Amortized Inference for Gaussian Process Hyperparameters of Structured Kernels

Abstract

Learning the kernel parameters for Gaussian processes is often the computational bottleneck in applications such as online learning, Bayesian optimization, or active learning. Amortizing parameter inference over different datasets is a promising approach to dramatically speed up training time. However, existing methods restrict the amortized inference procedure to a fixed kernel structure. The amortization network must be redesigned manually and trained again in case a different kernel is employed, which leads to a large overhead in design time and training time. We propose amortizing kernel parameter inference over a complete *kernel-structure-family* rather than a fixed kernel structure. We do that via defining an amortization network over pairs of datasets and kernel structures. This enables fast kernel inference for each element in the kernel family without retraining the amortization network. As a by-product, our amortization network is able to do fast ensembling over kernel structures. In our experiments, we show drastically reduced inference time combined with competitive test performance for a large set of kernels and datasets.

1 INTRODUCTION

Gaussian processes (GPs) are an important class of models that can be used in a wide range of tasks such as Bayesian optimization [Snoek et al., 2012, Korovina et al., 2020, Bitzer et al., 2022], active learning [Zimmer et al., 2018, Li et al., 2022], or regression [Duvenaud et al., 2013]. Introducing an inductive bias for GPs is achieved by specifying the kernel structure. For example, smoothness, nonstationarity or periodicity can be induced very elegantly by configuring the corresponding kernel.

Learning the kernel parameters is often a major computational bottleneck and is usually done via marginal likelihood maximization, also called *Type-2-ML*, or via evidence-lower-bound maximization (ELBO) in sparse GP's [Hensman et al., 2015]. These methods often require hundreds of optimization steps to learn the kernel parameters. In Liu et al. [2020b], this problem is circumvented by using amortized inference [Kingma and Welling, 2014, Rezende et al., 2014] to predict the kernel parameters via a neural network in one step. This leads to a dramatic reduction in inference time for medium-sized datasets.

However, the method of Liu et al. [2020b] defines the amortization only for a fixed kernel structure. Importantly, specifying the kernel structure is a crucial design choice for GP's and is often used to induce prior knowledge of the task at hand such as smoothness, nonstationarity, linearity or periodicity. In case a different kernel should be used the network in Liu et al. [2020b] would need to be redesigned and retrained, which is a time-consuming and costly task, considering the vast space of possible kernel structures. We therefore propose amortizing the kernel inference for GP's over the combined space of kernel structures and datasets.

We define an amortization neural network that gets as input a complete dataset and a symbolical description of the kernel, based on the kernel grammar [Duvenaud et al., 2013], and outputs the learned kernel parameters. We design the neural network explicitly to cope with the natural invariances of the underlying spaces. Here, we make use of the transformer architecture [Vaswani et al., 2017] and its equivariance properties [Lee et al., 2019]. We empirically show that our method leads to a drastic decrease in inference time, while delivering competitive predictive results on real-world datasets. Additionally, we illustrate the generality of our method via defining a fast ensembling over kernel structures that explicitly leverages our architecture. In short, our contributions are

1. We construct an amortization neural network that is defined on the combined space of kernel structures

Submitted to the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023). **To be used for reviewing only**.

and datasets. We explicitly incorporate invariances and equivariances of the underlying spaces in the architecture.

2. We empirically demonstrate the effectiveness of the amortization over several simulated and real world datasets and kernel structures.
3. We show the generality of our approach by enabling a fast ensembling over kernel structures.

2 BACKGROUND

In the following section we give necessary background information about Gaussian processes with a focus on the hyperparameter optimization involved. We start by introducing the standard technique to hyperparameter inference for GP's and will then consider amortized inference over multiple datasets, as proposed in Liu et al. [2020b]. Finally, we consider a broad kernel space over which we will define our proposed *combined* amortization scheme.

Gaussian processes. Let $\mathcal{X} \subset \mathbb{R}^d$ be the input space for some $d \in \mathbb{N}$. A Gaussian Process defines a distribution over mappings $f : \mathcal{X} \to \mathbb{R}$ and is fully specified via a positive-definite kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and a mean function $m : \mathcal{X} \to \mathbb{R}$. It is characterized by the property, that, for any finite selection of input points $\mathbf{X} = \{x_1, \dots, x_n\} \subset \mathcal{X}$ and any $n \in \mathbb{N}$, the collection of function evaluations $(f(x_1), \dots, f(x_n))^\intercal$ is multivariate Gaussian with mean $m(\mathbf{X}) := (m(x_1), \dots, m(x_n))^\intercal$ and covariance matrix $k(\mathbf{X}, \mathbf{X}) := [k(x_i, x_j)]_{i,j=1,\dots,n}$. We write $f \sim \mathcal{GP}(m, k)$ to denote that the function f is drawn from a Gaussian process.

Let $\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^{d+1}, i = 1, \dots, n\}$ be a dataset for which we want to do regression. The typical modeling assumption for Gaussian process regression presumes a latent function $f \sim \mathcal{GP}(m, k_\theta)$ with a Gaussian likelihood, thus, $y_i = f(x_i) + \epsilon_i$ with $\epsilon_i \in \mathcal{N}(0, \sigma^2)$. The kernel is parameterized with $\theta \in \Theta \subset \mathbb{R}^p$ and the complete parameter vector of the GP, including the likelihood variance σ^2, is given with $\phi = (\theta, \sigma^2) \in \Phi \subset \mathbb{R}^{p+1}$. An important property of this model is that the marginal likelihood, marginalized over the latent function f can be computed analytically:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \theta, \sigma^2) &= \int p(\mathbf{y}|f, \mathbf{X}, \sigma^2) p(f|\mathbf{X}, \theta) df \qquad (1) \\ &= \mathcal{N}(\mathbf{y}; m(\mathbf{X}), k_\theta(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}). \end{aligned}$$

Inference of the inner parameters, which in this case is the infinite dimensional function f, can be done analytically. For the outer hyperparameters θ and σ^2 the classical way of training is maximizing the marginal-likelihood (1), also called *type-2 maximum likelihood*. Thus, we want to solve the following optimization problem

$$(\theta_*, \sigma_*^2) = \arg \max_{(\theta, \sigma^2) \in \Phi} \log p(\mathbf{y}|\mathbf{X}, \theta, \sigma^2) \qquad (2)$$

for a given dataset $\mathcal{D}$. The optimization problem is usually solved via gradient-based optimizers like Adam or L-BFGS. Each step in the optimizer requires a calculation of the marginal-likelihood, which scales cubically in N. Furthermore, several hundred optimization steps might be necessary to reach convergence and, depending on the kernel and dataset, multiple restarts are necessary as the optimization problem is non-convex and might end up in a local maxima. In the next section, we will consider an alternative approach of solving (2) that only requires one forward-pass through an amortization network.

Parameter amortization. Liu et al. [2020b] presented an alternative method for learning the GP hyperparameters based on amortizing the inference over multiple datasets. In this method, parameter inference for GP's reduces to a prediction via an amortization neural network. The amortization network $g_\psi : \mathscr{D} \mapsto \Theta$ with weights ψ is defined on the set of all datasets $\mathscr{D}$ meaning that for any $n \in \mathbb{N}$ and $d \in \mathbb{N}$ the dataset $\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^{d+1}, i = 1, \dots, n\}$ is part of the input set of the network, thus $\mathcal{D} \in \mathscr{D}$. The output-space is the parameter space Θ of the respective kernel for which amortized inference should be done. In case of Liu et al. [2020b], the Spectral Mixture Product (SMP) kernel is used, which consists of parameters $\theta_j = \{\{w_{m,j}\}_{m=1}^M, \{\mu_{m,j}\}_{m=1}^M, \{\sigma_{m,j}^2\}_{m=1}^M\}$ in the j-th dimension for some fixed M. The amortization network is designed via consecutive transformer blocks such that it can handle different input sizes and input dimensions of the respective dataset. The network is trained on a dataset of datasets $\{\mathcal{D}^{(l)}\}_{l=1}^L \subset \mathscr{D}$ via minimization of the mean average negative marginal likelihood of the datasets. After training, the network is used for one-shot prediction of the kernel parameters $\hat{\theta} = g_\psi(\mathcal{D}^*)$ on an unseen dataset $\mathcal{D}^*$. In Liu et al. [2020b], the kernel structure is fixed to the one of the SMP kernel. To use it with a different kernel, the network needs to be redesigned and retrained. For example, Rehn [2022] changed the architecture to cope with the RBF kernel. Our goal is to do amortized inference over the *combined space of datasets and kernel structures*, which drastically reduces redesign and retrain time and enables fast inference for many existing kernel structures via only one neural network.

Our amortization network consists of a dataset encoder, that is inspired by the architecture of Liu et al. [2020b] and a novel kernel encoder-decoder block that enables amortization over the combined input of kernel structure and dataset. Both blocks are designed to capture the natural invariances of the underlying structure.

Kernel space. Our goal is to define an amortization procedure over a family of kernels. To be more precise, we consider a family of *structural forms of kernels*. The basis for this kernel space is the kernel grammar presented in Duvenaud et al. [2013]. Here, each kernel structure is expressed

as a symbolic expression $\mathcal{S}$ made of base symbols $\mathcal{B}$. The base symbols might include simple elementary kernels like the Squared-Exponential kernel represented as the symbol SE, the linear kernel as LIN or the periodic kernel as PER. More complex expressions can be formed with multiplication and addition of base kernels/symbols. For example, one might construct a more complex structural form of a kernel via the expression $\mathrm{SE} \times \mathrm{LIN} + \mathrm{PER}$. Each expression $\mathcal{S}$ describes a structural form of a kernel - thus, the mathematical equation that governs the associated kernel k. The base kernels and therefore the combined expressions come with parameters θ and thus each expression has its own associated parameter space $\Theta_{\mathcal{S}}$.

The kernel grammar in Duvenaud et al. [2013] considers all possible algebraic expressions of the base kernels. We consider a subset of the kernel grammar that leads to a rich kernel space on the one hand and one that can be easily represented in a neural network on the other hand.

First, we define a set of base symbols $\mathcal{B}$ which consists of a set of elementary kernels like $\mathrm{SE}, \mathrm{LIN}$ and PER and its two-gram multiplications like $\mathrm{SE} \times \mathrm{LIN}$, $\mathrm{SE} \times \mathrm{PER}$ and $\mathrm{LIN} \times \mathrm{PER}$. This is a similar symbol set as used in Simpson et al. [2021]. All base symbols are defined on single dimensions, and we denote the concrete dimension via an index, e.g. SE_i for the Squared-Exponential kernel on dimension i and summarize the sets of indexed base symbols to $\mathcal{B}^{(i)}$. Our kernel space is then defined as an addition of base symbols within the dimension and a multiplication over dimensions: $\mathcal{S} = \prod_{i=1}^{d} \sum_{j=1}^{N_i} \mathcal{S}_{i,j}$ with $S_{i,j} \in \mathcal{B}^{(i)}$. For example, the following kernel would be part of the complete kernel space:

$$\overbrace{\underbrace{(\underbrace{\mathrm{SE}_1 \times \mathrm{LIN}_1}_{\text{symbol of } \mathcal{B}} + \mathrm{SE}_1)}_{\text{Addition within dimension}} \times (\mathrm{SE}_2 + \mathrm{PER}_2)}^{\text{Multiplication over dimensions}}. \tag{3}$$

We denote the complete kernel space with $\mathcal{K}$. This kernel space contains popular kernels like the ARD-RBF kernel with $\prod_{i=1}^{d} \mathrm{SE}_i$ or the d-dimensional periodic kernel with $\prod_{i=1}^{d} \mathrm{PER}_i$. Additionally, kernels that act differently on different dimensions are included in the kernel space.

Depending on the kernel expression, the parameter space $\Theta_{\mathcal{S}}$ can vary significantly in dimensionality. For example, the ARD-RBF kernel $\mathcal{S} = \prod_{i=1}^{d} \mathrm{SE}_i$ on dimension d contains one lengthscale and variance[1] parameter per dimension, such that $\Theta_{\mathcal{S}} \subset \mathbb{R}^{2d}$. The d-dimensional periodic kernel $\mathcal{S} = \prod_{i=1}^{d} \mathrm{PER}_i$ contains an additional feature specific period parameter such that $\Theta_{\mathcal{S}} \subset \mathbb{R}^{3d}$. Thus, being able to deal with different sizes of parameter spaces will be important for our proposed amortization scheme.

[1]We use the parameterization of the base kernels from the kernel grammar Duvenaud et al. [2013]. Here each base kernel in each dimension has its own variance.

3 METHOD

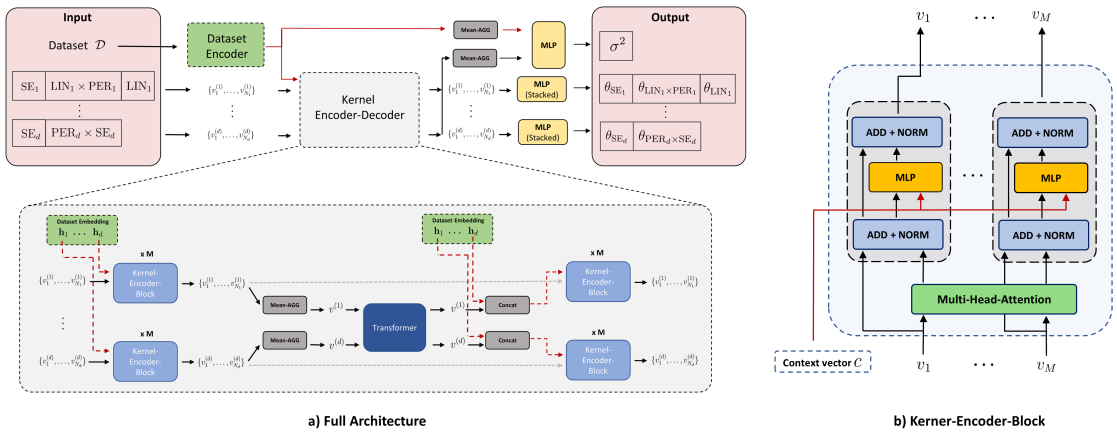
We propose amortizing the kernel inference for GP's over the combined space of datasets and kernel structures. This enables fast inference for many kernels, as well as fast ensembling. For this, we construct an amortization network $(\mathscr{D}, \mathcal{K}) \ni (\mathcal{D}, \mathcal{S}) \mapsto g_\psi(\mathcal{D}, \mathcal{S}) \in \Theta_{\mathcal{S}} \cup \mathbb{R}_+$ that maps from the combined space of datasets $\mathscr{D}$ and kernel structures $\mathcal{K}$ to the parameter space of the GP with respective kernel $\Theta_{\mathcal{S}} \cup \mathbb{R}_+$. The trained network is then used to one-shot predict GP parameters $(\hat{\theta}_{\mathcal{S}}, \hat{\sigma}^2) = g_\psi(\mathcal{D}^*, \mathcal{S})$ of the specified kernel structure $\mathcal{S}$ for an unseen dataset $\mathcal{D}^*$. We denote with ψ the (trainable) parameters of the amortization network. In the following subsections, we describe the architecture of the network and the learning procedure of g_ψ.

3.1 ARCHITECTURE

The model gets as input a dataset $\mathcal{D} = \{(x_i, y_i) \in \mathbb{R}^{d+1} | i = 1, \ldots, n\}$ where $n \in \mathbb{N}$ is the number of datapoints and $d \in \mathbb{N}$ is the number of input dimensions. Additionally, the model receives the kernel expression $\mathcal{S}$ as input. As $\mathcal{S}$ is a multiplication over dimension-wise sub-expressions $\mathcal{S}_i$, we can represent the expression $\mathcal{S}$ as a sequence of its sub-expressions $[\mathcal{S}_1, \ldots, \mathcal{S}_d]$. Similarly, we can decompose the expressions in each dimension into a sequence of base symbols. Thus, we represent/store the expression as a sequence of sequences of base symbols $\left[[B_1^{(i)}, \ldots, B_{N_i}^{(i)}] | i = 1, \ldots, d; B_j^{(i)} \in \mathcal{B}\right]$. We encode each base symbol via one-hot-encoding such that the sub-expression in each dimension $\mathcal{S}_i$ is represented via a sequence of vectors $\mathcal{V}_i = [v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ with $v_j^{(i)} \in \mathbb{R}^{|\mathcal{B}|}$. The whole expression is then represented via $\mathcal{V}_{\mathcal{S}} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$.

Our architecture consists of three main parts, the dataset encoder g_D, that takes as input the dataset $\mathcal{D}$ and returns a sequence of dimension-wise embeddings $\mathbf{h}_{\mathcal{D}} = [\mathbf{h}_1, \ldots, \mathbf{h}_d]$, a kernel encoder-decoder $g_k(\mathbf{h}_{\mathcal{D}}, \mathcal{V}_{\mathcal{S}})$ that gets as input a sequence of sequences of encoded base symbols $[\mathcal{V}_1, \ldots, \mathcal{V}_d]$ and the dataset embeddings $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ and outputs a transformed sequence of sequences of kernel embeddings $[\mathcal{V}_1, \ldots, \mathcal{V}_d]$ and finally an output layer that maps the kernel embeddings to the respective parameter space of the base kernels.

We design the different parts of the architecture to cope with several symmetries. Similar to Liu et al. [2020b], our network is permutation invariant to the shuffling of the dataset elements. Furthermore, the dataset encoder and the kernel encoder-decoder are equivariant to the permutation of input dimensions. Incorporating these symmetries enables generalization to datasets with sizes and input dimensions that were not present in the training phase. Lastly, the final output is invariant to a shuffling of the base symbols in each



a) Full Architecture

b) Kerner-Encoder-Block

Figure 1: a) Illustration of the full amortization network, which gets as input the dataset $\mathcal{D}$ and a kernel expression $\mathcal{S}$ decomposed in a sequence of sequences of base symbols and outputs the kernel hyperparameters θ in the respective parameter space Θ_S. b) The main layer used in the Kernel-Encoder-Decoder. It gets as input a sequence of vectors and a context vector and outputs a transformed sequence of vectors. The context vector enters the MLP layer.

dimension, which is important as the sequences describe additions. The prediction of the network is thus not dependent on the order in which the additions are represented in $\mathcal{S}$.

We present the single parts of the architecture in the following paragraphs and an overview in Figure 1 a).

Dataset-Encoder. The dataset encoder takes as input the dataset $\mathcal{D}$ and returns a sequence of dimension embeddings $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$. We utilize the encoder part of the Transformer architecture without positional encoding [Vaswani et al., 2017, Lee et al., 2019] in multiple parts of our architecture and refer to it as a **Transformer** block. Each block maps a sequence of vectors to a sequence of transformed vectors $[a_1, \ldots, a_1] \leftarrow \textbf{Transformer}([a_1, \ldots, a_1])$ using multiple multi-head-self-attention layers [Vaswani et al., 2017]. We consecutively apply **Transformer** blocks to different hidden embeddings to construct a sequence of dimension embeddings $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$ with $\mathbf{h}_i \in \mathbb{R}^{2h}$. It involves the following steps:

1. The dataset is divided into dimension-wise sequences $[(x_j^{(i)}, y_j)]_{j=1}^n$ where $x_j^{(i)}$ is the i-th dimension of point x_j.
2. Each sequence $[(x_j^{(i)}, y_j)]_{j=1}^n$ is mapped element-wise via a linear layer to construct a sequence of embeddings per dimension $[h_1^{(i)}, \ldots, h_n^{(i)}]$ with $h_j^{(i)} \in \mathbb{R}^h$.
3. Each sequence $[h_j^{(i)}]_{j=1}^n, i = 1, \ldots, d$ is given to a **Transformer** block (shared over the d sequences) that outputs a transformed sequence $[h_j^{(i)}]_{j=1}^n$.
4. So far, each datapoint was only able to attend to other datapoints inside its dimension. In a next step, we create per datapoint embeddings via mean aggregation $h_j = \textbf{MeanAGG}([h_j^{(1)}, \ldots, h_j^{(d)}])$ leading to a sequence of datapoint embeddings $[h_j]_{j=1}^n$ with $h_j \in \mathbb{R}^h$.
5. The datapoint embeddings are put into a **Transformer** block to form a transformed sequence of datapoint embeddings $[h_1, \ldots, h_n]$.
6. In order to construct embeddings per dimension again, we append the datapoint embedding to the sequences of embeddings of step 3, thus $h_j^{(i)} \leftarrow \textbf{Concat}(h_j, h_j^{(i)})$ which results in sequences $[h_1^{(i)}, \ldots, h_n^{(i)}]$ with $h_j^{(i)} \in \mathbb{R}^{2h}$.
7. Each updated sequence $[h_1^{(i)}, \ldots, h_n^{(i)}], i = 1, \ldots, d$ is again given to a (shared) **Transformer** block that outputs a transformed sequence $[h_1^{(i)}, \ldots, h_n^{(i)}]$.
8. In order to get dimension embeddings, we aggregate the sequence via mean aggregation to $\mathbf{h}_i = \textbf{MeanAGG}([h_1^{(i)}, \ldots, h_n^{(i)}])$ leading to a sequence of dimension embeddings $[\mathbf{h}_i]_{i=1}^d$ with $\mathbf{h}_i \in \mathbb{R}^{2h}$.
9. The sequence of dimension embeddings $[\mathbf{h}_i]_{i=1}^d$ is again put through a **Transformer** block to get a sequence of dimension embeddings $[\mathbf{h}_i]_{i=1}^d$ that contains shared information across dimensions.

The encoder is very similar to the one in Liu et al. [2020b]. The only difference are the steps 3. to 6. We incorporate these steps to prevent the permutation invariance to shuffling in the seperated sequences $[(x_j^{(i)}, y_j)]_{j=1}^n$, which can lead to pathologies (see Appendix C). Our encoder is still invariant to a shuffling of the dataset elements and permutation equivariant to a shuffling of the input dimensions. We give rigorous proofs in Appendix C.

Kernel-Encoder-Decoder. The kernel encoder-decoder block is meant to translate the structure of the kernel given

through $\mathcal{V}_{\mathcal{S}} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$ into transformed embeddings $\mathcal{V}_{\mathcal{S}} = [\mathcal{V}_1, \ldots, \mathcal{V}_d]$ that incorporates the information of the dataset and the global information about the kernel structure. These embeddings can be used to predict kernel parameters of the base symbols that are associated with each embedding element $v_j^{(i)}$. We call the block encoder-decoder, as the global structure of the expression $\mathcal{S}$ needs to be encoded and then, using the information of the global structure and the dataset, each embedding of base symbols $v_j^{(i)}$ needs to be decoded into a vector that contains information about the kernel parameters of the base symbol/kernel.

The main building block is the **Kernel-Encoder-Block** as shown in Figure 1 b). This block maps a context vector $c \in \mathbb{R}^l$ and a sequence of vectors $[v_1, \ldots, v_M]$ with $v_j \in \mathbb{R}^h$ to a transformed sequence of vectors $[v_1, \ldots, v_M]$ with $v_j \in \mathbb{R}^h$. It first applies self-attention to the sequence, followed by a concatenation of the context vector to the input of the element-wise multi-layer-perceptron (MLP) layer. Given a context vector c, this layer is permutation-equivariant to a shuffling of the sequence.

The **Kernel Encoder-Decoder** consists of the following steps:

1. Each sequence of base kernel embeddings per dimension $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ is given to a (shared) stack of **Kernel-Encoder-Block** layers with the dataset embedding of the respective dimension $\mathbf{h}_i$ as context vector. The output is a transformed sequence $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$.
2. So far, only the information of the base kernels along one dimension is shared. Thus, we form dimension-wise kernel embeddings via mean aggregation to $\mathbf{v}_i = \mathbf{MeanAGG}([v_1^{(i)}, \ldots, v_n^{(i)}])$ leading to the sequence $[\mathbf{v}_1, \ldots, \mathbf{v}_d]$ with $\mathbf{v}_i \in \mathbb{R}^h$. Each element in the sequence is an embedding of the kernel inside dimension i.
3. To form a shared (global) representation of the kernel, we put the sequence of dimension-wise kernel embeddings to a **Transformer** block and receive a transformed sequence of dimension-wise kernel embeddings $[\mathbf{v}_1, \ldots, \mathbf{v}_d]$.
4. Finally, we apply again a (shared) stack of **Kernel-Encoder-Block** layers to the base kernel embeddings per dimension $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ with extended context vector $c_i = \mathbf{Concat}(\mathbf{h}_i, \mathbf{v}_i)$ such that the shared kernel representation as well as the dataset encoding are part of the context. This gives us the final sequence of kernel embeddings per dimension $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$.

Output-Layer. The final part of the architecture is the prediction head for the kernel parameters. For a given expression $\mathcal{S}$, this layer has as output space the corresponding parameters space $\Theta_{\mathcal{S}}$. It gets as input the kernel embeddings per dimension $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$ of the **Kernel Encoder-Decoder**. Each embedding $v_j^{(i)}$ is associated with one base-symbol $B_j^{(i)}$ and each base-symbol has its own, fixed parameter space $\Theta_{B_j^{(i)}}$, like for example $\Theta_{SE} \subset \mathbb{R}^2$. We therefore realize the final mapping to $\Theta_{\mathcal{S}}$ via mapping each symbol related embedding $v_j^{(i)} \in \mathbb{R}^h$ to the respective parameter space of the base-symbol. We do this via separate **MLP** blocks for each base-symbol (more details in Appendix C).

In order to get an end-to-end amortization network, we also need a prediction for the likelihood variance. The variance depends on the kernel choice and the dataset. Thus, we form global embeddings of the kernel, via mean aggregation of all kernel embeddings per dimension $[v_1^{(i)}, \ldots, v_{N_i}^{(i)}]$, and of the dataset via mean aggregation of the dimension embeddings $[\mathbf{h}_1, \ldots, \mathbf{h}_d]$. We concatenate both global embeddings and use an **MLP** block to predict the noise variance.

In summary, our network predicts the kernel parameters and noise variance for a given dataset $\mathcal{D}$ and expression $\mathcal{S}$:

$$(\hat{\theta}_{\mathcal{S}}, \hat{\sigma}^2) = g_\psi(\mathcal{D}, \mathcal{S}). \tag{4}$$

It accounts for the natural invariances/equivariances of the respective spaces, which we elaborate in Appendix C.

3.2 TRAINING PROCEDURE

Our objective is to train a general purpose prediction network $g(\mathcal{D}, \mathcal{S})$ that can act on, in principle, all (medium-sized) datasets and all expressions $\mathcal{S}$ in the described kernel space. To conquer this challenge with enough data, we train our network purely on simulated datasets. We reflect the variety of inputs via sampling pairs $(\mathcal{D}_l, \mathcal{S}_l)$ from a broad distributions $(\mathcal{D}_l, \mathcal{S}_l) \sim p(\mathcal{D}, \mathcal{S})$. Given a dataset of sampled dataset-kernel pairs $\{(\mathcal{D}_l, \mathcal{S}_l)\}_{l=1}^L$, we utilize the average mean negative marginal-likelihood

$$\begin{aligned} &\mathcal{L}(\psi, \{(\mathcal{D}_l, \mathcal{S}_l)\}_{l=1}^L) \\ &= -\frac{1}{L} \sum_{l=1}^{L} \frac{1}{|\mathcal{D}_l|} \log p\left(\mathbf{y}_l \middle| \mathbf{X}_l, (\theta_l, \sigma_l^2) = g_\psi(\mathcal{D}_l, \mathcal{S}_l) \right) \end{aligned} \tag{5}$$

as loss-function. This reflects our goal to train a network that resembles the marginal-likelihood optimization of the kernel hyperparameters for a given kernel structure and a given dataset.

Sampling distribution. We sample $(\mathcal{D}, \mathcal{S})$ using the following scheme (we give a sketch here, details on the utilized distributions/priors can be found in Appendix A). First, we draw the number of input-dimensions d and datapoints n. Given d we draw a kernel expression $\mathcal{S}$, where we draw the subexpressions $\mathcal{S}_i$ independently of each other. Each subexpression can have a different number of base symbols. Each base-symbol/base-kernel comes with a prior on its hyperparameters. In order to generate a dataset $\mathcal{D}$ that stems

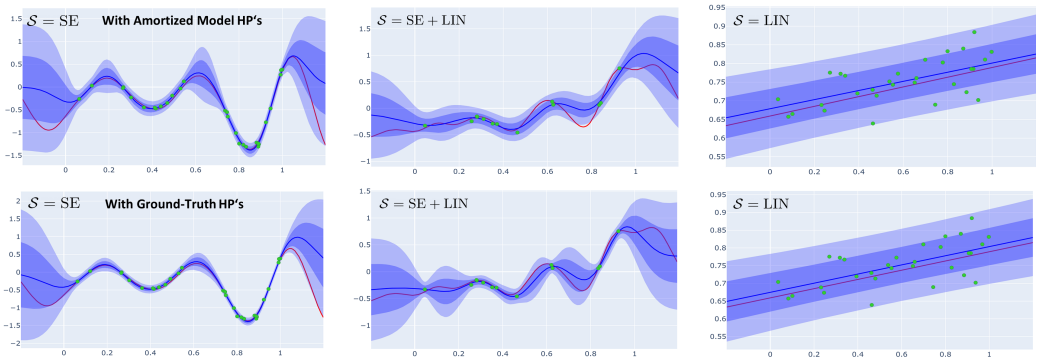


Figure 2: Illustration of the predictive distributions. Each column contains a ground truth function (red-line) drawn from a GP with kernel $\mathcal{S}$ and hyperparameter $\phi^*_{\mathcal{S}}$. Noisy datapoints from the ground truth function are shown in green. The upper row shows the resulting predictive distribution with predicted GP parameters $\hat{\phi}_{\mathcal{S}} = g_\psi(\mathcal{D}, \mathcal{S})$ and the lower row shows the predictive distribution with ground truth hyperparameter $\phi^*_{\mathcal{S}}$.

from the induced prior in function space of $\mathcal{S}$, we sample from the hyperparameter prior $\theta \sim p_{\mathcal{S}}(\theta)$ with $\theta \in \Theta_{\mathcal{S}}$. Next, we draw the input set $\mathbf{X} = \{x_1, \ldots, x_n\}$ uniformly from $[0, 1]^d$. Finally, we draw the observations from the GP via $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, k_{\mathcal{S},\theta}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})$, where $\sigma^2 \sim p(\sigma^2)$.

When constructing the pair $(\mathcal{D}, \mathcal{S})$, we distinguish two modes. The first mode is that $\mathcal{D}$ is sampled from the induced prior of $\mathcal{S}$ - we refer to this mode as the *positive* sample. For the second mode, we sample $\mathcal{D}$ using a different expression $\tilde{\mathcal{S}}$ - we refer to this mode as the *negative* sample. The reason for these two modes is that we cannot assume that only datasets from the induced prior of $\mathcal{S}$ will be used as input to the prediction network. There will always be a misspecification of the kernel.

Training parameters. During training, we sample each batch $\{(\mathcal{D}_l, \mathcal{S}_l)\}_{l=1}^L$ of size L on-the-fly from the sampling distribution $p(\mathcal{D}, \mathcal{S})$. This enables processing a huge corpus of dataset-kernel pairs. We employ RAdam [Liu et al., 2020a] as optimizer with a constant lengthscale.

Noise variance fine tuning. The noise level is a crucial property of a dataset and determines the predictive performance significantly. We therefore do a dedicated fine-tuning phase after the initial training phase of minimizing the negative marginal-likelihood $\mathcal{L}(\psi, \{(\mathcal{D}_l, \mathcal{S}_l)\}_{l=1}^L)$ in (5). We do the fine-tuning via minimizing the extended loss

$$\alpha\mathcal{L}(\psi, \{(\mathcal{D}_l, \mathcal{S}_l)\}_{l=1}^L) + \beta\frac{1}{L}\|\sigma_k - \hat{\sigma}_l\|_2^2, \qquad (6)$$

where we additionally regularize the prediction of the noise-variance to be close to the known ground-truth noise-variance. Importantly, we only draw *positive* samples $(\mathcal{D}_l, \mathcal{S}_l)$. We call this step a fine-tuning step, as we only do it on significantly fewer datasets than in the first phase. We observe a major increase in robustness of the noise-prediction on real-world datasets.

4 RELATED WORK

Amortized inference with fixed kernel structure. Our method extends the work of Liu et al. [2020b] to enable amortization over the combined space of datasets and kernel structures. Compared to Liu et al. [2020b] our network is not restricted to a single kernel, meaning that practitioners can insert any kernel structure $\mathcal{S}$ from our space and can utilize the amortization network out of the box for parameter inference via $\hat{\phi}_{\mathcal{S}} = g_\psi(\mathcal{D}, \mathcal{S})$. From a software perspective, we can view the work of Liu et al. [2020b] as an emulation of the inference functionality of a typical GP framework for a fixed kernel through a large neural network. Our method enlarges this emulation further via rendering the kernel configurable directly in the neural network.

Amortized model selection. In Simpson et al. [2021] an amortized structure selection is proposed. Here, the kernel structure $\mathcal{S}$ itself is predicted as a sequence of tokens via an amortization network $g(\mathcal{D})$. After selecting $\mathcal{S}$ the hyperparameters of the kernel needs to be trained via maximum likelihood. Our method complements their method, as after taking care that the kernel spaces are identical, one might use our method to predict the kernel parameters of the selected kernel structure. This would amortize the full pipeline of kernel selection and hyperparameter optimization.

Kernel grammar. Our input space is based on the kernel grammar in Duvenaud et al. [2013]. The kernel grammar is part of a greater research line called the *Automatic Statistician* [Duvenaud et al., 2013, Lloyd et al., 2014, Bitzer et al., 2022], which tries to infer interpretable GP models and dataset description from data in an automatic way. Our work can be used to enhance the GP parameter inference for each GP representation in the search procedure.

Hypernetworks. Our method can be seen as a hypernetwork [Ha et al., 2017] for Gaussian process models. Usually,

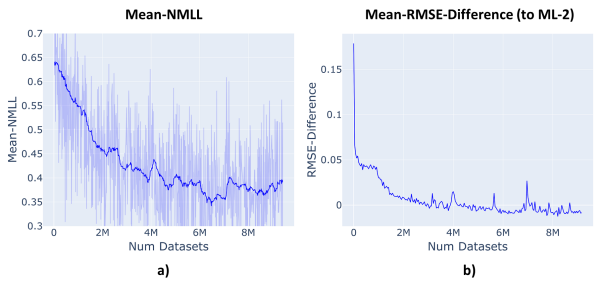


Figure 3: Training curves. In a) the training loss is shown for the first phase of the training procedure. In b) the average difference to the prediction of a type-2 ML GP on unseen simulated dataset-kernel pairs is shown over the training.

a hypernetwork predicts neural network weights from some sort of input. The input can be a hyperparameter [Lorraine and Duvenaud, 2018] or a latent representation of a layer [Ha et al., 2017]. Notably, Knyazev et al. [2021] proposed a hypernetwork to predict the neural network weights for a fixed dataset with a description of the architecture as input, which can be seen as a related task compared to ours in the neural network world. We note that our method predicts Gaussian process hyperparameters rather than neural network weights and amortizes over the combined space of datasets and kernel descriptions.

Prior-Fitted-Networks. In Müller et al. [2022] a method called *Prior-Fitted-Networks (PFN)* is proposed. Here a transformer is used to form an end-to-end prediction from a dataset $\mathcal{D}$ and a test point x^* to a predictive distribution $p(y^*|x^*, \mathcal{D})$ of a given prior. The difference to our approach is two-fold. Firstly, we predict a full GP via inferring its parameters. A PFN predicts only slices of the predictive distribution. Secondly, we render the prior configurable via making the kernel configurable. In this way, practitioners can include prior knowledge of the task at hand.

5 EXPERIMENTS

In the following section, we empirically analyze our amortization scheme on regression benchmarks and compare against common methods to do GP inference. First, we illustrate the prediction capabilities on toy datasets. Subsequently, we analyze the learning behavior and the performance on real-world datasets. In the last subsection, we analyze ensembling as a possible plug-and-play extension of our method.

Experimental setting. We train our amortization network on a stream of mini-batches of size $L = 128$ for a total of 9 million dataset-kernel pairs $(\mathcal{D}_l, \mathcal{S}_l)$ in the initial training phase [shown in Figure 3 a)]. We continue the training with the noise-variance fine-tuning phase, which is performed over 200.000 dataset-kernel pairs. In both phases, we utilize

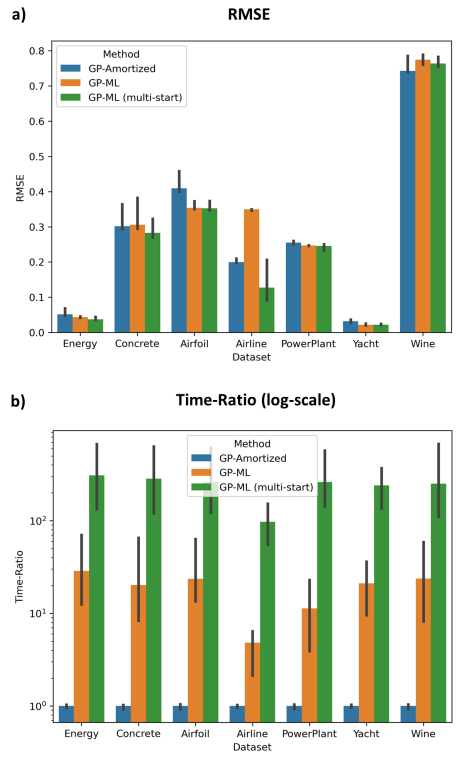


Figure 4: In a) the RMSE scores of each method are shown for held-out test datapoints. For each dataset several kernels $\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$ are evaluated where each bar shows the median RMSE value and the error-bars show the 20th and 80th percentiles of the RMSE scores of the different kernels. In b) the corresponding ratios of inference times to our method is shown in log-scale.

SE, LIN and PER and its 2-gram multiplications like, e.g. SE $\times$ LIN as base-symbols and simulate datasets of size $n \in [10, 250]$ and input dimension $d \in [1, 8]$. Further training details can be found in Appendix A.

Performance on simulated data. We evaluate the final amortization network g_ψ via its inference capabilities on unseen datasets $\tilde{\mathcal{D}}$. Each unseen dataset $\tilde{\mathcal{D}}$ is splitted into training $\tilde{\mathcal{D}}_{train}$ and test dataset $\tilde{\mathcal{D}}_{test}$ and we evaluate for a given kernel expression $\mathcal{S}$ the predictive performance of the final GP with predicted hyperparameters $\hat{\phi}_{\mathcal{S}} = g_\psi(\tilde{\mathcal{D}}_{train}, \mathcal{S})$ on $\tilde{\mathcal{D}}_{test}$. We give example predictive distributions on simulated datasets in Figure 2. We observe that our amortization network leads to accurate predictive distributions - notably only via evaluating a neural network to predict the kernel parameters. We add several more prediction plots in Appendix B, including plots with misspecified kernels, small datasets and more complex kernels.

Furthermore, we show the average difference in test-RMSE scores of 200 unseen, simulated dataset-kernel pairs $(\tilde{\mathcal{D}}_l, \mathcal{S}_l)$ in Figure 3 b). The difference is taken between a GP with

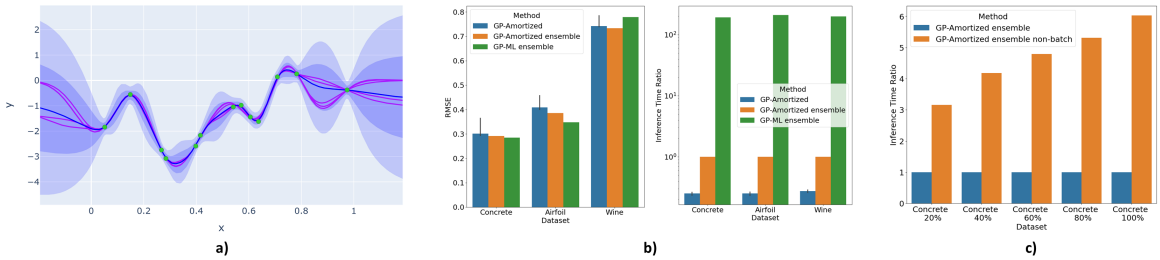


Figure 5: Fast ensembling. a) Predictive distribution resulting from a fast ensemble of five kernel structures (in blue). In red, we show the predictive means of the single GP's. b) RMSE and inference time ratios for the fast ensembling based on our amortization model (error bars of standard amortization are over the same 24 kernels that are in the ensemble). c) Ratios of inference time between batch and non-batch ensembling shown over different dataset sizes.

predicted hyperparameters from our method and a GP via learned hyperparameters (via Type-2-ML). We observe that the difference approaches zero over the initial training phase and in the end our method leads to even more precise predictions as the Type-2-ML inference. This illustrates the quality of the predicted hyperparameters.

Regression benchmarks. Our main evaluation considers seven real-world datasets. We split each dataset into a training and test set (we set $\mathrm{n}_{\mathrm{train}} = 500$ for all datasets except the smaller datasets Airline and Yacht, details in Appendix A) and evaluate the predictive performance over a set of $m = 24$ kernels $\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$ that are drawn randomly. We compare against the standard way of GP hyperparameter inference via Type-2-ML. We consider two versions of Type-2-ML once with a single run from initial parameters and once with 10 randomized restarts. For the Type-2-ML inferences, we maximize the marginal likelihood via Adam with $\mathrm{lr} = 0.1$ for 150 iterations and early-stop once the loss is converged.

In Figure 4 a), we show the resulting test-RMSE scores of the respective method. Here, the black bars correspond to percentiles of the RMSE scores of the set of different kernels $\{\mathcal{S}^{(i)}\}_{i=1}^{m}$. We observe that our methods leads to comparable predictive performance to Type-2-ML. Importantly, we observe in Figure 4 b) that our proposed method leads to a *major decrease in inference time for a diverse set of kernels* $\{\mathcal{S}^{(i)}\}_{i=1}^{m}$. This goes so far that for certain kernel structures our method is 800 times faster than Type-2-ML with restart.

Fast ensembling. Our method offers a general inference machine for GP hyperparameters over a structured kernel space. This can be utilized to construct ensembles in a computationally efficient way. We construct a Bayesian-Model-Average (BMA) over a set of kernel structures $\{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$, where we use the predicted marginal likelihood values as ensemble weights (see Appendix A). In 5 a) we show an example of the fast ensembling over five kernel structures. Importantly, we observe a high diversity in the predictive mean functions, which is a desirable property for an ensemble. In 5 b) we show the predictive performance of the fast ensemble over the set of 24 kernels and compare it to the range of predictive scores of the non-ensemble predictions (we show the three datasets that had the biggest diversity of RMSE scores over kernels). Furthermore, we compare against an ensemble, with Type-2 ML inferred GP parameters. Firstly, we observe that ensembling results in the expected performance gain over standard predictions and secondly, we see that our method offers a drastic speed-up compared to the conventional method.

Importantly, our architecture is particularly tangled to process this kind of ensembles over kernel structures, as it only needs to process the dataset-encoder once and can process multiple kernels in a batch through the kernel-encoder-decoder. This leads to an additional speed-up for inferring ensembles that can be seen in Figure 5 c).

Limitations. We note that predicting kernel parameters instead of optimizing them also has limitations. In particular, our method tends to favor conservative predictive distributions with broader prediction intervals out-of-data (see Appendix B). While this is often beneficial, it might not be desired, for example for extrapolation tasks with the periodic kernel where our method favors explaining the data via the lengthscale of the periodic kernel rather than the periodicity (see qualitative analysis in Appendix B).

6 CONCLUSION

In this work, we proposed an amortization scheme for the hyperparameters of Gaussian process models. The main novelty is to amortize over the combined space of datasets and kernel structures. Our proposed amortization network is explicitly designed to cope with the respective symmetries of this task. In our experiments, we show a drastic speed-up in inference time for diverse kernel structures. At the same time, we show that our method can predict kernel parameters that lead to competitive predictions on real world data.

References

Matthias Bitzer, Mona Meister, and Christoph Zimmer. Structural kernel search via bayesian optimization and symbolical optimal transport. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=-bLLVk-WRPy.

David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1166–1174, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable Variational Gaussian Process Classification. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 351–360, San Diego, California, USA, 09–12 May 2015. PMLR. URL https://proceedings.mlr.press/v38/hensman15.html.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero. Parameter prediction for unseen deep architectures. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=vqHak8NLk25.

Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3393–3403. PMLR, 26–28 Aug 2020.

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.

Cen-You Li, Barbara Rakitsch, and Christoph Zimmer. Safe active learning for multi-output gaussian processes. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 4512–4551. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/li22d.html.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020a. URL https://openreview.net/forum?id=rkgz2aEKDr.

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., 2020b. URL https://proceedings.neurips.cc/paper/2020/file/f52db9f7c0ae7017ee41f63c2a7353bc-Paper.pdf.

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1242–1250. AAAI Press, 2014.

Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *CoRR*, abs/1802.09419, 2018. URL http://arxiv.org/abs/1802.09419.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KSugKcbNf9.

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

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rezende14.html.

Fergus Simpson, Ian Davies, Vidhi Lalchand, Alessandro Vullo, Nicolas Durrande, and Carl Edward Rasmussen. Kernel identification through transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 10483–10495. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/56c3b2c6ea3a83aaeeff35eeb45d700d-Paper.pdf.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

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.

Christoph Zimmer, Mona Meister, and Duy Nguyen-Tuong. Safe active learning for time-series modeling with gaussian processes. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.