}$ the output sequence, denoted by $\tilde{\mathcal{V}}_i = [\tilde{v}_1^{(i)}, \ldots, \tilde{v}_{N_i}^{(i)}]$ is permutation equivariant to a permutation $\tilde{\pi}$ of the input sequence $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$. We call it permutation of the base-symbols within dimension i.

First, we will show that the **Kernel-Encoder-Block**, as a function from a sequence $[v_1, \ldots, v_N]$ and a context c to a transformed sequence $[v_1, \ldots, v_N]$, is permutation equivariant given the context c.

Theorem 2 *The **Kernel-Encoder-Block** g as described in Figure 1 b) in the main paper is permutation equivariant as a mapping from a sequence $[v_1, \ldots, v_N]$ with $v_j \in \mathbb{R}^h$ to a sequence $[v_1, \ldots, v_N]$ given a context vector $c \in \mathbb{R}^l$, meaning for a permutation $\pi \in \mathcal{G}$ and a context vector $c \in \mathbb{R}^l$:*

$$g(\pi \cdot [v_1, \ldots, v_N], c) = \pi \cdot g([v_1, \ldots, v_N], c).$$

Proof: The **Kernel-Encoder-Block** first maps the sequence through a multi-head-self-attention **MHSA** layer with output sequence $[\tilde{v}_1, \ldots, \tilde{v}_N]$. After that it adds the input sequence $[v_1, \ldots, v_N]$ element-wise to the output sequence and applies a layer normalization step element-wise. We denote the output of that operation as $[\bar{v}_1, \ldots, \bar{v}_N]$. This sequence is concatenated with the context vector to $[(\bar{v}_1, c), \ldots, (\bar{v}_N, c)]$. This is given element-wise to a shared **MLP** layer, where we denote the output with $[\hat{v}_1, \ldots, \hat{v}_N]$. This is followed by an addition of $[\bar{v}_1, \ldots, \bar{v}_N]$ to $[\hat{v}_1, \ldots, \hat{v}_N]$ and a layer normalization step that is applied element-wise.

As **MHSA** is permutation equivariant the sequence $[\tilde{v}_1, \ldots, \tilde{v}_N]$ is permutation equivariant and thus also the addition to the original sequence $[v_1, \ldots, v_N]$. As the layer normalizaton is applied element-wise, via Lemma 1, the sequence $[\bar{v}_1, \ldots, \bar{v}_N]$ is permutation equivariant. As we concatenate equivariant elements $\bar{v}_j$ with a fixed quantity c, the sequence $[(\bar{v}_1, c), \ldots, (\bar{v}_N, c)]$ is permutation equivariant to a shuffling of $[v_1, \ldots, v_N]$. The $\mathbf{MLP}_W$ with weights W is applied element-wise:

$$[\hat{v}_1, \ldots, \hat{v}_N] = [\mathbf{MLP}_W(\bar{v}_1, c), \ldots, \mathbf{MLP}_W(\bar{v}_N, c)]$$

and via Lemma 1 it holds that the output sequence is permutation-equivariant. As $[\bar{v}_1, \ldots, \bar{v}_N]$ is permutation equivariant as well as $[\hat{v}_1, \ldots, \hat{v}_N]$ their addition is permutation equivariant. As the final layer normalization is applied element-wise, via Lemma 1, the final output is permutation equivariant.■

Next, we prove the mentioned invariances for the kernel-encoder-decoder g_k.

Theorem 3 *Let $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ be the output of g_D for the dataset $\mathcal{D}$ and let $\mathcal{V}_\mathcal{S} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$ be the sequence of sequences of input base-symbol encodings. The output sequence of sequences $[\tilde{\mathcal{V}}_1, \ldots, \tilde{\mathcal{V}}_d]$ of the kernel-encoder-decoder $g_k(\mathbf{h}_\mathcal{D}, \mathcal{V}_\mathcal{S})$ is permutation equivariant to a shuffling of the dimensions. Furthermore, for any dimension $i \in \{1, \ldots, d\}$ the single output sequence $\tilde{\mathcal{V}}_i = [\tilde{v}_1^{(i)}, \ldots, \tilde{v}_{N_i}^{(i)}]$ is equivariant to a permutation π of the respective input sequence $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$.*

Proofs: First, we show that a stacked version of the **Kernel-Encoder-Block** is also permutation equivariant given a context (just as a single **Kernel-Encoder-Block** according to Theorem 2). To show that, we denote the **Kernel-Encoder-Block** as the mapping $\mathbf{KernelEncoder}_W : \mathbb{X}^N \times \mathbb{R}^l \to \mathbb{X}^N$ with weights W and a stacked version with

$$\mathbf{KernelEncoderStacked}(\mathcal{V}, c) := \mathbf{KernelEncoder}_{\tilde{W}_1}(\mathbf{KernelEncoder}_{\tilde{W}_2}(\mathcal{V}, c), c)$$

here w.l.o.g with two layers and weights $\tilde{W}_1, \tilde{W}_2$. It is permutation equivariant given a context $c \in \mathbb{R}^h$ with $h \in \mathbb{N}$ since

$$\begin{aligned}
\pi\mathbf{KernelEncoderStacked}(\mathcal{V}, c) &= \pi\mathbf{KernelEncoder}_{\tilde{W}_1}(\mathbf{KernelEncoder}_{\tilde{W}_2}(\mathcal{V}, c), c) \\
&= \mathbf{KernelEncoder}_{\tilde{W}_1}(\pi\mathbf{KernelEncoder}_{\tilde{W}_2}(\mathcal{V}, c), c) \\
&= \mathbf{KernelEncoder}_{\tilde{W}_1}(\mathbf{KernelEncoder}_{\tilde{W}_2}(\pi\mathcal{V}, c), c).
\end{aligned}$$

We denote a stack of **Kernel-Encoder-Block** layers with weights $\tilde{W}_1, \ldots, \tilde{W}_k$ with $\mathbf{KernelEncoderStacked}_W$ where we summarize all weights to W.

Now, let $\mathcal{D}$ be a dataset with $n \in \mathbb{N}$ datapoints and $d \in \mathbb{N}$ input-dimensions. Let $[\mathbf{h}_1, \ldots, \mathbf{h}_d] = g_D(\mathcal{D})$ and let $\mathcal{V}_\mathcal{S} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$ be the sequence of sequences of input base-symbol encodings with $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$. Furthermore, let $l \in \{1, \ldots, d\}$ be the dimension on which we apply the permutation $\tilde{\pi}$ of the base symbol sequence $\mathcal{V}_l = [v_1^{(l)}, \ldots, v_{N_l}^{(l)}]$.

Similar as in Theorem 1 we now go along the computation steps 1.-4. of the kernel-encoder-decoder and consider the invariance/equivariances of the respective quantities. We add the respective computation step index as right-most lower index to each quantity and weight of the respective learnable function:

1. Here, each sequence $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}], i = 1, \ldots, d$ is given to a $\mathbf{KernelEncoderStacked}_{W_1}$ block with weights W_1 together with context $\mathbf{h}_i$:

   $$[v_{1,1}^{(i)}, \ldots, v_{N_i,1}^{(i)}] = \mathbf{KernelEncoderStacked}_W([v_1^{(i)}, \ldots, v_{N_i}^{(i)}], \mathbf{h}_i)$$

   Shuffling of base symbols in dimension l: As $\mathbf{KernelEncoderStacked}_{W_1}$ is equivariant given a context (and we set the context to the fixed quantity $\mathbf{h}_l$), the transformed sequence $\mathcal{V}_{l,1} = [v_{1,1}^{(l)}, \ldots, v_{N_l,1}^{(l)}]$ is permutation equivariant to a shuffling of the base-symbols in dimension l.

   Shuffling of dimension: The sequence $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ is permutation equivariant to a shuffling of the input dimensions according to Theorem 1 and the sequence $[\mathcal{V}_1, \ldots, \mathcal{V}_d]$ by definition. Thus, the sequence of tuples $[(\mathcal{V}_1, \mathbf{h}_1), \ldots, (\mathcal{V}_d, \mathbf{h}_d)]$ is equivariant to a shuffling of the dimension and we apply the same $\mathbf{KernelEncoderStacked}_{W_1}$ block to each tuple with

   $$[\mathcal{V}_{1,1}, \ldots, \mathcal{V}_{d,1}] = [\mathbf{KernelEncoderStacked}_{W_1}(\mathcal{V}_1, \mathbf{h}_1), \ldots, \mathbf{KernelEncoderStacked}_{W_1}(\mathcal{V}_d, \mathbf{h}_d)].$$

   Thus, the transformed sequence of sequences $[\mathcal{V}_{1,1}, \ldots, \mathcal{V}_{d,1}]$ is equivariant to dimension shuffling, according to Lemma 1 with $g = \mathbf{KernelEncoderStacked}_{W_1}$.

2. Here, we apply a mean aggregation on the sequences $\mathcal{V}_{i,1} = [v_{1,1}^{(i)}, \ldots, v_{N_i,1}^{(i)}], i = 1, \ldots, d$ via

   $$\mathbf{v}_{i,2} = \mathbf{MeanAGG}([v_{1,1}^{(i)}, \ldots, v_{N_i,1}^{(i)}])$$

   to construct the sequence $[\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}]$.

   Shuffling of base symbols in dimension l: As the sequence $[v_{1,1}^{(l)}, \ldots, v_{N_l,1}^{(l)}]$ is equivariant to a shuffling of the base-symbols in dimension l and the mean aggregation is permutation invariant, the kernel embedding in dimension l, given as $\mathbf{v}_{l,2} = \mathbf{MeanAGG}([v_{1,1}^{(l)}, \ldots, v_{N_l,1}^{(l)}])$, is invariant to a shuffling of the base-symbols in dimension l via Lemma 2. For any other dimension $i \neq l$ the quantity $\mathbf{v}_{i,2}$ is also invariant to a shuffling of base symbols in dimension l, as the permuted sequence $\mathcal{V}_l$ is not an input to that quantity. Thus, the complete sequence $[\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}]$ is invariant to a shuffling of base symbols in dimension l.

   Shuffling of dimension: Furthermore, the sequence of embeddings $[\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}]$ is equivariant to dimension shuffling as we apply the same mapping onto the equivariant sequence $[\mathcal{V}_{1,1}, \ldots, \mathcal{V}_{d,1}]$ via

   $$[\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}] = [\mathbf{MeanAGG}(\mathcal{V}_{1,1}), \ldots, \mathbf{MeanAGG}(\mathcal{V}_{d,1})]$$

   and applying Lemma 1 with $g = \mathbf{MeanAGG}$.

3. Here, we apply a $\mathbf{Transformer}_{W_3}$ with weights W_3 to get the transformed sequence

   $$[\mathbf{v}_{1,3}, \ldots, \mathbf{v}_{d,3}] = \mathbf{Transformer}_{W_3}([\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}]).$$

   Shuffling of base symbols in dimension l: As $[\mathbf{v}_{1,2}, \ldots, \mathbf{v}_{d,2}]$ is invariant to a shuffling of base symbols in dimension l, also the transformation $[\mathbf{v}_{1,3}, \ldots, \mathbf{v}_{d,3}]$ is, via Lemma 4.

   Shuffling of dimension: As the transformer block $\mathbf{Transformer}_{W_3}$ is equivariant, the equivariance to dimension shuffling is preserved for the transformed sequence $[\mathbf{v}_{1,3}, \ldots, \mathbf{v}_{d,3}]$

4. In the last step, each sequence $\mathcal{V}_{i,1} = [v_{1,1}^{(i)}, \ldots, v_{N_i,1}^{(i)}]$ from step 1 with $i = 1, \ldots, d$ is given to a $\mathbf{KernelEncoderStacked}_{W_4}$ block with weights W_4:

   $$[v_{1,4}^{(i)}, \ldots, v_{N_i,4}^{(i)}] = \mathbf{KernelEncoderStacked}_{W_4}([v_{1,1}^{(i)}, \ldots, v_{N_i,1}^{(i)}], c_{i,4})$$

together with the extended context $c_{i,4}$, where

$$c_{i,4} = \mathbf{Concat}(\mathbf{h}_i, \mathbf{v}_{i,3}).$$

Shuffling of base symbols in dimension l: As $\mathbf{v}_{l,3}$ is invariant to a base-symbol shuffling within dimension l also the concatenated context $c_{l,4} = \mathbf{Concat}(\mathbf{h}_l, \mathbf{v}_{l,3})$ is invariant to that shuffling. Thus, independent of the shuffling of the base-symbols within the dimension l, the context vectors stays the same and we can apply Theorem 2. Thus, via Theorem 2 the $\mathbf{KernelEncoderStacked}_{W_4}$ is equivariant given the context $c_{l,4}$, therefore, with the fact that $[v^{(l)}_{1,1}, \dots, v^{(l)}_{N_l,1}]$ is equivariant to a shuffling of the base symbols in dimension l we conclude that also the transformed sequence $\mathcal{V}_{l,4} = [v^{(l)}_{1,4}, \dots, v^{(l)}_{N_l,4}]$ is permutation equivariant to a shuffling of the base-symbols in dimension l.
Shuffling of dimension: The sequence of input tuples $[(\mathcal{V}_{1,1}, c_{1,4}), \dots, (\mathcal{V}_{d,1}, c_{d,4})]$ is equivariant to a shuffling of the dimension as $[c_{1,4}, \dots, c_{d,4}]$ and $[\mathcal{V}_{1,1}, \dots, \mathcal{V}_{d,1}]$ are equivariant. Applying the same $\mathbf{KernelEncoderStacked}_{W_4}$ block to each tuple renders the transformed sequence

$$[\mathcal{V}_{1,4}, \dots, \mathcal{V}_{d,4}] = [\mathbf{KernelEncoderStacked}_{W_4}(\mathcal{V}_{1,1}, c_{1,4}), \dots, \mathbf{KernelEncoderStacked}_{W_4}(\mathcal{V}_{d,1}, c_{d,4})]$$

equivariant to dimension shuffling, according to Lemma 1 with $g = \mathbf{KernelEncoderStacked}_{W_4}$. ■

Noise Variance Predictor: We give a short detailed notation for the noise variance predictor. The noise variance predictor g_{NV} gets as input the output of the dataset encoder $[\mathbf{h}_i]_{i=1}^d$ and the output of the kernel-encoder-decoder $\tilde{V}_{\mathcal{S}} = [\tilde{\mathcal{V}}_1, \dots, \tilde{\mathcal{V}}_d]$ and maps to a single real value $\hat{\sigma}^2 = g_{NV}([\mathbf{h}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}})$ that is the prediction for the noise variance σ^2. It is defined via:

$$g_{NV}([\mathbf{h}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}}) = \mathbf{MLP}_W\left(\mathbf{Concat}\left(\mathbf{MeanAgg}([\mathbf{h}_i]_{i=1}^d), \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}})\right)\right)$$

where $\mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}})$ is the mean over all $\tilde{v}^{(i)}_j$ in $\tilde{V}_{\mathcal{S}}$ and W are the weights of the **MLP** layer.

Kernel Parameter Predictor: We give a short detailed notation for the kernel parameter predictor $g_{\Theta_{\mathcal{S}}}$. It gets as input the output of the kernel-encoder decoder $\tilde{V}_{\mathcal{S}} = [\tilde{\mathcal{V}}_1, \dots, \tilde{\mathcal{V}}_d]$ and the sequence of sequences of base-symbols $\mathcal{B}_{\mathcal{S}} := \left[[B^{(i)}_1, \dots, B^{(i)}_{N_i}] | i = 1, \dots, d\right]$ of $\mathcal{S}$ with $B^{(i)}_j \in \mathcal{B}$ and is defined via

$$g_{\Theta_{\mathcal{S}}}(\tilde{V}_{\mathcal{S}}, \mathcal{B}_{\mathcal{S}}) := \left[[\mathbf{MLP}_{B^{(i)}_1}(v^{(i)}_1), \dots, \mathbf{MLP}_{B^{(i)}_{N_i}}(v^{(i)}_{N_i})] \middle| i = 1, \dots, d\right]$$

where for each symbol $B \in \mathcal{B}$ in the corpus $\mathcal{B}$ a separate $\mathbf{MLP}_B$ layer with weights W_B is used.

The next theorem is our main theorem and combines the previous theorems to state the main invariances of the output quantities, namely the predicted kernel parameters and the predicted likelihood noise variance.

Theorem 4 *Let $\mathcal{D}$ be a dataset with $d \in \mathbb{N}$ input dimensions and $n \in \mathbb{N}$ datapoints and $\mathcal{S}$ an expression as described in Section 2 of the paper. For the output of our amortization network $g(\mathcal{D}, \mathcal{S})$ given through the prediction of the kernel parameters $\theta_{B^{(i)}_j} \in \Theta_{B^{(i)}_j}$ for each base symbol in $\mathcal{S}$ with $i = 1, \dots, d$ and $j = 1, \dots, N_i$ and a prediction of the likelihood variance σ^2 the following properties hold:*

1. *The output is invariant to a shuffling of the dataset elements, meaning for any valid $\mathcal{S}$ and shuffeled dataset $\mathcal{D}_\pi$ it holds:*

$$g(\mathcal{D}, \mathcal{S}) = g(\mathcal{D}_\pi, \mathcal{S}).$$

2. *The output kernel parameters are equivariant to a dimension shuffling, meaning for a dimension shuffeled dataset $\mathcal{D}_{\hat{\pi}}$ and a dimension shuffeld expression $\mathcal{S}_{\hat{\pi}}$ with permuted sequence of sequence of base-kernels $\mathcal{B}_{\mathcal{S}_{\hat{\pi}}}$ it holds:*

$$\hat{\theta}_{\hat{B}^{(i)}_j} = \theta_{B^{(\hat{\pi}(i))}_j} \in \Theta_{\hat{B}^{(i)}_j} = \Theta_{B^{(\hat{\pi}(i))}_j},$$

where $\hat{\theta}_{\hat{B}^{(i)}_j}$ is the output of $g(\mathcal{D}_{\hat{\pi}}, \mathcal{S}_{\hat{\pi}})$ for symbol $\hat{B}^{(i)}_j \in \mathcal{B}_{\mathcal{S}_{\hat{\pi}}}$ and $\theta_{B^{(i)}_j}$ is the output of $g(\mathcal{D}, \mathcal{S})$ for symbol $B^{(i)}_j \in \mathcal{B}_{\mathcal{S}}$.

3. *For any $i \in \{1, \dots, d\}$ the output kernel parameters are equivariant to a permutation $\tilde{\pi}$ of the the base-symbols inside the subexpression $\mathcal{S}_i$ meaning for the shuffeled expression $\mathcal{S}_{\tilde{\pi}}$ with permuted sequence of sequence of base-kernels $\mathcal{B}_{\mathcal{S}_{\tilde{\pi}}}$ it holds for all $j = 1, \dots, N_i$:*

$$\hat{\theta}_{\hat{B}_j^{(i)}} = \theta_{B^{(i)}_{\tilde{\pi}(j)}} \in \Theta_{\hat{B}_j^{(i)}} = \Theta_{B^{(i)}_{\tilde{\pi}(j)}},$$

where $\hat{\theta}_{\hat{B}_j^{(i)}}$ is the output of $g(\mathcal{D}, \mathcal{S}_{\tilde{\pi}})$ for symbol $\hat{B}_j^{(i)} \in \mathcal{B}_{\mathcal{S}_{\tilde{\pi}}}$ and $\theta_{B_j^{(i)}}$ is the output of $g(\mathcal{D}, \mathcal{S})$ for symbol $B_j^{(i)} \in \mathcal{B}_{\mathcal{S}}$.

4. *The likelihood variance prediction is invariant to a permutation of the input dimension $\hat{\pi}$ and invariant to a permutation $\tilde{\pi}$ of the the base-symbols inside a subexpression $\mathcal{S}_i$ applied to any input dimension $i \in \{1, \dots, d\}$.*

Proof:
1. For the noise variance prediction $\hat{\sigma}^2$ it holds that:

$$\hat{\sigma}^2_{\pi} = g_{NV}(g_D(\mathcal{D}_{\pi}), g_k(g_D(\mathcal{D}_{\pi}), \mathcal{V}_{\mathcal{S}})) = g_{NV}(g_D(\mathcal{D}), g_k(g_D(\mathcal{D}), \mathcal{V}_{\mathcal{S}})) = \hat{\sigma}^2$$

as $g_D(\mathcal{D}) = g_D(\mathcal{D}_{\pi})$ according to Theorem 1. For the kernel parameter prediction $\hat{\theta}_{\mathcal{S}}$ it holds similarly

$$\hat{\theta}^{\pi}_{\mathcal{S}} = g_{\Theta_{\mathcal{S}}}(g_k(g_D(\mathcal{D}_{\pi}), \mathcal{V}_{\mathcal{S}}), \mathcal{B}_{\mathcal{S}}) = g_{\Theta_{\mathcal{S}}}(g_k(g_D(\mathcal{D}), \mathcal{V}_{\mathcal{S}}), \mathcal{B}_{\mathcal{S}}) = \hat{\theta}_{\mathcal{S}}$$

as $g_D(\mathcal{D}) = g_D(\mathcal{D}_{\pi})$ according to Theorem 1.

2. Let $\hat{\pi}$ be a permutation applied to the input dimensions. Let $[\hat{\mathcal{V}}_1, \dots, \hat{\mathcal{V}}_d]$ be the output of the kernel encoder-decoder for the permuted input and $[\tilde{\mathcal{V}}_1, \dots, \tilde{\mathcal{V}}_d]$ for the unpermuted input. As $\hat{B}_j^{(i)} = B_j^{(\hat{\pi}(i))}$ (by definition) and $\hat{v}_j^{(i)} = \tilde{v}_j^{(\hat{\pi}(i))}$ by Theorem 3 with $\hat{v}_j^{(i)} \in \hat{\mathcal{V}}_i$ and $\tilde{v}_j^{(i)} \in \tilde{\mathcal{V}}_i$ it follows

$$\hat{\theta}_{\hat{B}_j^{(i)}} = \mathbf{MLP}_{\hat{B}_j^{(i)}}(\hat{v}_j^{(i)}) = \mathbf{MLP}_{B_j^{(\hat{\pi}(i))}}(\tilde{v}_j^{(\hat{\pi}(i))}) = \theta_{B_j^{(\hat{\pi}(i))}}.$$

3. Let $i \in \{1, \dots, d\}$ and let $\tilde{\pi}$ be a permutation of the base symbols in dimension i. Let $\tilde{\mathcal{V}}_i = [\tilde{v}_1^{(i)}, \dots, \tilde{v}_{N_i}^{(i)}]$ be the output of the kernel-encoder-decoder for the unpermuted input and $\hat{\mathcal{V}}_i = [\hat{v}_1^{(i)}, \dots, \hat{v}_{N_i}^{(i)}]$ for the permuted input. As $\hat{B}_j^{(i)} = B_{\tilde{\pi}(j)}^{(i)}$ (by definition) and $\hat{v}_j^{(i)} = \tilde{v}_{\tilde{\pi}(j)}^{(i)}$ by Theorem 3 it follows

$$\hat{\theta}_{\hat{B}_j^{(i)}} = \mathbf{MLP}_{\hat{B}_j^{(i)}}(\hat{v}_j^{(i)}) = \mathbf{MLP}_{B^{(i)}_{\tilde{\pi}(j)}}(\tilde{v}^{(i)}_{\tilde{\pi}(j)}) = \theta_{B^{(i)}_{\tilde{\pi}(j)}}.$$

4. Let $\hat{\pi}$ be a permutation applied to the input dimensions. Let $[\mathbf{h}_i]_{i=1}^d$ be the output of the dataset-encoder for the unpermuted input dataset $\mathcal{D}$ and let $[\hat{\mathbf{h}}_i]_{i=1}^d$ be the output of the dataset-encoder for the permuted input dataset $\mathcal{D}_{\hat{\pi}}$. Furthermore, let $\tilde{\mathcal{V}}_{\mathcal{S}_{\hat{\pi}}}$ be the output of the kernel-encoder-decoder for the permuted expression $\mathcal{S}_{\hat{\pi}}$. Then, it holds for the noise variance prediction:

$$\begin{aligned}
\hat{\sigma}^2_{\hat{\pi}} = g_{NV}([\hat{\mathbf{h}}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}_{\hat{\pi}}}) &= \mathbf{MLP}_W\left(\mathbf{Concat}\left(\mathbf{MeanAgg}([\hat{\mathbf{h}}_i]_{i=1}^d), \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}_{\hat{\pi}}})\right)\right) \\
&= \mathbf{MLP}_W\left(\mathbf{Concat}\left(\mathbf{MeanAgg}([\mathbf{h}_i]_{i=1}^d), \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}})\right)\right) \\
&= g_{NV}([\mathbf{h}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}}) = \hat{\sigma}^2,
\end{aligned}$$

since $\mathbf{MeanAgg}([\hat{\mathbf{h}}_i]_{i=1}^d) = \mathbf{MeanAgg}([\mathbf{h}_i]_{i=1}^d)$ and $\mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}_{\hat{\pi}}}) = \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}})$ as the mean aggregation is permutation invariant. For the permutation $\tilde{\pi}$ of the base symbols in dimension i the proof goes accordingly via permuted output $\tilde{\mathcal{V}}_{\mathcal{S}_{\tilde{\pi}}}$ of the kernel-encoder-decoder and

$$\begin{aligned}
\hat{\sigma}^2_{\tilde{\pi}} = g_{NV}([\mathbf{h}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}_{\tilde{\pi}}}) &= \mathbf{MLP}_W\left(\mathbf{Concat}\left(\mathbf{MeanAgg}([\mathbf{h}_i]_{i=1}^d), \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}_{\tilde{\pi}}})\right)\right) \\
&= \mathbf{MLP}_W\left(\mathbf{Concat}\left(\mathbf{MeanAgg}([\mathbf{h}_i]_{i=1}^d), \mathbf{MeanAgg}(\tilde{V}_{\mathcal{S}})\right)\right) \\
&= g_{NV}([\mathbf{h}_i]_{i=1}^d, \tilde{V}_{\mathcal{S}}) = \hat{\sigma}^2.
\end{aligned}$$

■

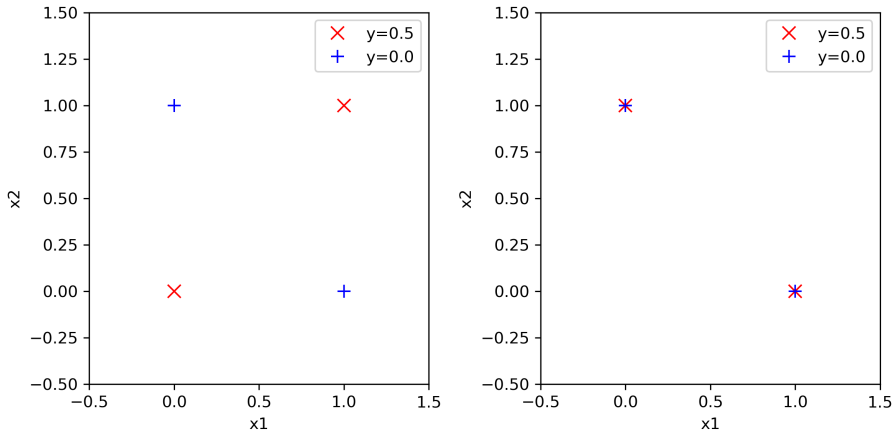

Figure 8: In this figure, we see two datasets with four datapoints each. Both datasets would be treated equivalently by the dataset encoder if it would be invariant to a permutation π of a single dimension-wise input sequence $\mathcal{H}^{(i)} = [(x_j^{(i)}, y_j)]_{j=1}^n$ for some $i \in \{1, \ldots, d\}$. For the second dataset, we swap the values of the first dimension of the red datapoints, thus doing a permutation of a single dimension-wise input sequence. Considering the y values, the second dataset has high noise, while the first dataset could also come from a low noise process. An invariant network would output the exact same GP parameters for both datasets, a clearly undesirable property.

C.2 REASON FOR CHANGE OF DATASET ENCODER

In this section, we analyze a property that motivates the change to the architecture of the dataset encoder compared to Liu et al. [2020]. We consider a permutation π of a single dimension-wise input sequence $\mathcal{H}^{(i)} = [(x_j^{(i)}, y_j)]_{j=1}^n$ for some $i \in \{1, \ldots, d\}$. Our architecture is in general not invariant to such a permutation, in contrast to the dataset encoder in Liu et al. [2020] which would output the same kernel parameters also after a permutation of a single dimension-wise input sequence. In Figure 8 we show two datasets that would be treated exactly as the same dataset in case such an invariance exists - the first dataset might come from a low noise process while the second dataset clearly contains large noise. Thus, assigning the same GP parameters to both datasets is undesirable in this case. Our trained amortization network, equipped with an SE kernel in each dimension, predicts for the left dataset a noise value of $\hat{\sigma}_{\mathcal{D}_1} = 0.18$ and for the right dataset $\hat{\sigma}_{\mathcal{D}_2} = 0.213$. This shows that our network is not invariant to such a permutation and also assigns higher noise to the second dataset.

References

Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/lee19d.html.

Sulin Liu, Xingyuan Sun, Peter J Ramadge, and Ryan P Adams. Task-agnostic amortized inference of gaussian process hyperparameters. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21440–21452. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/f52db9f7c0ae7017ee41f63c2a7353bc-Paper.pdf.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017. URL http://jmlr.org/papers/v18/16-537.html.

Aki Rehn. Amortized bayesian inference of gaussian process hyperparameters. *Master Thesis*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.