Supplementary Materials for the Submission: Trust Your Robots! Predictive Uncertainty Estimation of Neural Networks with Sparse Gaussian Processes

Jongseok Lee[1,2] **Jianxiang Feng**[1,3] **Matthias Humt**[1,3] **Marcus G. Müller**[1,4] **Rudolph Triebel**[1,3]
[1]Institute of Robotics and Mechatronics, German Aerospace Center (DLR)
[2]High Performance Humanoid Technologies, Karlsruhe Institute of Technology (KIT)
[3]Chair of Computer Vision and Artificial Intelligence, Technical University of Munich (TUM)
[4]Autonomous Systems Laboratory, ETH Zürich (ETHZ)

Abstract: This document is a supplementary material for the paper titled "Trust Your Robots! Predictive Uncertainty Estimation of Neural Networks with Sparse Gaussian Processes" [1]. This supplementary document provides (a) background materials, (b) derivation of theory and proofs (supporting the claims of the paper), (c) algorithmic overview, (d) implementation details of the experiments, and (e) further discussions and limitations of the method.

Video: https://www.youtube.com/watch?v=vu2TnDEqDRk

Code: https://github.com/DLR-RM/moegplib

Keywords: Robotic Introspection, Bayesian Deep Learning, Gaussian Processes

1 Overview

This document is the appendix to the submission "Trust Your Robots! Predictive Uncertainty Estimation of Neural Networks with Sparse Gaussian Processes". To recap the main paper, we focus on the problem of uncertainty estimation in deep neural network (DNN) predictions for robotic systems. As the robots are real-time systems with limited computations on-board (e.g. a Micro Aerial Vehicle - MAV), we aim to provide a solution, which is (i) sampling-free, i.e. a method that do not require combining several predictions of DNNs for a single test input, and (ii) improve the reliability of uncertainty estimates using the formulation of Gaussian Processes (GPs) with the Neural Tangent Kernel (NTK). To this end, we propose to combine DNNs with scalable GPs for the quantification of the predictive uncertainty in DNN predictions.

An overview of this document is as follows.

Background (section 2): Our submission builds on several concepts from different sub-fields of machine learning and robotics. These include Bayesian Neural Networks (section 2.1), Gaussian processes (section 2.2), the kernel methods for clustering (section 2.4), active learning (section 2.3) and model compression (section 2.5). For the interested readers, we provide a short background section.

Derivations of Theory and Proofs (section 3): In the main paper, we have briefly outlined the theory behind our approach - DNNs can be cast as a mixtures of Gaussian Process experts (MoE-GPs). In sections 3.1, 3.2 and 3.3, we provide our derivations, results, and their discussions, which constitute the main theoretic contributions of the paper. Meanwhile, sections 3.4 and 3.5 also describe some examples, on how the Neural Linear Models (NLMs) and their equivalent GPs can estimate the predictive uncertainty of neural networks.

Algorithmic overview (section 4): We provide both the learning and the prediction algorithms for an overview, which helps to understand the proposed methodology.

Implementation Details (section 5): We provide the implementation details about our experiments. Notably, the implementation details of the baselines are outlined in sections 5.1 and 5.2. We also

discuss about the pitfalls of the used measure - the Negative Log Likelihood (NLL). Then, we describe per experiments, the implementation details and additional results of the proposed method.

Discussions and Limitations (section 6): In our real-world experiments with a MAV and a Jetson TX2, we find several interesting points for discussion, and also limitations of our approach. Section 6 outlines these, and further discuss the potential future directions of research.

For the appendix to be self-contained, we re-introduce the notations and some of the concepts from the main paper. The code and the video are further provided as supplementary materials.

2 Background

In this section, we introduce several related concepts to our work. As a background material, we provide a short recap on the related concepts of machine learning and robotics.

2.1 Bayesian Neural Networks

Bayesian framework of a Neural Network (so called Bayesian Neural Network [2]) offers probabilistic interpretation of a DNN by inferring a distribution over the models weights or parameters. In other words, given an input-target pair $(\boldsymbol{x}, \boldsymbol{y})$ the posterior distribution over the space of parameters $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})$, which represents the model's uncertainty, are being modeled by assuming a prior distribution over the parameters $p(\boldsymbol{\theta})$:

$$p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}) = \frac{p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})}. \tag{1}$$

We note that equation 1 is an application of Bayes rule. Also, in Bayesian Neural Networks, a DNN is represented as probability distributions $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})$ as oppose to the single, most likely set of parameters (called the point estimates). The normalization constant in equation 1 is called the model evidence $p(\boldsymbol{y}|\boldsymbol{\theta})$. Typically conjugate priors do not exist for complex models such as DNNs. This is due to the non-linearity of DNNs with respect to the parameters. The integral of the model evidence is also not computational tractable as the size of the data and number of parameters grows:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}. \tag{2}$$

To tackle the intractability, several approximate Bayesian inference methods such as variational inference, sampling methods and Laplace approximation exists [2].

Once the posterior distribution over the weights have been estimated, the prediction of an output for a new input data $\boldsymbol{x}^*$ can be obtained by Bayesian Model Averaging or Full Bayesian Analysis by marginalizing the predictions with the posterior distributions $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})$:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{x}, \boldsymbol{y}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta})p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})d\boldsymbol{\theta}. \tag{3}$$

Unfortunately, we again do not have a closed form solution for the integrals in equation 3 due to the non-linearity in DNNs. To this end, many existing works resort to a technique called Monte Carlo integration, which is a sampling based method to evaluate the integrals:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{x}, \boldsymbol{y}) \approx \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}^*(\boldsymbol{x}^*, \boldsymbol{\theta}_t^s) \text{ for } \boldsymbol{\theta}_t^s \sim p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}).$$

This step generates the samples $\boldsymbol{\theta}_t^s$ from the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y})$, and combine the predictions of many sampled DNNs, in order to compute the predictive uncertainty.

We stress a drawback of this Monte Carlo integration for obtaining the predictive uncertainty from Bayesian Neural Networks. This is due to the sampling operations. Concretely, for a single test input $\boldsymbol{x}^*$, Bayesian Neural Networks require obtaining multiple predictions from the samples of the posterior distribution. As this can be slow at test-time, we attempt to exploit GP regression, where

the evaluation of the predictive uncertainty is analytic, and does not require sampling. Lastly, in our theory (section 3), we examine Bayesian Neural Networks with the Gaussian approximations to the posterior distributions, in order to derive a mathematical relationship between DNNs and MoE-GPs. We refer to the recent survey [3] for more comprehensive treatment.

2.2 Gaussian Processes

Gaussian processes [4], as one of the popular methods in Bayesian non-parametric modeling, provide a principled probabilistic framework for doing inference in function space. To recap this concept, we briefly introduce GPs for regression. Formally, a GP is defined as a collection of random variables indexed by an index set. Any finite combination of these random variables follows a joint Gaussian distribution, specified by a mean function $m(\cdot)$ and covariance function $\boldsymbol{k}_\theta(\cdot,\cdot)$, where θ denotes the hyperparameters.

Specifically, given a dataset $\mathcal{D} = \left\{\mathcal{X}, \mathcal{Y}\right\} = \left\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\right\}_{i=1}^{N}$, where $\boldsymbol{x}_i \in \mathbb{R}^D, \boldsymbol{y}_i \in \mathbb{R}^K$ are input and output, respectively. Firstly we assume a Gaussian likelihood of y_i given a latent function value f_i: $\boldsymbol{y}_i \sim \mathcal{N}(f_i, \sigma_0)$. According to the definition of GPs, when choosing the set of all possible inputs as the index set, we can obtain a joint Gaussian distribution over any finite number of the latent function values, e.g. on the given dataset $\mathcal{D}$: $\boldsymbol{f} = [f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_n)] \sim \mathcal{N}(m(\mathcal{X}), \boldsymbol{k}_\theta(\mathcal{X}, \mathcal{X}))$. In other word, the latent function over all possible inputs follow a GP specified by a mean and a covariance function:

$$f(\boldsymbol{x}_i) \sim \mathcal{GP}(m(\boldsymbol{x}_i), \boldsymbol{k}_\theta(\boldsymbol{x}_i, \boldsymbol{x}_j)) \tag{4}$$

In a Bayesian context, we want to capture the function space prior brought by the given dataset with a GP and then infer the posterior distribution of the function value, which can be employed to obtain the predictive distribution for a new test datum. Thanks to the conjugacy of Gaussian distributions, all of the aforementioned distributions can be computed analytically. While the GP prior can be learned on the given dataset $\mathcal{D}$ by performing maximum marginal log likelihood estimation over the hyper-parameters θ of the covariance function, the analytical form of the posterior distribution over the latent function for a new test datum $\boldsymbol{x}_*$ can be expressed as:

$$p(f(\boldsymbol{x}^*)|\mathcal{D}) = \mathcal{N}(\boldsymbol{k}_*^T(\boldsymbol{K} + \sigma_0\boldsymbol{I})^{-1}\boldsymbol{y}, \boldsymbol{k}_{**} - \boldsymbol{k}_*^T(\boldsymbol{K} + \sigma_0\boldsymbol{I})^{-1}\boldsymbol{k}_*) \tag{5}$$

where $\boldsymbol{K} = \boldsymbol{k}_\theta(\mathcal{X}, \mathcal{X})$, $\boldsymbol{k}_* = \boldsymbol{k}_\theta(\mathcal{X}, \boldsymbol{x}^*)$, $\boldsymbol{k}_{**} = \boldsymbol{k}_\theta(\boldsymbol{x}^*, \boldsymbol{x}^*)$, and $\boldsymbol{y}$ is assumed to be centered.

The equivalence between Bayesian linear regressions and GPs for regression from view point of weight space and function space can be easily recognized by restricting the $f(\boldsymbol{x}_i)$ to a linear function of the input features $f(\boldsymbol{x}_i) = \boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{w}$, where the weights $\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{I})$. Given an observed dataset $\mathcal{D}$, the posterior distribution of the weights can be computed analytically according to the Bayes rule, which is: $\boldsymbol{w}|\mathcal{D} \sim \mathcal{N}(\sigma_0^{-1}\boldsymbol{A}^{-1}\boldsymbol{\phi}(\mathcal{X})\boldsymbol{y}, \boldsymbol{A}^{-1})$, where $\boldsymbol{A} = \sigma_0^{-1}\boldsymbol{\phi}(\mathcal{X})\boldsymbol{\phi}(\mathcal{X})^T + \boldsymbol{I}$. Because $f(\boldsymbol{x}_i) = \boldsymbol{\phi}(\boldsymbol{x}_i)^T\boldsymbol{w}$, the posterior distribution of latent function for a new test datum $\boldsymbol{x}^*$ can be calculated analytically by marginalizing over the weights:

$$\begin{aligned} p(f(\boldsymbol{x}^*)|\mathcal{D}) &= \int p(f^*|\boldsymbol{x}^*, \boldsymbol{w}) p(\boldsymbol{w}|\mathcal{D}) d\boldsymbol{w} \\ &= \mathcal{N}(\sigma_0^{-1}\boldsymbol{\phi}(\boldsymbol{x}^*)^T\boldsymbol{A}^{-1}\boldsymbol{\phi}(\mathcal{X})\boldsymbol{y}, \boldsymbol{\phi}(\boldsymbol{x}^*)^T\boldsymbol{A}^{-1}\boldsymbol{\phi}(\boldsymbol{x}^*)) \end{aligned} \tag{6}$$

By comparing equation 5 and equation 6, with some algebraic operations [4], it can be shown that these two distributions are equivalent to each other when the covariance function $k_\theta(\cdot,\cdot)$ is set to $\boldsymbol{\phi}(\boldsymbol{x_i})^T\boldsymbol{\phi}(\boldsymbol{x_j})$. With this equivalence, we can infer the posterior function distribution of a Bayesian linear regression with basis function $\boldsymbol{\phi}(\cdot)$ by doing GP inference with the corresponding covariance function.

Besides its elegance of tractability and principledness, to put GPs into practice, there are still two main hurdles to overcome. The first one is its scalability to large dataset, and the second one is the expressivity of the covariance function. For the former, the inverse operation on the covariance matrix in equation 5 induces $O(n^3)$ computation and $O(n^2)$ storage complexities, where n is the size of dataset. This significantly limits the scalability of GPs on large scale datasets nowadays when disregarding other approximation techniques and sparse variants. When it comes to the second critical

issue, there has been some argument [5] stating that the local kernels such as radial basis function (RBF) kernel are struggling to discover effective representations for high dimensional data, which impedes the applicability on a variety of tasks today. As alternatives, neural tangent kernel (NTK)[6] and neural network kernel [7] are promising by complementing GPs with the strong representation learning capacity of DNNs. We also note that in mixtures of GP experts, the data noise term is in practice, learned independently for each data partition.

2.3 Active Learning

Contrasting to passive learning, active learning aims to utilize the training data as efficiently as possible, by actively selecting those data points that are the most informative for the learner. The motivation behind this paradigm arises in the some cases of supervised learning, where the data labeling process is time-consuming or resource-intensive. On the other hand, it can be beneficial to use as few data examples as possible to train the model if more is unnecessary, especially for those models whose efficiency is heavily affected by the amount of training data such as GPs. While there are different scenarios for active learning depending on the storage type of available data at hand, for brevity and conciseness, we focus on the pool-based scenario, where a pool of unlabeled data is available, to recap this concept. We refer to Settles [8] for more details about active learning.

To formulate the problem, we assume there are two datasets generated from the same data distribution, one is with labels $\mathcal{D} = \left\{\mathcal{X}, \mathcal{Y}\right\} = \left\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\right\}_{i=1}^{N}$, another one is without labels $\mathcal{U} = \{\mathcal{X}\}$, where $\boldsymbol{x}_i \in \mathbb{R}^D$, $\boldsymbol{y}_i \in \mathbb{R}^K$ and $|\mathcal{L}| \ll |\mathcal{U}|$. In supervised learning, we start from training a model for $f : \mathbb{R}^D \to \mathbb{R}^K$ on $\mathcal{L}$. Then the model queries the elements in $\mathcal{U}$ and ask an oracle to label them. These queried data points will be added to the $\mathcal{L}$ and used to re-train the model to improve the performance. In order to collect the most informative data points, which can improve the performance most after being incorporated into training, there is an acquisition function $a : \mathbb{R}^D \to \mathbb{R}$ used to weigh the utility score of each data point. To put it another way, the core problem of active learning lays in how to select the data point in the unlabeled pool set efficiently, which is:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x} \in \mathcal{U}} a(\boldsymbol{x}). \tag{7}$$

There is a diverse set of choices for the acquisition function according to the model used and computation resources available, raging from output/model uncertainty sampling, expected error reduction, data density weighting and etc [9]. The motivations from different query strategies differ in different ways to access the utility of a data point with respect to the model, e.g. while output/model uncertainty sampling focus more on information theoretic aspect, expected error reduction starts from insights of statistical analysis and density weighting takes into account the density of data.

In our case, uncertainty sampling is employed within our GP formulation. Since a GP is a probabilistic approach and is able to produce predictive distribution of the output of interest, which is a Gaussian. With the clear probabilistic interpretation, we directly use the output variance or entropy as the acquisition function to select the most informative data points from the pool $\mathcal{U}$, in order to achieve a satisfactory performance with as few data points as possible.

2.4 Kernel Methods for Clustering

To motivate kernelized *Principal Component Analysis* (PCA), recall the standard PCA setting. Given data $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ and its covariance matrix $\Sigma = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T$, PCA performs an eigendecomposition of Σ. Ranking the eigenvectors by the magnitude of their eigenvalues one can perform dimensionality reduction through the removal of low-variance dimensions and subsequent projection of $\mathcal{X}$ onto the remaining *principal components*.

In clustering, linearly separating $\bar{D}$ points in $D < \bar{D}$ dimension can be difficult while it is trivial in $D \geq N$ dimensions. Simply lifting $\boldsymbol{x}_i \in \mathbb{R}^D$ to $\mathbb{R}^{\bar{D}}$ using a function $\Phi : \mathbb{R}^D \to \mathbb{R}^{\bar{D}}$ creates linearly independent vectors with diagonal covariance on which PCA cannot be applied.

Instead, using the kernel $\boldsymbol{K} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j)$ one can perform PCA in the potentially infinite-dimensional $\Phi(\boldsymbol{x})$-space (e.g. using a Gaussian kernel) without ever actually having to evaluate the inner product (see *Kernel trick*). While this means one cannot obtain the eigenvectors

of $\boldsymbol{K}$ explicitly, the projection of all input points within $\mathcal{X}$, onto those eigenvectors can be obtained, making it easier to separate and cluster the data.

In relation to our paper, we employ the kernel PCA with NTK, and apply K-means clustering within the low dimensional space. This results in a gating function which divides and assigns data points to the individual GP experts.

2.5 Model Compression

Model compression techniques aim at reducing either or both of computational and memory requirements of DNNs. This gets especially relevant on resource constrained systems like mobile devices and robots.

There is a multitude of approaches of which we will briefly discuss four as identified by [10]. The first is parameter pruning which involves the identification of redundant or uncritical model parameters and their removal. The most common approach is to remove network weights with small magnitude after training [11]. Memory requirements are reduced proportional to the amount of pruned parameters but computational requirements and therefore latency during inference remains constant as the resulting parameter matrices, while becoming sparse, remain full-rank. Pruned networks can be re-trained after pruning to improve performance though convergence is delayed. Lower parameter counts help to combat overfitting and can increase generalization capabilities [12]. Instead of the removal of small weights, pruning can also be achieved through hashed binning of similar weights.

We then briefly discuss other class of methods for model compression. Contrary to pruning, parameter quantization retains all weights but reduces the memory footprint and often also computational cost on compatible hardware through half-precision floating point representations. While reductions from 32 to 16 bits [13] are common practice, 8 bits [14] and even binary representations are explored [15], though penalties in model performance seem to be unavoidable. Quantization can be employed during training or afterwards.

While pruning produces sparse but full-rank parameter matrices, low-rank factorization techniques [16, 17] try to produce dense low-rank representations through matrix factorization like *singular value decomposition* [18]. It is applied on a layer-wise basis and thus cannot exploit model-wide parameter redundancy. Constraints can already be imposed during model design and training. Structural weight matrices enforce symmetries in weight space [19] while compact filters reuse negated or rotated convolutional filters or replace large filters with concatenation of multiple smaller ones [20, 21]. Sparsity can be encouraged through regularization terms during training [22, 23].

Finally, knowledge distillation employs large high-performance teacher networks to train small student networks which try to mimic the teachers output distribution [24, 25]. Distillation from ensembles of teachers is also possible [26].

We note that many techniques can be used in tandem to achieve even stronger levels of compression [27]. As an example, the popular ResNet-50 model with a memory requirement of 95 MB and 3.8 billion FLOPs per inference can be compressed to 75% of its size and 50% of its computation time without loss in performance [10].

In relation to our method, we employ the pruning methods to reduce the size of NTK, which is a tangent to the Jacobians of DNNs. In particular, we consider the approaches in Microsoft Neural Network Intelligence library [1].

3 On theory: derivations, main results and proofs

In the main paper, we have outlined our main idea, i.e. projecting the NLMs to function space in order to obtain MoE-GPs, which divides the input space into smaller local regimes, where the individual GP experts learn and make predictions. In this section, we describe the theoretical contributions of our paper, which focuses on establishing the mathematical relationships between DNNs and MoE-GPs.

Before explaining our contributions, we stress the relevance of the theoretical work within the context of the paper. First, as we derive our theory in this section, several insights behind our method are revealed. These include various design choices that motivates the individual components of our

[1] Link: https://github.com/microsoft/nni

algorithm. Second, the theory provides a foundation to the related algorithms. Concretely, we argue that the proposed practical algorithm is only one way of exploiting the derived theory in practice. We hope that such theoretic foundations can help the community to build on our work for developing more effective uncertainty quantification techniques. Lastly, there has been several works that established the relationships between DNNs and GPs [28, 29, 30, 6]. In this sense, our work extends the prior work towards better understandings of DNNs.

We depict an overview of this section in figure 1, which summarizes our results and the derived proof paths. We build up the concepts in sections 3.1 and 3.2, and describe the main results in section 3.3. In addition, we provide derivations and discussions on how NLMs and MoE-GPs can estimate the predictive uncertainty of DNNs, in sections 3.4 and 3.5.

3.1 Mixtures of Neural Network Experts

Consider again a supervised learning task on a data set consisting of input-output pairs $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^D$, $\boldsymbol{y}_i \in \mathbb{R}^K$. Whenever possible, we drop the indices i for simplicity. Defining a neural network as a $\boldsymbol{\theta} \in \mathbb{R}^P$ parametrized function $f_{\boldsymbol{\theta}} : \mathbb{R}^D \to \mathbb{R}^K$ that maps the inputs $\mathcal{X}$ to the outputs $\mathcal{Y}$, learning seeks to obtain an empirical risk minimizer of the loss function, i.e. $\min_{\boldsymbol{\theta}} \quad \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) + \frac{\delta}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$ where δ is an L_2 regularizer. Here, a mini-batch $\mathcal{B} \subset \mathcal{D}$ is often used instead to find a local maximum-a-posteriori (MAP) solution $\hat{\boldsymbol{\theta}}$. Importantly, we assume a twice differentiable and strictly convex loss function $\mathcal{L}$ and piece-wise linear activations in $f_{\boldsymbol{\theta}}$. For example, this includes square loss or cross entropy loss with RELU, which are often used in modern DNNs. [2]

Now, a Mixture of Expert (MoE) consists of M experts defined as learners $\mathcal{F} = \{f_{\boldsymbol{\theta}_1}, \cdots, f_{\boldsymbol{\theta}_M}\}$, and a gating function $g : \mathbb{R}^D \to \Delta^{M-1}$ that maps any input $\boldsymbol{x}$ to $g(\boldsymbol{x}) = [g_1(\boldsymbol{x}), \cdots, g_M(\boldsymbol{x})]$ [31]. Each expert of the MoE learns and predicts within a subset of the input domain, and a gating function generates these subsets. We depict an example on the right side of figure 1 (Lemma 2.1), where the experts are GPs. Here, the gating function divides the input space $\mathcal{X}$ and assigns each datum $\boldsymbol{x}$ to an individual GP expert. We denote such models as MoE-GP [32, 33], and further also define a MoE-DNN, where each the experts are DNNs [34, 35] (refer to the left side of figure 1 in Lemma 2.1). Importantly, for MoE-DNNs, we assume a gating function: $g_m(\boldsymbol{x}) = 1$ in just one coordinate for each input [36] where the subscripts $m = 1, 2, \cdots M$ denote the m^{th} expert. For the partitioned data $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots \mathcal{D}_M\}$ the training minimizes an individual loss function: $\min_{\boldsymbol{\theta}_m} \frac{1}{|\mathcal{D}_m|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_m} \mathcal{L}(f_{\boldsymbol{\theta}_m}(\boldsymbol{x}), \boldsymbol{y}) + \frac{\delta}{2}\boldsymbol{\theta}_m^T\boldsymbol{\theta}_m$. For a clear exposition, we distinct the notation of ground truth $\boldsymbol{y}$ with $\mathbf{y} = \sum_{m=1}^M g_m(\boldsymbol{x}) f_{\boldsymbol{\theta}_m}(\boldsymbol{x})$ to represent the prediction in the remaining texts.

3.2 Laplace Approximation for Individual Experts

Next, consider a Bayesian treatment of MoE-DNNs with a prior on the parameters of the individual DNN experts $p(\boldsymbol{\theta}_m)$, and also their posterior distributions given the data $p(\boldsymbol{\theta}_m \mid \mathcal{D}_m)$ $\forall m$. Concretely, we employ the Laplace Approximation [37] to compute the posterior probabilities of the DNNs experts, which can be obtained by taking the second-order Taylor series expansion of the log posterior. Using a Gaussian prior with precision δ_m, we obtain:

$$\log p(\boldsymbol{\theta}_m \mid \mathcal{D}_m) \approx \log p(\hat{\boldsymbol{\theta}}_m \mid \mathcal{D}_m) + \frac{1}{2}(\boldsymbol{\theta}_m - \hat{\boldsymbol{\theta}}_m)^T(\boldsymbol{H}_m + \delta_m I)(\boldsymbol{\theta}_m - \hat{\boldsymbol{\theta}}_m),$$

where the first-order term vanishes as the gradient of the log posterior $\nabla \log p(\boldsymbol{\theta}_m \mid \mathcal{D}_m)$ is close to zero at $\hat{\boldsymbol{\theta}}_m$. Taking the exponential on both sides and approximating integrals by reverse engineering densities, the weight posterior is approximately a Gaussian with mean $\hat{\boldsymbol{\theta}}_m$ and covariance matrix $\boldsymbol{\Sigma}_m = (\boldsymbol{H}_m + \delta_m \boldsymbol{I})^{-1}$ where $\boldsymbol{H}_m$ is the Hessian of $\log p(\boldsymbol{\theta}_m \mid \mathcal{D}_m)$ [37, 2].

Unfortunately, working with $\boldsymbol{H}_m$ is difficult, as it is not in general positive semi-definite (PSD). Fortunately, when using standard loss and piece-wise linear activations, a good approximation of $\boldsymbol{H}_m$ is given by Gauss-Newton Matrix[38]:

[2] Our theories are for any DNN architecture that satisfies the given assumption. Only feedforward network shown in figure 1, but our approach also applies to convolution, and recurrent neural networks

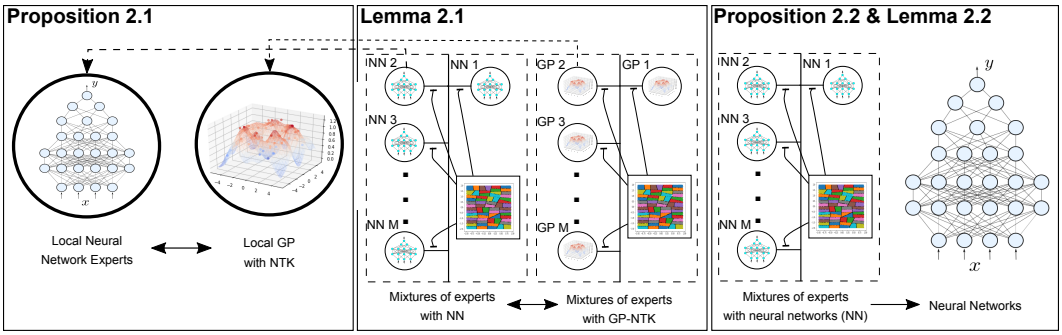

Figure 1: Illustration of the main results and their derivations. First we demonstrate a connection between the local neural network, and the linear models (or Gaussian Processes with Neural Tangent Kernel in function space view). Building upon, a connection between the mixtures of experts with neural networks and Gaussian Processes is established assuming a hard gating function. Lastly, when we assume that all neural experts are equivalent, the mixtures model can be thought as a single neural network, thereby, connecting the networks to the mixtures of experts with Gaussian Processes. Lemma 2.2 quantifies the influence of this assumption on GPs.

$$\boldsymbol{H}_m \approx \frac{1}{|\mathcal{D}_m|} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_m} \boldsymbol{J}_{f_m}(\boldsymbol{x})^T \boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y}) \boldsymbol{J}_{f_m}(\boldsymbol{x}), \quad \text{where}$$

$$\boldsymbol{J}_{f_m}(\boldsymbol{x}) = \frac{\partial f_{\boldsymbol{\theta}_m}(\boldsymbol{x})}{\partial \boldsymbol{\theta}_m^T}\ ,\ \boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y}) = \frac{\partial^2 \mathcal{L}(f_{\boldsymbol{\theta}_m}(\boldsymbol{x}),\boldsymbol{y})}{\partial f_{\boldsymbol{\theta}_m}(\boldsymbol{x})^T \partial f_{\boldsymbol{\theta}_m}(\boldsymbol{x})}\ ,\ \boldsymbol{R}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y}) = \frac{\partial \mathcal{L}(f_{\boldsymbol{\theta}_m}(\boldsymbol{x}),\boldsymbol{y})}{\partial f_{\boldsymbol{\theta}_m}(\boldsymbol{x})}, \tag{8}$$

are the Jacobian of $f_{\boldsymbol{\theta}_m}(\boldsymbol{x})$ wrt. parameters $\boldsymbol{\theta}_m$, the Hessian of loss function $\mathcal{L}(\boldsymbol{x},\boldsymbol{y})$ wrt. $f_{\boldsymbol{\theta}_m}(\boldsymbol{x})$, and a residual vector $\boldsymbol{R}_{\mathcal{L}_m} \in \mathbb{R}^K$ respectively. Notice that $\boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^{K\times K}$, $\boldsymbol{J}_{f_m}(\boldsymbol{x}) \in \mathbb{R}^{K\times P}$ and hence $\boldsymbol{H}_m \in \mathbb{R}^{P\times P}$. The result of these steps turn a mixture of DNN experts into a mixture of Bayesian Neural Network experts, where the parameters are represented using Gaussian distributions.

3.3 Neural Networks as Mixtures of Gaussian Process Experts

So far, we have defined MoE-DNNs with a hard gating function, and then showed how their parameters posterior can be approximated with a Gaussian rather than the point estimates $\hat{\boldsymbol{\theta}}_m$ for all m. This step results in a MoE model, where the experts are Bayesian Neural Networks. Having these essentials, we make the following statement (a specific instance of Khan et al. [29], which is in turn, based on the NLMs [39]).

Proposition. ***2.1 (Local Duality):*** *Let the posterior for each Deep Neural Network expert $p(\boldsymbol{\theta}_m|\mathcal{D}_m)$ approximated with a Gaussian distribution: $p(\boldsymbol{\theta}_m|\mathcal{D}_m) \sim \mathcal{N}(\hat{\boldsymbol{\theta}}_m, \boldsymbol{\Sigma}_m)$. Define a transformed data-set $\widetilde{\mathcal{D}}_m = \left\{\mathcal{X}_m, \widetilde{\mathcal{Y}}_m\right\}$ with pseudo output datum $\widetilde{\mathcal{Y}}_m = \{\widetilde{\boldsymbol{y}}_{m,1}, \widetilde{\boldsymbol{y}}_{m,2}, \cdots \widetilde{\boldsymbol{y}}_{m,N_m}\}$ such that $\widetilde{\boldsymbol{y}}_{m,j_m} := \boldsymbol{J}_{f_m}(\boldsymbol{x})\hat{\boldsymbol{\theta}}_m - \boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y})^{-1}\boldsymbol{R}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y})$ for $j_m = 1,2,\cdots,N_m$. Then, $\widetilde{\mathbf{y}}_m = \boldsymbol{J}_{f_m}(\boldsymbol{x})\boldsymbol{\theta}_m + \epsilon_m$ where $\epsilon_m \sim \mathcal{N}(\mathbf{0}, \boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x},\boldsymbol{y})^{-1})$ and $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \delta_m^{-1}\boldsymbol{I})$ is a neural linear model that has equivalent posterior $p(\boldsymbol{\theta}_m|\widetilde{\mathcal{D}}_m) \sim \mathcal{N}(\hat{\boldsymbol{\theta}}_m, \boldsymbol{\Sigma}_m)$ $\forall m$.*

Proof. The proof sketch is as follows. To show the local equivalence between the Laplace approximated DNN experts posterior and that of a neural linear model, we first reformulate the DNN experts posterior $p(\boldsymbol{\theta}_m|\mathcal{D}_m)$ with so-called the information form or natural parameters of Multivariate Normal Distribution (MND). Then, we demonstrate that the posterior of the individual m^{th} linear model can also be reformulated to match the given DNN experts posterior.

As said, we reformulate the generalized Gauss Newton (GGN) approximated posterior distribution for individual experts as:

$$\begin{aligned} p(\boldsymbol{\theta}_m|\mathcal{D}_m) &\sim \mathcal{N}(\hat{\boldsymbol{\theta}}_m, \boldsymbol{\Sigma}_m) \\ &= \frac{1}{\sqrt{(2\pi)^p|\boldsymbol{\Sigma}_m|}}\exp\left(-\frac{1}{2}(\boldsymbol{\theta}_m-\hat{\boldsymbol{\theta}}_m)^T\boldsymbol{\Sigma}_m^{-1}(\boldsymbol{\theta}_m-\hat{\boldsymbol{\theta}}_m)\right), \\ &= \exp\left(\boldsymbol{a}+\boldsymbol{\eta}^T\boldsymbol{\theta}_m-\frac{1}{2}\boldsymbol{\theta}_m^T\boldsymbol{\Lambda}\boldsymbol{\theta}_m\right). \end{aligned} \tag{9}$$

where $\boldsymbol{a}$ is the normalizing constant. In this formulation, MND is parameterized in the canonical form with the information vector $\boldsymbol{\eta} = \boldsymbol{\Sigma}_m^{-1}\hat{\boldsymbol{\theta}}_m$, and matrix $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}_m^{-1}$, instead of the mean and the covariance matrix $(\hat{\boldsymbol{\theta}}_m, \boldsymbol{\Sigma}_m)$. In the following, for better readability, we denote $\boldsymbol{J}_{f_m}(\boldsymbol{x}_{m,i})$, $\boldsymbol{R}_{\mathcal{L}_m}(\boldsymbol{x}_{m,i}, \boldsymbol{y}_{m,i})$, $\boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x}_{m,i}, \boldsymbol{y}_{m,i})$ by $\boldsymbol{J}_{f_m,i}$, $\boldsymbol{R}_{\mathcal{L}_{m,i}}$, $\boldsymbol{H}_{\mathcal{L}_m,i}$, respectively. The information vector and matrix can be expressed with the gradients and the Hessian:

$$\boldsymbol{\Sigma}_m^{-1} = \boldsymbol{H} + \delta_m\boldsymbol{I} \approx \sum_{i=1}^{N_m}\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i} + \delta_m\boldsymbol{I}. \tag{10}$$

$$\begin{aligned} \boldsymbol{\Sigma}_m^{-1}\hat{\boldsymbol{\theta}}_m &= -\left(\boldsymbol{J}_{f_m}{}^T\boldsymbol{R}_{\mathcal{L}_m} + \delta_m\hat{\boldsymbol{\theta}}_m\right) + \boldsymbol{\Sigma}_m^{-1}\hat{\boldsymbol{\theta}}_m, \\ &= \sum_{i=1}^{N_m}\left[-\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{R}_{\mathcal{L}_{m,i}} + \boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m\right], \end{aligned} \tag{11}$$

In above equations, we use the stationary assumption in Laplace Approximation, i.e. the loss is close to zero ($-\left(\boldsymbol{J}_{f_m}{}^T\boldsymbol{R}_{\mathcal{L}_m} + \delta_m\hat{\boldsymbol{\theta}}_m\right) \approx 0$), and added $\boldsymbol{\Sigma}_m^{-1}\hat{\boldsymbol{\theta}}_m$ on both sides. Again, we assume a Gaussian prior for the network parameters. Then, substituting this formulation of the information vector and matrix (equations 11, 10) into the given posterior distribution (equation 9) results in:

$$\begin{aligned} &p(\boldsymbol{\theta}_m|\mathcal{D}_m) \\ &\propto \exp(-\frac{1}{2}\boldsymbol{\theta}_m^T\left[\sum_{i=1}^{N_m}\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i} + \delta_m\boldsymbol{I}\right]\boldsymbol{\theta}_m + \boldsymbol{\theta}_m^T\sum_{i=1}^{N_m}\left[-\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{R}_{\mathcal{L}_{m,i}} + \boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m\right]), \\ &\propto \exp\left(-\frac{\delta_m}{2}\boldsymbol{\theta}_m^T\boldsymbol{\theta}_m\right)\prod_{i=1}^{N_m}\exp(-\frac{1}{2}\boldsymbol{\theta}_m^T\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{\theta}_m + \boldsymbol{\theta}_m^T\boldsymbol{J}_{f_m,i}{}^T\left[\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m - \boldsymbol{R}_{\mathcal{L}_m,i}\right]). \end{aligned} \tag{12}$$

Now, we show that the posterior distribution of the neural linear model can also be written as equation 12, which would complete the proof. Expressing the posterior $p(\boldsymbol{\theta}_m|\widetilde{\mathcal{D}})$:

$$\begin{aligned} &\propto \mathcal{N}(\boldsymbol{\theta}_m|0, \delta_m^{-1}\boldsymbol{I})\mathcal{N}(\widetilde{\boldsymbol{y}}_m|\boldsymbol{J}_{f_m}\boldsymbol{\theta}_m, \boldsymbol{H}_{\mathcal{L}_m}{}^{-1}), \\ &\propto \exp\left(-\frac{\delta_m}{2}\boldsymbol{\theta}_m^T\boldsymbol{\theta}_m\right)\prod_{i=1}^{N_m}\exp\left(-\frac{1}{2}(\widetilde{\boldsymbol{y}}_{m,i} - \boldsymbol{J}_{f_m,i}\boldsymbol{\theta}_m)^T\boldsymbol{H}_{\mathcal{L}_m,i}(\widetilde{\boldsymbol{y}}_{m,i} - \boldsymbol{J}_{f_m,i}\boldsymbol{\theta}_m)\right), \\ &\propto \exp\left(-\frac{\delta_m}{2}\boldsymbol{\theta}_m^T\boldsymbol{\theta}_m\right)\prod_{i=1}^{N_m}\exp\left(-\frac{1}{2}\widetilde{\boldsymbol{y}}_{m,i}^T\boldsymbol{H}_{\mathcal{L}_m,i}\widetilde{\boldsymbol{y}}_{m,i} + \boldsymbol{\theta}_m^T\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\widetilde{\boldsymbol{y}}_{m,i} - \frac{1}{2}\boldsymbol{\theta}_m^T\boldsymbol{J}_{f_m,i}{}^T\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}\boldsymbol{\theta}_m)\right). \end{aligned} \tag{13}$$

Now, we can substitute $\widetilde{\boldsymbol{y}}_{m,i} := \boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m - \boldsymbol{H}_{\mathcal{L}_m,i}{}^{-1}\boldsymbol{R}_{\mathcal{L}_m,i}$ into equation 13. We note that:

$$-\frac{1}{2}\widetilde{\boldsymbol{y}}_{m,i}^T\boldsymbol{H}_{\mathcal{L}_m,i}\widetilde{\boldsymbol{y}}_{m,i} = -\frac{1}{2}[\boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m - \boldsymbol{H}_{\mathcal{L}_m,i}{}^{-1}\boldsymbol{R}_{\mathcal{L}_m,i}]^T[\boldsymbol{H}_{\mathcal{L}_m,i}\boldsymbol{J}_{f_m,i}\hat{\boldsymbol{\theta}}_m - \boldsymbol{R}_{\mathcal{L}_m,i}], \tag{14}$$

is constant w.r.t the random variables $\boldsymbol{\theta}_m$. Furthermore, we also obtain:

$$\begin{aligned}\boldsymbol{\theta}_m^T \boldsymbol{J}_{f_m,i}{}^T \boldsymbol{H}_{\mathcal{L}_m,i} \widetilde{\boldsymbol{y}}_{m,i} &= \boldsymbol{\theta}_m^T \boldsymbol{J}_{f_m,i}{}^T \boldsymbol{H}_{\mathcal{L}_m,i} [\boldsymbol{J}_{f_m,i} \hat{\boldsymbol{\theta}}_m - \boldsymbol{H}_{\mathcal{L}_m,i}^{-1} \boldsymbol{R}_{\mathcal{L}_m,i}] \\ &= \boldsymbol{\theta}_m^T \boldsymbol{J}_{f_m,i}{}^T \Big[\boldsymbol{H}_{\mathcal{L}_m,i} \boldsymbol{J}_{f_m,i} \hat{\boldsymbol{\theta}}_m - \boldsymbol{R}_{\mathcal{L}_m,i}\Big].\end{aligned} \tag{15}$$

Substituting equations 14 and 15 into equation 13 results in $p(\boldsymbol{\theta}_m|\widetilde{\mathcal{D}})$:

$$\propto \exp\Big(-\frac{\delta_m}{2}\boldsymbol{\theta}_m^T\boldsymbol{\theta}_m\Big)\prod_{i=1}^{N_m}\exp\Big(-\frac{1}{2}\boldsymbol{\theta}_m^T \boldsymbol{J}_{f_m,i}{}^T \boldsymbol{H}_{\mathcal{L}_m,i} \boldsymbol{J}_{f_m,i}{}^T \boldsymbol{\theta}_m + \boldsymbol{\theta}_m^T \boldsymbol{J}_{f_m,i}{}^T \Big[\boldsymbol{H}_{\mathcal{L}_m,i} \boldsymbol{J}_{f_m,i} \hat{\boldsymbol{\theta}}_m - \boldsymbol{R}_{\mathcal{L}_m,i}\Big]\Big) + \text{constant}. \tag{16}$$

This completes the proof.

■

Remark. *We remark that for each DNN expert with Laplace Approximation, there is a counterpart linear model with features maps $\boldsymbol{J}_{f_m}(\boldsymbol{x})$ that has the same posterior distribution as the original DNN expert. This is visualized in figure 1 in a function space view, mapping a linear model to a GP.*

An alternative path to obtain the NLMs is a linearization trick of MacKay [39]. MacKay [39] applies the first order tayler series to the output of neural network. Then, considering the Bayesian formulation of linear models with Gaussian priors and the same Gaussian posterior distributions of the original network (obtained using Laplace Approximation), MacKay [39] shows that the predictive uncertainty can be obtained in a closed form solution, due to the conjugating properties of Gaussians for linear models. An elegant work of Khan et al. [29] extends MacKay [39] by pointing out that we can recover GPs from a neural network with Gaussian posterior distributions. Our proof here is a specific instance of these two works - we project the same set-up within MoE models with a hard gating function.

Now, we extend to consider MoEs with a hard gating function globally. Having the individual DNN experts having an equivalent GPs (as stated in the previous proposition), we show similar derivation step on how MoEs with DNNs have an equivalent posterior distribution with MoE-GPs.

Lemma. ***2.1 (Global Duality)*** *Let m Deep Neural Network experts form a hard mixture of experts model such that $\mathbf{y} = \sum_{m=1}^{M} g_m(\boldsymbol{x}) f_{\boldsymbol{\theta}_m}(\boldsymbol{x})$. We denote its Laplace Approximation based posterior as $p(\boldsymbol{\theta}|\mathcal{D})$. Under a transformation $\widetilde{\mathcal{D}} = \Big\{\widetilde{\mathcal{D}}_1, \cdots, \widetilde{\mathcal{D}}_m\Big\}$, $\widetilde{\mathbf{y}} = \sum_{m=1}^{M} g_m(\boldsymbol{x}) \widetilde{f}_{m_{\boldsymbol{\theta}}}(\boldsymbol{x})$ is a mixture of experts with linear regressor that has an equivalent posterior distribution $p(\boldsymbol{\theta}|\widetilde{\mathcal{D}})$.*

Proof. The proof sketch is as follows. We first show that for both $\boldsymbol{y}$ and $\widetilde{\boldsymbol{y}}$, the posterior distribution can be factorized as a product of individual m^{th} expert's posterior distribution: $p(\boldsymbol{\theta}_m|\mathcal{D}_m)$ and $p(\boldsymbol{\theta}_m|\widetilde{\mathcal{D}})$. Then, using the results of Lemma 1, we proof that for a hard mixture of experts, the posterior distribution of two models are equivalent.

First, we express the likelihood of the MoE with DNNs as,

$$\begin{aligned}p(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta}_m) &= \prod_{i}^{N}\sum_{m=1}^{M} g_m(\boldsymbol{x}_i) p(\mathbf{y}_i|\boldsymbol{x}_i, \boldsymbol{\theta}_m), \\ &= \sum_{m=1}^{M} g_m(\boldsymbol{x}_1) p(\mathbf{y}_1|\boldsymbol{x}_1, \boldsymbol{\theta}_m) \sum_{m=1}^{M} g_m(\boldsymbol{x}_2) p(\mathbf{y}_2|\boldsymbol{x}_2, \boldsymbol{\theta}_m) \cdots \sum_{m=1}^{M} g_m(\boldsymbol{x}_N) p(\mathbf{y}_N|\boldsymbol{x}_N, \boldsymbol{\theta}_m).\end{aligned} \tag{17}$$

Here, we have assumed i.i.d data. Without loss of generality, we assumed an ordered assignment of data to individual experts, that is, $i \in \{1, 2, \cdots N\}$ is decomposed into $i \in \{i_1, i_2, \cdots i_M\}$ where for all m, $i_m \in \{1, 2, \cdots N^m\}$. This means that each m^{th} expert is responsible for data points i_m such that

$g_m(\boldsymbol{x}_{i_m}) = 1$ for all $i_m = 1, 2, \cdots N_m$. Otherwise, the gating network outputs zero by definition. As a result, we can further decompose equation 17:

$$
\begin{aligned}
&= \prod_{i_1}^{N_1} \sum_{m=1}^{M} g_m(\boldsymbol{x}_{i_1}) p(\mathbf{y}_{i_1}|\boldsymbol{x}_{i_1}, \boldsymbol{\theta}_m) \prod_{i_2}^{N_2} \sum_{m=1}^{M} g_m(\boldsymbol{x}_{i_2}) p(\mathbf{y}_{i_2}|\boldsymbol{x}_{i_2}, \boldsymbol{\theta}_m) \cdots \prod_{i_M}^{N_M} \sum_{m=1}^{M} g_m(\boldsymbol{x}_{i_M}) p(\mathbf{y}_{i_M}|\boldsymbol{x}_{i_M}, \boldsymbol{\theta}_m), \\
&= \prod_{i_1}^{N_1} p(\mathbf{y}_{i_1}|\boldsymbol{x}_{i_1}, \boldsymbol{\theta}_m) \prod_{i_2}^{N_2} p(\mathbf{y}_{i_2}|\boldsymbol{x}_{i_2}, \boldsymbol{\theta}_m) \cdots \prod_{i_M}^{N_M} p(\mathbf{y}_{i_M}|\boldsymbol{x}_{i_M}, \boldsymbol{\theta}_m) \\
&= \prod_{m}^{M} \prod_{i_m}^{N_m} p(\mathbf{y}_{i_m}|\boldsymbol{x}_{i_m}, \boldsymbol{\theta}_m).
\end{aligned}
\tag{18}
$$

We now define the prior over the parameters $\boldsymbol{\theta}$ by adopting the classical idea of automatic relevance determination [28], which assumes a factorized Gaussian prior:

$$
p(\boldsymbol{\theta}) = \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{\theta}_m|\mathbf{0}, \boldsymbol{A}_m) \quad \text{where } \boldsymbol{A}_m = \text{diag}(\delta_1, \delta_2, \cdots, \delta_m). \tag{19}
$$

This follows our previous definition of prior for individual m expert parameters. Now, we write the posterior distribution of mixtures of experts model:

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathcal{X}, \mathcal{Y}) &\propto \prod_{m}^{M} \prod_{i_m}^{N_m} p(\mathbf{y}_{i_m}|\boldsymbol{x}_{i_m}, \boldsymbol{\theta}_m) \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{\theta}_m|\mathbf{0}, \boldsymbol{A}_m), \\
&\propto \prod_{m}^{M} \prod_{i_m}^{N_m} p(\mathbf{y}_{i_m}|\boldsymbol{x}_{i_m}, \boldsymbol{\theta}_m) \mathcal{N}(\boldsymbol{\theta}_m|\mathbf{0}, \boldsymbol{A}_m), \\
&\propto \prod_{m}^{M} p(\boldsymbol{\theta}_m|\mathcal{D}).
\end{aligned}
\tag{20}
$$

Similarly, we can also express equation 20 under the data transformation as:

$$
p(\boldsymbol{\theta}|\mathcal{X}, \widetilde{\mathcal{Y}}) \propto \prod_{m}^{M} p(\boldsymbol{\theta}_m|\widetilde{\mathcal{D}}). \tag{21}
$$

This holds as the gating network and the input data $\mathcal{X}$ are kept the same, and the data transformation is defined on transformed output by the Jacobians of neural networks. Now, as we have already showed in Lemma 1 that individual experts have the equivalent posterior distribution, it also follows that $p(\boldsymbol{\theta}|\mathcal{X}, \mathcal{Y})$ and $p(\boldsymbol{\theta}|\mathcal{X}, \widetilde{\mathcal{Y}})$ are equivalent. We note that hard gating network enables this step, making it into a valid probability distribution under reverse engineering the densities.

This completes the proof. ■

Remark. *This Lemma states that if we assume a hard MoE where the experts are DNNs with Gaussian posteriors (obtained using Laplace Approximation), there then exists a hard MoE with the NLMs, which is dictated by the same gating function and has equal distribution of the parameters given the data. This establishes the duality of the two models in Bayesian sense, where the models are represented by the same probability distribution. Intuitively, such relationships can be obtained as we take a hard gating function, making each DNN experts probabilistic-ally independent.*

An important ramification of this result can be obtained in a well-known equivalence of weight space and function space view. To explain, we can obtain stochastic processes:

$$\widetilde{\mathbf{y}} = \sum_{m=1}^{M} g_m(\boldsymbol{x}) \widetilde{f}_{\mathrm{GP}_m}(\boldsymbol{x}) + \epsilon \quad \text{with} \tag{22}$$

$$\widetilde{f}_{\mathrm{GP}_m}(\boldsymbol{x}) \sim \mathrm{GP}(\mathbf{0}, \frac{1}{\delta_m} \boldsymbol{J}_{f_m}(\boldsymbol{x})^T \boldsymbol{J}_{f_m}(\boldsymbol{x})), \tag{23}$$

which is a MoE-GP with NTK, i.e. a tangent kernel with $\boldsymbol{J}_{f_m}(\boldsymbol{x})$. Consequently, the generative model and the predictions $\mathcal{N}(\widetilde{\mathbf{y}}_m^*, \widetilde{\boldsymbol{\Sigma}}_m(\boldsymbol{x}^*))$ on the new test datum $\boldsymbol{x}^*$ are given by:

$$\begin{bmatrix} \widetilde{\boldsymbol{y}}_m \\ \widetilde{f}_{\mathrm{GP}_m}(\boldsymbol{x}^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \quad \begin{bmatrix} \boldsymbol{K}_m(\mathcal{X},\mathcal{X}) + \sigma_{0,m}\boldsymbol{I} & \boldsymbol{k}_m(\mathcal{X},\boldsymbol{x}^*) \\ \boldsymbol{k}_m(\boldsymbol{x}^*,\mathcal{X}) & \boldsymbol{k}_m(\boldsymbol{x}^*,\boldsymbol{x}^*) \end{bmatrix},\right)$$

$$\begin{aligned} \widetilde{\mathbf{y}}_m^*(\boldsymbol{x}^*) &= \boldsymbol{k}_{m,*}^T (\boldsymbol{K}_m + \sigma_{0,m}\boldsymbol{I})^{-1} \widetilde{\boldsymbol{y}}_m, \\ \widetilde{\boldsymbol{\Sigma}}_m(\boldsymbol{x}^*) &= \boldsymbol{k}_m(\boldsymbol{x}^*,\boldsymbol{x}^*) - \boldsymbol{k}_T^{m,*} (\boldsymbol{K}_m + \sigma_{0,m}\boldsymbol{I})^{-1} \boldsymbol{k}_{m,*} + \sigma_{0,m}, \end{aligned} \tag{24}$$

respectively. Here, $\boldsymbol{K}_m(\mathcal{X},\mathcal{X}) = \boldsymbol{K}_m = \frac{1}{\delta_m} \boldsymbol{J}_{f_m}(\boldsymbol{x})^T \boldsymbol{J}_{f_m}(\boldsymbol{x})$, and $\boldsymbol{k}_m(\mathcal{X},\boldsymbol{x}^*) = \boldsymbol{k}_{m,*}$. Also, data uncertainty is captured by $\sigma_{0,m}\boldsymbol{I}$. Overall, with these steps, we show the duality of MoEs with DNNs and MoE-GPs in a Bayesian sense. We note that if MoEs with DNNs and Gaussian approximations, we can exploit their equivalent MoE-GPs without any approximation errors. Unfortunately, as MoEs with DNNs may often not be used in practice, we expand our theoretic work to a single DNN.

Therefore, we now establish a connection between a DNN, and MoE-GP as shown in figure 1, in order to increase the applicability of our theoretic results. To do so, we point out that a single trained DNN can be treated as a MoE-DNN, if all the DNN experts are essentially the same DNNs and a hard gating function is designed to strictly divide the input space amongst the experts, i.e. only one local DNN expert per division.

Proposition. ***2.2 (Equivalence in Input-Output Relationships)*** *Let $f_{\boldsymbol{\theta}}$ be a deterministic deep neural network with the input-prediction set given by $\left\{(\boldsymbol{x}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right\}_{i=1}^N$ for all N data points. Define a mixtures of experts model, with a hard gating function $g_m(\boldsymbol{x}_i)$ and the neural network experts $f_{\boldsymbol{\theta}_m} = f_{\boldsymbol{\theta}}$ for all $m = 1,..,M$ experts. Then, the mixtures of experts model have an equivalent input-prediction set to the deterministic deep neural network, given by $\left\{(\boldsymbol{x}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right\}_{i=1}^N$.*

Proof. The proof follows by the definition of the given mixtures of neural network experts.

From the definition, we have a hard gating function that assigns only a single neural network experts per local subset. This means we do not perform any weighted averaging over the predictions. For the inputs $\boldsymbol{x}_i$, the predictions of the mixtures of neural network experts are given by:

$$\mathbf{y}_i = \sum_{m=1}^{M} g_m(\boldsymbol{x}_i) f_{\boldsymbol{\theta}_m}(\boldsymbol{x}_i), \tag{25}$$

for all i and m. From our definition, we have $f_{\boldsymbol{\theta}_m} = f_{\boldsymbol{\theta}}$. Also, $g_m(\boldsymbol{x}_i) = 1$ if $\boldsymbol{x}_i$ belongs to the m^{th} expert. Otherwise, we have $g_m(\boldsymbol{x}_i) = 1$. This results in $\left\{(\boldsymbol{x}_i, \mathbf{y}_i)\right\}_{i=1}^N = \left\{(\boldsymbol{x}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))\right\}_{i=1}^N$.

This completes the proof. ■

Remark. *Under these given conditions, the input-prediction relationships of above two models are the same for both train and test data. This implies that (i) a single DNN can also be seen as a hard MoE with GPs (see section 3.3 and illustration in figure 1), and (ii) we assume that the data is stationary. We note that obtaining a hard MoE with DNNs, under the stated conditions, may not necessarily involve a training step.*

You can imagine training a single DNN on the entire dataset, and theoretically, this is the same as having a hard MoE with the same trained DNNs for the predictions within training and test set. For example, lets try to form an ensembles of DNNs with exactly the same network with respect to the structure of the learner and the network parameters. Moreover, imagine a predictive model, where we do not average the ensembles of DNNs for predictions, but pick only a single member of the ensemble per input data. Intuitively, the input-prediction relationships are the same to a single DNN with the same network parameters and structure.

What does then proposition 2 implies to the GP experts? So far, we have (i) shown the equivalence between MoEs with DNNs and GPs previously, and (ii) established a connection between MoEs with DNNs to a single DNNs. This is by introducing the specific conditions on the gating function, and how MoEs with DNNs are formed. Considering the GP experts, the given extension from MoEs with DNNs to a single DNN implies $\boldsymbol{K}_1 = \boldsymbol{K}_2 = \cdots = \boldsymbol{K}_M$ for all M. This is because we have taken a single DNN as the DNN experts within MoEs model, and brings an assumption that the data is stationary. We note that the overall kernel matrix can model non-stationary process as it is still possible to keep different hyper-parameters δ_m and σ_m. We present a formal statement about the proposed approximation, when compared to a single DNN with Gaussian approximations, which has a true equivalent GPs f_{GP}.

Lemma. ***2.2 (On Approximation Error)*** *Let $f_{\text{GP}}(\boldsymbol{x})$ be a true stationary process such that $\boldsymbol{K}_m(\cdot,\cdot) = \boldsymbol{K}(\cdot,\cdot)\ \forall m$. Then, our approximation to $\boldsymbol{K}_{true}(\mathcal{X},\mathcal{X})$ is $\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \mathbb{1}_{\boldsymbol{c}_i=\boldsymbol{c}_j}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)$ where $\boldsymbol{c}_i$ is the expert assignment for $\boldsymbol{x}_i$, indicator function $\mathbb{1}_{a=b} = 1$ when $a = b$ and 0 otherwise. Then, the approximation error of the MoE-GP kernel is $\|\boldsymbol{K}(\mathcal{X},\mathcal{X}) - \boldsymbol{K}_{true}(\mathcal{X},\mathcal{X})\|_F^2 = \sum_{ij}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2 - \sum_{m=1}^{M}\sum_{ij\in\mathfrak{V}_{\mathfrak{m}}}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2$.*

Proof. Assume we have ordered sets of points as before, and assume the same kernel matrices $\boldsymbol{K}_m(\cdot,\cdot) = \boldsymbol{K}(\cdot,\cdot)\ \forall m$. Now, recall the definition of our gating network that $g_m(\boldsymbol{x}) = 1$ in just one coordinate for each input, $\forall m$. The generative model, considering the entire data to experts are then:

$$\begin{bmatrix}\boldsymbol{y}_1\\ \boldsymbol{y}_2\\ \vdots\\ \boldsymbol{y}_M\end{bmatrix} = \text{GP}\left(0, \begin{bmatrix}\boldsymbol{K}_1 & 0 & 0 & 0\\ 0 & \boldsymbol{K}_2 & 0 & 0\\ 0 & 0 & \ddots & 0\\ 0 & 0 & 0 & \boldsymbol{K}_M\end{bmatrix}\right)$$

.

This explains the resulting approximation, that $\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \mathbb{1}_{\boldsymbol{c}_i=\boldsymbol{c}_j}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)$ where $\boldsymbol{c}_i$ is the partition for $\boldsymbol{x}_i$, $\mathbb{1}_{a=b} = 1$ when $a = b$ and 0 otherwise. In other words, assuming a MoE-DNN with a single, equivalent DNN results in a block-diagonal approximation in the kernel matrix. Decomposing the Frobenius norm, then results in $\|\boldsymbol{K}(\mathcal{X},\mathcal{X}) - \boldsymbol{K}_{true}(\mathcal{X},\mathcal{X})\|_F^2 = \sum_{ij}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2 - \sum_{m=1}^{M}\sum_{ij\in\mathfrak{V}_{\mathfrak{m}}}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2$.

This completes the proof. ■

Remark. *Here, $\sum_{ij}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2$ is a constant while the term $\sum_{m=1}^{M}\sum_{ij\in\mathfrak{V}_{\mathfrak{m}}}\boldsymbol{K}(\boldsymbol{x}_i,\boldsymbol{x}_j)^2$ specifies that less approximation error occurs when less correlated data points (or off-diagonal elements) are assumed to be independent while strongly correlated points by NTK should be within the same GP experts. In other words, the more block diagonal a true equivalent GP, the better MoE-GPs can approximate it.*

Intuitively, a MoE-DNN with Gaussian approximations and a MoE-GP have the equivalent posterior distribution, where GPs have their kernel matrix defined by their counterpart DNN experts (Lemma 2.1). If we force a MoE-DNN to a single DNN with the stated conditions, the GP experts have the kernel matrix, which is defined by the counterpart DNN experts that have the same posterior distribution. This Lemma quantifies the result of this step, and as the GP experts have the same kernel function, the given MoE-GP assumes stationary data.

We discussed the derivations and the results of our theoretic work, where a relationship has been established between a DNN and MoE-GPs with NTK. In summary, we showed how a MoE-DNN and a MoE-GPs are dual in Bayesian sense, and further established a connection to a single DNN. With this, we have gained an insight of the proposed idea, i.e. the consequences of the resulting

approximations and the conditions on when the two models are equivalent. We note that the gained insight has been explored in the design of our gating function from the main paper. Next, we provide more theoretic justifications, and the key benefits that stems from the derived theory.

3.4 Theoretic Justifications and Key Benefits

Our proposed predictive model comprises of a DNN for accurate predictions, as well as a MoE-GP with NTK for the predictive uncertainty. We first justify the resulting predictive model as follows.

Consider the square loss $\mathcal{L}_m = \frac{1}{2}\left(\boldsymbol{y}_m - f_{\boldsymbol{\theta}_m}(\boldsymbol{x}_m)\right)^2$ so that $\boldsymbol{R}_{\mathcal{L}_m}(\boldsymbol{x}) = \sigma_m^{-2}(f_{\boldsymbol{\theta}_m}(\boldsymbol{x}) - \boldsymbol{y}_m)$ and $\boldsymbol{H}_{\mathcal{L}_m}(\boldsymbol{x}) = \sigma_m^{-2}\boldsymbol{I}$, the mapping between $\mathcal{D}_m$ and $\widetilde{\mathcal{D}}_m$ are simply related by $\widetilde{\mathbf{y}}_m^* = \boldsymbol{J}_{f_m}(\boldsymbol{x}^*)\hat{\boldsymbol{\theta}}_m - (f_{\boldsymbol{\theta}_m}(\boldsymbol{x}^*) - \boldsymbol{y}^*)$ for new predictions $\mathbf{y}^*$ and $\widetilde{\mathbf{y}}_m^*$. Rearranging the terms simply give us: $\boldsymbol{y}^* = \widetilde{\boldsymbol{y}}_m^* + f_{\boldsymbol{\theta}_m}(\boldsymbol{x}^*) - \boldsymbol{J}_{f_m}(\boldsymbol{x}^*)\hat{\boldsymbol{\theta}}_m$. Now, the variance will be the same for both $\boldsymbol{y}^*$ and $\widetilde{\boldsymbol{y}}_{m^*}$ due to other terms being a constant deterministic variables. This justifies the use of a GP model for uncertainty estimates, on top of DNN predictions. We show several more examples below from our experiments (section 3.5). Note that the outputs of our GPs are transformed outputs $\widetilde{\mathbf{y}}_m^*$, which are parametrized by the DNNs via $\boldsymbol{J}_{f_m}(\boldsymbol{x}^*)$ and $\hat{\boldsymbol{\theta}}_m$. Thus, we can keep the DNN predictions $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and disregard the mean outputs of the GPs.

It is crucial to note that there are several benefits of our proposed predictive model. First, by the design, we can inherit the benefits of the mixture of GP experts model [32, 33] for uncertainty estimates. These are (i) a divide-and-conquer principle that significantly reduces the computational complexity of the GPs, (ii) the ability to model non-stationary covariance for both epistemic and aleatoric uncertainty, and (iii) a natural support for a distributed computation. Second, the predicition accuracy is the same as for the original DNN while the uncertainty estimates can be computed from the equivalent GP model. In other words, we attain the best of both worlds: the predictive power of DNNs and the uncertainty estimates from scalable GPs. This is achieved in a decoupled manner, e.g. the DNN does not require re-training, nor degrade its predictive performance. And finally, as shown in equation 24, our method does not require multiple forward passes through the DNNs. This is in contrast to many existing methods which often require multiple samples of the DNN predictions for a single input datum (either from a deep ensemble [40] or model uncertainty [41]). Hence, the resulting probabilistic predictor is fast and accurate, which is of importance to robotics.

3.5 Additional derivations

Our framework uses the loss functions in order to derive the relationships between the predictions of DNNs and GPs. This means that per loss function, we need a derivation for the model information (e.g. the first or second order information such as Hessian or residuals). Here, we provide a derivation for two loss functions that are used in the considered network architectures of our experiments. These include mean squared errors and cross entropy. Unlike the rest of the paper, we omit superscript m, that indicates a specific expert but the discussion herein applies to any m. This is for the clarity of the exposition.

3.5.1 Inverse dynamics task

For the inverse dynamics task, we used a multi-layer perceptron (MLP) with means squared error as the loss function $\mathcal{L}$ per single input. It is defined as:

$$\mathcal{L} = \frac{1}{2}\left(\boldsymbol{y} - f_{\boldsymbol{\theta}}(\boldsymbol{x})\right)^2, \tag{26}$$

where the $\boldsymbol{x}$ is input point, $\boldsymbol{y}$ is the measured output (7 joints), $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the predictions of the trained MLP. Then, assuming an independent GP per joint, the Hessian $\boldsymbol{H}_{\mathcal{L}_m} = 1$, the residual $\boldsymbol{R}_{\mathcal{L}_m} = (f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y})$, and consequently, the transformed output $\widetilde{\mathbf{y}} = \boldsymbol{J}_f(\boldsymbol{x})\hat{\boldsymbol{\theta}} + f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y}$. Here, the quantities we infer are $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and $\widetilde{\mathbf{y}}$, and for predictive uncertainty: $\boldsymbol{\Sigma}(\widetilde{\mathbf{y}}) = \boldsymbol{\Sigma}(\boldsymbol{J}_f(\boldsymbol{x})\hat{\boldsymbol{\theta}} + f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y}) = \boldsymbol{\Sigma}(f_{\boldsymbol{\theta}}(\boldsymbol{x}))$. This theoretically justifies that GP, trained on $(\boldsymbol{x}, \widetilde{\mathbf{y}})$, can be used for uncertainty estimates, while keeping the original MLP prediction $f_{\boldsymbol{\theta}}(\boldsymbol{x})$.

3.5.2 Object detection task

Next, we describe the mathematical derivation related to DETR architecture for object detection. DETR is a state-of-the art object detection method, that uses transformers and also built on end-to-end principle. Object detection involves both classification and regression (bound box), which extends the previously described cases that only considered a regression task. Let $\boldsymbol{y}$ be the ground truth with $\mathbf{y}_i = (\mathbf{c}_i, \boldsymbol{b}_i)$. Here, $\mathbf{c}_i$ is the class labels, and $\boldsymbol{b}_i \in \begin{bmatrix} 0 & 1 \end{bmatrix}^4$ is the representation of a bounding box with centers, height and width of the box. DETR uses K object queries (decoded by the transformer layer), followed by MLPs that overall predicts $\mathbf{y} = \{\hat{\mathbf{y}}_i\}_{i=1}^K$. To match the number K, the ground truth label $\boldsymbol{y}$ is padded with $\varnothing$ (no object) into K.

For training, DETR first performs matching between the supervised labels and the predictions (each representing a set):

$$\begin{aligned} &\hat{\sigma} = \arg\min_{\sigma \in \wp} \sum_{i}^{K} \mathcal{L}_{\text{match}}(\boldsymbol{y}_i, \mathbf{y}_{\sigma(i)}) \;\; \text{with} \\ &\mathcal{L}_{\text{match}}(\boldsymbol{y}_i, \mathbf{y}_{\sigma(i)}) = -\mathbf{1}_{\{\mathbf{c}_i \neq \varnothing\}} \hat{\boldsymbol{p}}_{\hat{\sigma}(i)}(\mathbf{c}_i) + \mathbf{1} \mathcal{L}_{\text{box}}(\boldsymbol{b}_i, \hat{\boldsymbol{b}}_{\hat{\sigma}(i)}). \end{aligned} \tag{27}$$

The match index is denoted $\sigma(i)$. Given the optimal match w.r.t the above criteria, the used loss is defined:

$$\mathcal{L}(y, \mathbf{y})_{\text{DETR}} = \sum_{i}^{K} \Big[-\log \hat{\boldsymbol{p}}_{\hat{\sigma}(i)}(\mathbf{c}_i) + \mathbf{1}_{\{\mathbf{c}_i \neq \varnothing\}} \left\| \boldsymbol{b}_i, \hat{\boldsymbol{b}}_{\hat{\sigma}(i)} \right\|_2^2 \Big]. \tag{28}$$

The above loss is back-propagated whenever the object query receives the optimal assignment $\hat{\sigma}$ (equivalently, equation 27 is solved using Hungarian algorithm, and is not back-propagated).

Now, notice that per input datum x, the regression loss, whenever $\mathbf{c}_i \neq \varnothing$, is simply the MSE loss, and otherwise, do not exist. This means that our derivation only need to take into account, the bounding box uncertainty when $\mathbf{c}_i \neq \varnothing$. Then, using the same techniques as before, the transformed bounding box output is: $\widetilde{\hat{\boldsymbol{b}}}_{\hat{\sigma}(i)} = \boldsymbol{J}_f(\boldsymbol{x}_k)\hat{\boldsymbol{\theta}} - \hat{\boldsymbol{b}}_{\hat{\sigma}(i)} + \boldsymbol{b}_i$ for all i, and $\boldsymbol{\Sigma}(\widetilde{\hat{\boldsymbol{b}}}_{\hat{\sigma}(i)}) = \boldsymbol{\Sigma}(\boldsymbol{J}_f(\boldsymbol{x}_k)\hat{\boldsymbol{\theta}} - \hat{\boldsymbol{b}}_{\hat{\sigma}(i)} + \boldsymbol{b}_i) = \boldsymbol{\Sigma}(\hat{\boldsymbol{b}}_{\hat{\sigma}(i)})$. This justifies the use of GP uncertainty for bounding box regression.

For classification loss $-\log \hat{\boldsymbol{p}}_{\hat{\sigma}(i)}(\mathbf{c}_i)$, which is the broadly used cross-entropy loss, we now derive the relationship between the transformed output, and true output. Let $\mathbf{c}_i$ is given by $h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]$, where h is the sigmoid function (used in DETR), and $f_{i,\boldsymbol{\theta}}(\boldsymbol{x})$ is the activation of the classification layer for input point x. It implies that DETR uses the Bernoulli likelihood function, and we may treat the classification as regression (similar to Lu et al. [42]). Then, the residual term is given by $h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})] - \mathbf{c}_i$ and the Hessian of the sigmoid is: $h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})])$. Similar to a regression task, the transformed output:

$$\widetilde{\mathbf{c}}_i = \boldsymbol{J}_f(\boldsymbol{x})\hat{\boldsymbol{\theta}} - \Big[h\left[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})\right](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \Big]^{-1} \Big(h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})] - \mathbf{c}_i \Big) \;\; \text{or} \tag{29}$$

$$\mathbf{c}_i = h\Big(f_{i,\boldsymbol{\theta}}(\boldsymbol{x})\Big) + \Big[h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \widetilde{\mathbf{c}}_i \Big] - \Big[h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \Big] \boldsymbol{J}_f(\boldsymbol{x})\hat{\boldsymbol{\theta}}. \tag{30}$$

Lastly, taking the variance on both sides:

$$\begin{aligned} \boldsymbol{\Sigma}(\mathbf{c}_i) &= \boldsymbol{\Sigma}\Big(h(f_{i,\boldsymbol{\theta}}(\boldsymbol{x})\Big) + \Big[h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \Big] \widetilde{\mathbf{c}}_i - \Big[h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \Big] \boldsymbol{J}_f(\boldsymbol{x})\hat{\boldsymbol{\theta}}), \\ &= \Big[h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})](1 - h[f_{i,\boldsymbol{\theta}}(\boldsymbol{x})]) \Big]^2 \boldsymbol{\Sigma}(\widetilde{\mathbf{c}}_i). \end{aligned} \tag{31}$$

In above equation, we reason about the variance of the class assignment. Overall, this derivation motivates the use of GP for classification uncertainty within DETR, as the variance of transformed output is related to the original output. Then, we compute the Jacobian of DNN for the given test input. This involves backpropagation of the DNNs and is again bounded to how efficient DNN is. Then, our gating function assigns the test data to an expert and we compute GP uncertainty.

4 Algorithmic Overview

Algorithms 1 and 2 summarize our approach for training and testing respectively. Implemented using Gpytorch [43], these algorithms are executed with GPU supports. For an efficient implementation, we use the algorithms of Gardner et al. [43] for training, and Lanczos approximation [44] for testing, leveraging recent advances in software infrastructure for GPs.

Algorithm 1: A divide and conquer solution (off-line).

Data: A trained DNN $f_{\boldsymbol{\theta}}$ and train data $\mathcal{D} = \mathcal{X}, \mathcal{Y}$.
Result: A trained MoE-GP with $\{\delta_m, \sigma_{0,m}\} \forall m$
Compute $\mathcal{Y} = f_{\boldsymbol{\theta}}(\mathcal{X})$, $\boldsymbol{J}_f(\mathcal{X})$ and $\widetilde{\mathcal{Y}}$;
Apply the pruning step 1 on $\boldsymbol{J}_f(\mathcal{X})$;
Perform NTK PCA ;
K-means on principal components ;
for *all the M experts* **do**
    Apply the pruning step 2 on $\boldsymbol{J}_{f_m}(\mathcal{X})$;
    Active learning on $\boldsymbol{J}_{f_m bl}(\mathcal{X}) \forall l$;
    Patchwork-GP prior $\boldsymbol{K}_{m,\text{patch}}$;
    GP expert training ;
end

The training of GP refers to the use of the marginal log likelihood for the model selection:

$$\begin{aligned} \log \mathrm{p}(\widetilde{\boldsymbol{y}_m}|\boldsymbol{x}_m, \delta_m, \sigma_{0,m}) = &-\frac{1}{2}\widetilde{\boldsymbol{y}}_m^T(\frac{1}{\delta_m}\boldsymbol{K}_{m,\text{patch}} + \sigma_{0,m})\widetilde{\boldsymbol{y}}_m \\ &-\frac{1}{2}\log(\frac{1}{\delta_m}\boldsymbol{K}_{m,\text{patch}} + \sigma_{0,m}) - \frac{n_m}{2}\log(2\pi). \end{aligned} \tag{32}$$

The result of the above algorithm is a set of GPs with their hyperparameters, and respective Jacobians on the subset of data. Gating function, which uses NTK PCA and K-means clustering, is also saved, which constitutes of (i) the kernel matrix that belongs to the randomly selected data for clustering, and (ii) the cluster centroids. We also note that distributed training is natually supported in the above algorithm - the training of GPs can be distributed across the GPUs. If we use a different gating function for different multi-output GP, the division step can be distributed across the GPUs per output dimension. The most computationally involving part of the algorithm is the Jacobian computations and saving operations to the disks.

Algorithm 2: A fast GP uncertainty (on-line).

Data: A trained DNN $f_{\boldsymbol{\theta}}$, MoE-GP and test data $\boldsymbol{x}^*$.
Result: Prediction $\mathbf{y}^*$ and uncertainty $p(\mathbf{y}^*|\boldsymbol{x}^*, \mathcal{D})$
Compute the prediction: $\mathbf{y}^* = f_{\boldsymbol{\theta}}(\boldsymbol{x}^*)$;
Compute the test Jacobian: $\boldsymbol{J}_f(\boldsymbol{x}^*) = \frac{\partial \boldsymbol{y}^*}{\partial \boldsymbol{\theta}}(\boldsymbol{x}^*)$;
Gating network assigns m^{th} expert ;
Compute GP uncertainty $p(\boldsymbol{y}^*|\boldsymbol{x}^*, \mathcal{D})$;

Algorithm 2 computes the predictive uncertainty of DNNs with MoE-GPs. First, we compute the prediction of a DNN, which indicate that the computational efficiency of our approach is bounded to how fast DNNs can be. This is usually true for the most of the predictive uncertainty estimation methods. After this, we use the gating function to assign the test input to an expert and compute GP based uncertainty. The gating function projects the Jacobian into a principle axes (using the obtained NTK-based PCA), and perform K-means clustering of this test sample. The Lanczos approximation [44] is used for the computation of GP-based uncertainty. The test Jacobian is pruned to match the dimensions of the train Jacobians, using the obtained pruning masks.

5 Implementation Details and Additional Results

Implementation details and additional results are presented in this section.

We used a GPU cluster that consists of NVIDIA GTX 1080 Ti (Pascal), NVIDIA Titan V (Volta), NVIDIA Titan RTX (Turing) and NVIDIA V100 for our off-line processing of the experiments. The CPU pairs used are respectively, 12-core Intel Xeon W2133, 16-core Intel Xeon and 32-core AMD CPUs. On a robot, we use NVIDIA Jetson TX2 for evaluation of the run-time, which uses the Dual-Core NVIDIA Denver 2 64-Bit and the Quad-Core ARM Cortex-A57 MPCore CPUs. The board features 256-core NVIDIA Pascal GPU architecture with 8GB of memory. Communication has been established to other computing components of our robot using ROS.

5.1 Implementation Details for Laplace Approximation Variants

We compare our method against two Laplace Approximation [37] based baselines. The first, referred to simply as *Laplace*, is similar to [45], except that we focus on the diagonal Fisher information matrix (FIM) approximation and only consider all linear layers, including multihead-attention layers, but omitting convolutional layers. This decision allows for a fairer comparison against the second approach referred to as *rLaplace*, which is a linearized version the first approach similar to [46] where no sampling is required. Linearized Laplace approximation relies on the Generalized Gauss-Newton matrix (GNN) of which we also only consider a diagonal approximation for computational reasons.

Both methods are implemented in Python using the PyTorch framework [47, 48, 49]. We make use of the same pre-trained networks for all approaches. To compute the diagonal FIM and GNN approximations, we loop once over the respective training datasets using the same parameterization as during training of the network weights. Afterwards, we perform a coarse hyperparameter search for each approach and network architecture to find appropriate multiplicative factors used for scaling the FIMs and GNNs prior to inversion. For the standard Laplace approximation approach, we sample 30 weight configurations and perform one forward pass for each sample to obtain the distribution over outputs. For the linear Laplace approximation, we compute one GNN per layer and output dimension to increase the accuracy of the approximation.

5.2 Implementation Details for MC-Dropout Variants

We take MC dropout [41] and its real-time variants [50] which we name approximate variance propagation as two of the baseline approaches. Because both approaches rely on the pre-trained network with dropout layer, for inverse dynamic experiments, we insert the dropout into the last layer of the 3 layer MLP architecture which can obtain satisfied results with this way from our preliminary experiments. By referring to the Tensorflow implementation of approximate variance propagation, we re-implement this method with Pytorch. We have verified the implementation by obtaining similar performance on the same benchmarks (UCI) compared to original implementation.

Nevertheless, we also find that the current implementation does not serve well for the some advanced layers such as mult-head attention and LSTM. Therefore we need to build up the variance propagation layer for them from scratch. On account for the high complexity of the analytical derivations of the Jacobian matrix for these layers, which is hard and error-prone to implement, we decide to make use of the automatic differentiation functionality of Pytorch to achieve it. This comes at cost of the running time because of the additional backward operation. Specifically, limited by the only backward-mode automatic differentiation support in Pytorch, we need to loop over the output dimension to obtain each row of the Jacobian matrix which further reduce the efficiency of this method. However, we believe that by this way we can make use of the full power of it and aim to conduct a more fair and comprehensive comparison with our approach and other baseline methods. To note that, for variance propagation, we propagate covariance matrix in inverse dynamic experiments and a diagonal covariance matrix. For MC-dropout, 30 samples are used for evaluation.

5.3 Implementation Details for Deep Ensembles

We closely follow the recent protocol of Sharma et al. [51] for the implementation of deep ensembles [40]. In total 5 models of identical architecture from random initialization are trained for each datasets namely sarcos, kuka1 and kuka2, which is repeated over 3 random seeds. The first member of the ensemble is chosen from the deterministic neural networks, which are used across other baselines.

5.4 Discussion on Negative Log Likelihood Calculation

When evaluating different uncertainty estimation approaches with negative log likelihood (NLL) as metric, we are confronted with the issue of different ways to compute it from the quantities predicted by different models. Some predict only the samples from the output distribution directly, such as Monte-Carlo Dropout, while the other predict the moments such as mean and variance of the output distribution directly based on variance propagation[50] or linearization over a parameter point estimate [46].

Because the calculation of Gaussian likelihood requires the moments (mean and variance) and the observation variance, we can compute it by either following [41] to evaluate the NLL of each sample predicted by the model (NLL_{samples}, refer to equation 33):

$$NLL_{\text{samples}}(\boldsymbol{x}, \boldsymbol{y}) = -\log \sum_{i}^{T} \exp\left(-\frac{(\hat{f}_i(\boldsymbol{x}) - \boldsymbol{y})^2}{2\tau}\right) + \log T + \frac{1}{2}\log 2\pi - \frac{1}{2}\log \tau, \tag{33}$$

where τ is the observation variance, T is the number of samples, $\hat{f}_i$ are the sampled prediction. For using the predicted moments directly (NLL_{moments}, refer to equation 34):

$$NLL_{\text{moments}}(\boldsymbol{x}, \boldsymbol{y}) = -\log \exp\left(-\frac{(f_{\text{mean}}(\boldsymbol{x}) - \boldsymbol{y})^2}{2(f_{\text{var}}(\boldsymbol{x}) + \tau)}\right) + \frac{1}{2}\log 2\pi - \frac{1}{2}\log\left(f_{\text{var}}(\boldsymbol{x}) + \tau\right), \tag{34}$$

The principled selection of NLL should depend on the quantities they produce, however, because of the different formulations of them (when evaluating on one data pair $(\boldsymbol{x}, \boldsymbol{y})$).

Also, f_{mean} and f_{var} are the moments of the Gaussian distribution, predicted by certain methods. We note that equation 34 is an exact formula, while the equation 33 involves approximations. Therefore, the final calculated values differ in the scale of magnitude as well. In order to facilitate a fair comparison, we select the ones which perform best among them. Nevertheless, the issue on this should raise an alarm or attract more attention when researchers decide to choose NLL as their evaluation metric. This also holds when practitioners decide to select one uncertainty estimation method for their specific purpose. Unfortunately, other measures of regression uncertainty, such as calibration measures, are subject to the results being different depending on the chosen binning [52].

5.5 Implementation Details and Additional Results for MoE-GP

We provide the implementation details of the presented experiments and our method below.

5.5.1 Toy regression

Our toy regression experiment used Snelson data-set, which is often used in GP literature [53]. Due to the small scale, many experimental variables can be carefully controlled for better insights. In this experiment, we trained a single-layer Multi-layer perceptron (MLP) with 200 units and a tangent activation. The reported figure in the main text used 7 GP experts on this data-set. We used the Negative Log Likelihood (NLL) as our loss function and used 10000 epochs with Adam optimizer. It used the decay of zero, and learning rate of 0.1. We did not use any memory reduction hyperparameters as the used data and the architectures are small. The only difference between "without patch" and "with patch" setting was the use of the proposed patchwork prior.

Additional results. We perform additional experiments with GPs as a baseline. The used setup is as follows. We use the same Snelson data-set and a MLP to the toy regression experiments, which is described in the paragraph above. In addition to the comparisons on "in-between" uncertainty estimates [46], we also evaluate on the original Snelson data-set by keeping all the training data. This is to create an additional setup to evaluate if the conclusions still hold. What motivates Snelson data-set over other experiment setups of our work is the relatively small size of the data-set where full GPs can scale without any further approximations. We note that in GP literature, it is often referred to as large scale problems if the data-set contains more than 10000 data points [53] where the computational complexity of a full GP is prohibitive. Thus, we compare our approach to the full NTK - the kernel we attempt to approximate in order to improve its applicability beyond 10000 data points. We also include a GP with RBF as an additional reference. Lastly, we choose two different

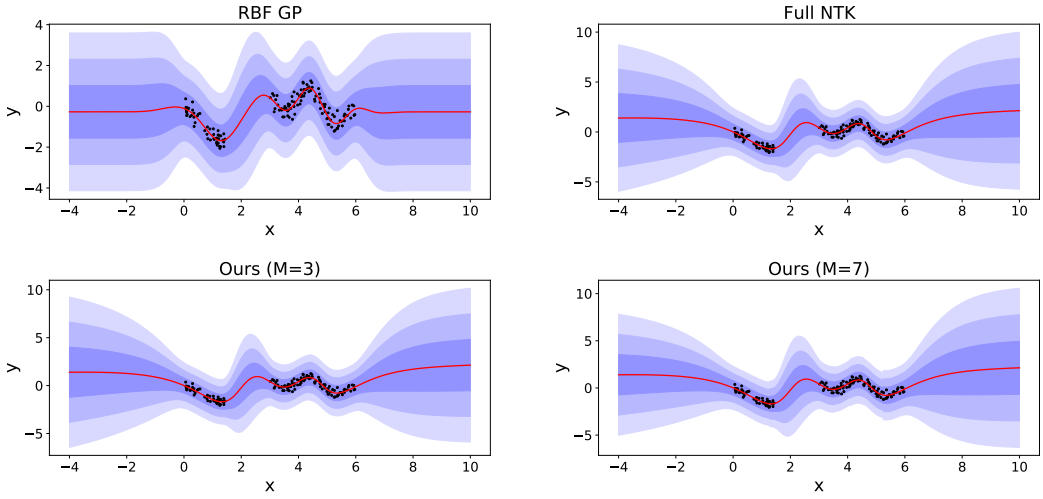

Figure 2: We visualize predictive uncertainty on Snelson dataset where we remove the training data of certain regime (between $x = -1$ and $x = 3$) to test the domain shift scenario. Evaluations on the entire snelson data is depicted in figure 3. The black dots are train data points, and the red line shows the mean predictions. Blue shades show up to three standard deviations. (Left-Top) We plot a GP with RBF kernel with the original predictions of the GP. (Right-Top) For a validation of the proposed concept, a full NTK without the mixtures of experts approximation is shown. (Left-Bottom) We plot the proposed concept with 3 GP experts. (Right-Bottom) The increase of experts to 7 is visualized. We note that the GP with RBF kernel utilizes a different kernel function and hence, deviations to the GP with the full NTK can be observed both in the predictions and their uncertainty estimates.

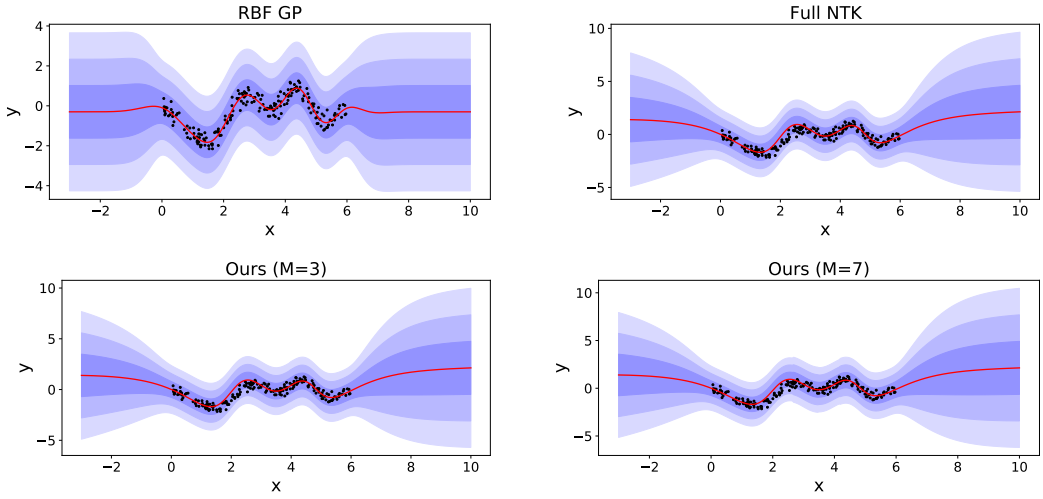

Figure 3: We visualize predictive uncertainty on Snelson dataset where we test the uncertainty estimates for out-of-distribution scenarios, and see if the data noise can be captured. Evaluations including the domain shift scenario is depicted in figure 2. The black dots are train data points, and the red line shows the mean predictions. Blue shades show up to three standard deviations. (Left-Top) We plot a GP with RBF kernel with the original predictions of the GP. (Right-Top) For a validation of the proposed concept, a full NTK without the mixtures of experts approximation is shown. (Left-Bottom) We plot the proposed concept with 3 GP experts. (Right-Bottom) The increase of experts to 7 is visualized. We note that the GP with RBF kernel utilizes a different kernel function and hence, the observed deviations to the GP with the full NTK can be observed both in the predictions and their uncertainty estimates.

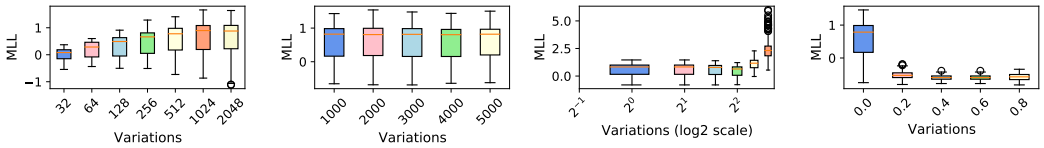

Figure 4: The effects of each hyperparameter is shown with train MLL, by varying them in different steps (Variations), and fixing the others to default settings. Lower the better.

settings of MoE-GPs with 3 experts and 7 experts, which is to intuitively support the ablation study on the hyperparameter choices in the paper. Figures 2 and 3 depict the results.

The following observations can be found in the results. First, the results show that all the considered models qualitatively provide well-calibrated uncertainty estimates, i.e. whenever there is no training data, the models show high uncertainty estimates for both domain-shift and out-of-distribution scenarios. When the regimes where the training data exists, all the models capture the data noise. Second, a GP with RBF kernel provides different predictions to the GPs that utilizes the NTK. This behavior is expected as both the models utilize different kernels. Lastly, we find that the proposed concept qualitatively provides similar uncertainty estimates to a GP with the full NTK. This result can be seen as an empirical evidence that the proposed approximations to the NTK can provide similar posterior predictive to a GP with the full NTK. The deviations also grow as more approximations are made, i.e. the number of data division grows. This observation is similar to the ablation study on the hyperparameter choices in the main paper, i.e. more number of experts can deteriorate the uncertainty estimates, as each GP experts contain less amount of data than the full GP with the full NTK. We refer to our theoretical results, which formalize the proposed approximations to the full NTK.

5.5.2 Ablation studies

In our ablation study, we focus on the issue of design choices within our approach. Concretely, there are 5 hyperparmeters namely, (i) the number of GP experts, (ii) the size of the subsets for clustering, the sparsity level of (iii) neural and (iv) jacobian pruning (can considered as a single hyperaparameter), and (v) the number of data points for the active learning. These design choices influence the uncertainty estimates, as well as the computational complexity. For the empirical study, we examine a regression task using a 5-layered, 200 units MLP on SARCOS data-set [53]. Often used, SARCOS contains 48933 data points with 21 input values from the joint sensors, and 7 joint torque values as targets. We used SGD optimizer with learning rate of 0.0001, moment of 0.9 and 500 epochs. The loss function was chosen as the mean squared error.

For the ablation studies, we vary each hyper-parameters while fixing the remaining ones, and we use the Marginal Log Likelihood (MLL) to evaluate on the training data and the Negative Log Likelihood (NLL) for the testing. The default setup is 512, 5000, 0, and 0 for the number of experts, the subsets for the division step, the level of pruning sparsity, and the proportion of actively selected points respectively. For the number of experts, the variations are: 32, 64, 128, 256, 512, 1024 and 2048. We varied the subset size for the division step from 1000 to 5000 in the steps of 5, and the result follows the conclusions of Chitta et al. [54]. We pruned the Jacobians from upto 0.00001 percent of the layers: 0.0, 0.3, 0.6, 0.9, 0.99, 0.9999, 0.99999. This is to examine the extreme cases where we keep only a few feature maps of the Jacobians for the GP experts. Lastly, we varied the size of the actively selected points from 0.0 to 0.8 in the steps of 0.2. Figure 4 shows the plot for MLL while we have shown NLL plots in the main paper. We did not see any significant difference in trend, when comparing both the NLL and the MLL.

The details of the stopping criterion for the active learning and the Jacobian pruning is as follows, which have been the implementation details adapted in all the relevant experiments. In the active learning, we choose the ratio of the most informative data points to select, e.g. 80 percent of the data points. Then, the number of query steps are decided, i.e. we choose 3 query steps in all the experiments. Selecting the maximization of information as an acquisition function, the active learning process continues unto the selected query steps. Similar one-shot pruning strategy is adapted as oppose to an iterative pruning. Similar to active learning, the pre-specified ratio of the pruning. Ranking the parameters according to the given metric, the top parameters in the ranking are kept

where the numbers of given by the pre-specified ratio. The recipe for the choice of these parameters have been discussed in the ablation study on hyperparameters.

5.5.3 Learning inverse dynamics of a manipulator

Now, we consider the task of robot inverse dynamics learning, and evaluate our approach against existing methods for uncertainty estimation. The goal herein is to learn a mapping from the joint positions, velocities and accelerations to toques τ (Nm) for all 7 joints. Again, our goal is not to obtain the most accurate prediction, but to estimate predictive uncertainty. A 5-layered MLP is trained on SARCOS, KUKA 1, KUKA 2 with the above mentioned data pairs from SARCOS and KUKA robot arms respectively. We use the test/training split ratio of 1/9 for SARCOS, KUKA 1 and KUKA 2, ratio of 1/99 for KUKA SIM dataset (used for testing). Besides the nominal settings, to evaluate our approach in a more realistic and challenging way, an OOD scenario, where the models are evaluated on a complete different data-set to training (e.g. train on SARCOS, and test on KUKA 1), are conducted. The results are averaged over three random seeds for the error bars of the numerical results.

The used hyperparameters are as follows. For SARCOS, we use 10 Nr. experts, 5000 subsets, sparsity levels of 0.5 and 0.9 for pre-pruning and post-pruning respectively, and use 0.3 for the level of active points. For KUKA, we use 50 Nr. experts, and keep the other hyperparameters the same as SARCOS. We believe that there are still rooms for better hyperparameter tuning. All GPU memory was not used, as the current implementation can fail due to some GP experts having more data points than others. While SARCOS contains $\sim$ 50k data pairs of $100hz$ rhythmic motions, KUKA 1 and KUKA 2 have $\sim$ 200k pairs of rhythmic motions at various speed, respectively. KUKA SIM has a larger size of 2 million data pairs recorded in simulation. We use the test/training split ratio of 1/9 for SARCOS, KUKA 1 and KUKA 2, ratio of 1/99 for KUKA SIM dataset.

5.5.4 Distributed training

We have tested the scalability of our method. From a theoretic view, MoE-GP scales to infinitely large data-sets by creating more and more GP experts. Yet, for the method to be practical, the question is how long the training takes to scale. Scalability is of significant importance in this case, as DNNs operate in the regime of big data, and an exact GP does not scale to such settings. To this end, using the same MLP as before, we train MoE-GP on KUKA SIM data-set [55], which contains 1984950 data points. At this scale, an exact GP is clearly not scalable, and many existing solutions resort to incremental learning [55]. Furthermore, we use a cluster with NVIDIA V100s, and perform a distributed training. In MoE-GPs, a key benefit over the other sparse GPs lies on its distributed, local nature, and we can exploit it in practice.

To this end, we use 512 GP experts per joint torques, and use 100 iterations for MLL optimization. For stable training due to the current implementations, we use less CPU cores per GPUs, while on a single GPU training, all the 32 CPU cores are used. Thus, we see a drop of training time performance when compared to a single GPU. The other hyperparameters used are (i) the subset size of 5000, (ii) the active learning parameter of 0.0 (iii) and the pruning parameters 0.5 and 0.999 respectively. The training time of an algorithm depends on several factors such as implementation, size of the DNN and available computational resources. Our attempt is to show how much MoE-GP can scale to large datasets within a reasonable time frame.

5.5.5 Probabilistic Object Detection

We also evaluate our method within the context of an on-going space demo mission for future planetary explorations. The scenario of interests involves a team of heterogeneous robots where the flying system is used for scouting. Once all areas of interest are scouted, the system returns back to the landing units (either on a lander, or on a rover). This task of homing is challenging, as the localization algorithm may have accumulated drift or because the roving unit has moved. Therefore, it is necessary to find these particular objects again. Importantly, it is mission critical, that the object detector outputs a correct confidence measure. For example, as the system should just attempt a landing, if it is confident enough to have detect the desired landing systems. This is why we choose to apply our method to the given use-case.

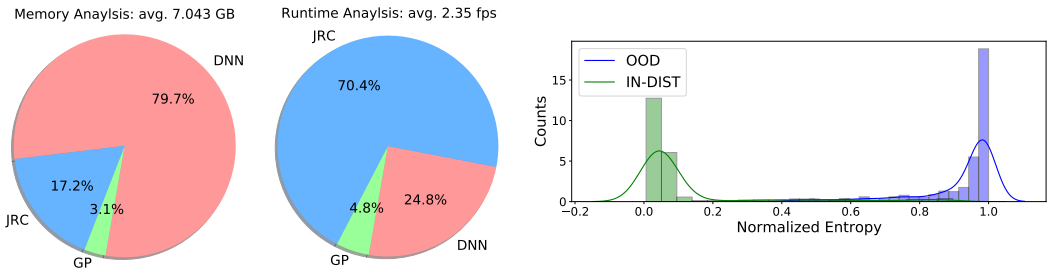

Figure 5: The GPU memory and run-time analysis shows the overhead caused by the uncertainty computations (left). JRC means the jacobians related computations. The normalized entropy is shown for OOD, and IN-DIST data, collected on our robot (right). A clear separation is observed, showing the ability to detect OOD samples. A configuration of the hyperparameters and the implementations are chosen to be a high-end. The memory requirement of a Jetson TX2 is 8GB.

The implementation details are as follows. MoE-GPs with the total 30 GP experts (3 GPs per output), and a single gating network with 5000 subsets are used. We select 80% of the neighboring GPs for the active learning hyperparameter. The same data augmentation as DNNs are used, keeping up to 10000 images of flips, and brightness changes. Importantly, for efficiency, we did not use torch.autograd but manually derived the Jacobians of fully connected layers, and incorporated them into our forward pass. As the original DETR architecture with ResNet backbones are not designed for real-time applications, we instead use EfficientNet backbone, combined with torch.jit functionality to make the DETR architecture more efficient. We do not consider other popular frameworks of object detection which requires several post-processing steps. This is because end-to-end architectures are a preferred way to evaluate uncertainty estimates, without dependencies on how uncertainty estimates are propagated through the post-processing steps. The uncertainties are computed only for the deterministic predictions above a threshold of 0.5. We further note that there are many different ways of designing the probabilistic object detector with our method, and we aim to see the potentials of our algorithm in practice.

We also report other trials we have conducted where we used the Jacobians of the entire DNN. As the first trial, we use the backpropagation algorithm (torch.autograd) to compute the Jacobians of the entire DNN online, and pruned the jacobians upto 4832 parameters. Again, torch.script and pruning of convolution and fully connected layers have been combined to increase the run-time of the DETR architecture. The results are shown in figure 5 which shows the run-time and memory analysis, as well as the ability to separate OOD samples via the entropy histogram. Interestingly, the most of computations are related to computing the jacobians of DNNs. We learned that this is specific to PyTorch, which is designed for back-propagation of loss, as oppose to the output. We note that this version of the implementation is slower than the version we present in the main paper, but achieves better OOD detection performance. This shows the relevance of selecting from the Jacobians of the entire DNN as oppose to the last few layers. Moreover, this empirically shows an application of our method to convolutions and transformer architecture. For improving the efficiency, one way to leverage this set-up in practice is the use of JAX for the Jacobians computations online.

We next discuss the lessons learned for the future venues of research.

6 Lessons Learned and Limitations

To our knowledge, we are the first to evaluate different uncertainty estimation techniques on a Jetson TX2, for DNN-based robot perception. During the implementation and experimentation we have conducted, we learned a few lessons which we would like to share with the community.

6.1 Ensembles or sampling can be parallelized on a robot. Decoration or real?

We find the difficulty of parallelizing the methods based on ensembles or sampling on a Jetson TX2. This is due to the limited on-board memory of a GPU and the limited number of CPU cores. While there might be a possibility to parallelize the DNN forward passes for the ground-based robots such as

autonomous cars, from our experience of building robots [56, 57, 58], the readers may also consider the number of other processes running onboard a robot. Such processes can be any localization methods and any planning methods, which can take several CPU cores.

The main problem of deep ensembles over MC-dropout is that for robot perception especially, it is often not feasible to deploy deep ensembles, because many large DNNs cannot be stored on a single embedded GPU simultaneously. As loading a DNN to GPU alone can take upto several seconds, deep ensembles tend to be slower than MC-dropout. One may consider an example of a robot perception task, where the robot is equipped with 8GB of memory (e.g. Jetson TX2) and a single neural network is 5GB in GPU memory. In this case, lowering the number of ensembles cannot allow for real-time applications. Even in the case where a single neural network is 2GB and we can fit 4 ensemble members to a single GPU, paralleling the predictions on the real robot may not be trivially done – or we are not aware of any existing implementations or proof that this is possible for the real-robot object detection.

In this regard, the new research line on "sampling free" methods are attempting to address this limitation [50], and the recent survey paper on probabilistic object detection [59] and others [3] also state this as an important direction of research. Therefore, we find the sampling-free or single-forward pass methods of computing the predictive uncertainty, to be relevant in practice for robot perception.

6.2 Software infrastructure for Jacobian computations

Our method depends on the Jacobian computations of a DNN. As reported in section 5.5, a naive way of using popular software infrastructure such as PyTorch can deteriorate the run-time efficiency. While an analytical derivation of the Jacobian is feasible for fully connected layers and convolution layers, such derivations may not be possible for more sophisticated DNNs such as transformers. We see JAX as a step forward for the methods that use NTK and consequently the Jacobian computations.

For deployments on a real robot, however, we find that using the Jacobian associated with the last few layers can also be a practical alternative. In our formulation, this corresponds to pruning all the other layers except for the Jacobian of last few layers. Our experiment suggests that by using the Jacobian of last few layers, we can still perform novelty detection better than the widely used technique called MC-dropout, while being about 12 times faster on our hardware. The performance is better by considering the Jacobian of the entire neural network along with pruning, but this result in slower inference time. Therefore, we find that using the Jacobian of the last few layers can be a practical alternative in terms of run-time and performance trade-offs.

6.3 A concrete use case of the proposed method

The proposed learning algorithm is subject to a trade-off, which plagues from the use of GPs with a high dimension feature space, i.e the Jacobians of DNNs. While we have shown what is possible with the proposed method (in terms of scalability, predictive uncertainty and run-time), the algorithm is inherently subject to the limitations of GPs. GPs are powerful tools for probabilistic machine learning, but may limited in the applicability with respect to both the growing number of available data points and the number of outputs. For the later, leveraging the recent advances in multi-output GPs can be an important direction of research. However, the memory consumption due to the fact that you need to store the training data, may limit the applicability within the text of uncertainty quantification for DNNs. Along with the model compression techniques, the data compression techniques such as the use of coresets, can be a way to reduce this limitation.

Despite these, we find a concrete use-case of our method, which is a transfer learning settings in robotics, where the number of data points are relatively small. Along with recent robot learning methods that attempt to work with the relatively small amount of data, e.g. meta learning, few-shot learning, etc, our paper suggests that advancing scalability of kernel methods such as GPs can be a promising direction of future research.

This is in comparison to the weight-space formulation of Bayesian Neural Networks (BNNs). To explain, consider an object detection scenario, where a developer has to create the labeled images for transfer learning. The NTK needs to then deal with the kernel scaling issues, which is associated with the number of data points, say 10000 images. An alternative is to deal with the scaling issues of BNNs in weight space, where the dimensions of network parameters are more than 1 million. Both approaches require various approximations to be applicable, but in many domains of robotics - due

to the limited amount of available data - dealing with the kernel scaling issue can be more practical than dealing with the approximations in the weight space (e.g. Bernoulli posterior in MC-dropout). In this regard, our experiments provide the relevance of this function-space view for robotics. By dealing with the challenges of the NTK to an extent, our real robot experiments show that the NTK can perform better than MC-dropout in terms of novelty detection and run-time.

We also stress that the proposed algorithm is only one way to exploit our theoretic insights. In this sense, our theoretic foundations can pave the ways towards different applications, beyond the use-cases of our algorithm. Finally, with our demonstrations, we hope the community to develop several algorithms, which can reduce the downside of GPs while leveraging the benefits of GPs for the predictive uncertainty estimation of DNNs.

References

[1] J. Lee, J. Feng, M. Humt, M. Müller, and R. Triebel. Trust your robots! predictive uncertainty estimation of neural networks with sparse gaussian processes. In *Proceedings of the Conference on Robot Learning*. PMLR, 08 Nov–11 Nov 2021.

[2] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.

[4] C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[5] Y. Bengio, O. Delalleau, and N. Le Roux. The curse of dimensionality for local kernel machines. Technical Report 1258, Département d'informatique et recherche opérationnelle, Université de Montréal, 2005.

[6] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pages 8571–8580, 2018.

[7] R. Novak, L. Xiao, Y. Bahri, J. Lee, G. Yang, D. A. Abolafia, J. Pennington, and J. Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019.

[8] B. Settles. Active learning. *Synthesis lectures on artificial intelligence and machine learning*, 6 (1):1–114, 2012.

[9] B. Settles. Active learning literature survey. 2009.

[10] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *ArXiv*, abs/1710.09282, 2017.

[11] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. *ArXiv*, abs/1506.02626, 2015.

[12] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *ArXiv*, abs/1412.6115, 2014.

[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *ICML*, 2015.

[14] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. 2011.

[15] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[16] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *ArXiv*, abs/1405.3866, 2014.

[18] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1131–1140, 2017.

[19] Y. Cheng, F. Yu, R. Feris, S. Kumar, A. Choudhary, and S. Chang. An exploration of parameter redundancy in deep networks with circulant projections. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2857–2865, 2015.

[20] F. N. Iandola, M. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *ArXiv*, abs/1602.07360, 2016.

[21] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[22] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2017.

[24] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NIPS*, 2014.

[25] A. Romero, N. Ballas, S. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2015.

[26] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[27] S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv: Computer Vision and Pattern Recognition*, 2016.

[28] R. M. Neal. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pages 29–53. Springer, 1996.

[29] M. E. E. Khan, A. Immer, E. Abedi, and M. Korzepa. Approximate inference turns deep networks into gaussian processes. In *Advances in Neural Information Processing Systems*, pages 3088–3098, 2019.

[30] J. Lee, Y. Bahri, R. Novak, S. Schoenholz, J. Pennington, and J. Sohl-dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[31] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[32] V. Tresp. Mixtures of gaussian processes. In *Advances in neural information processing systems*, pages 654–660, 2001.

[33] C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pages 881–888, 2002.

[34] H. Blum, A. Gawel, R. Siegwart, and C. Cadena. Modular sensor fusion for semantic segmentation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3670–3677. IEEE, 2018.

[35] O. Mees, A. Eitel, and W. Burgard. Choosing smartly: Adaptive multimodal fusion for object detection in changing environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 151–156. IEEE, 2016.

[36] S. Gross, M. Ranzato, and A. Szlam. Hard mixtures of experts for large scale weakly supervised vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6865–6873, 2017.

[37] D. J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[38] J. Martens and R. B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2408–2417, 2015.

[39] D. J. MacKay. Information-based objective functions for active data selection. *Neural computation*, 4(4):590–604, 1992.

[40] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pages 6402–6413, 2017.

[41] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[42] Z. Lu, E. Ie, and F. Sha. Uncertainty estimation with infinitesimal jackknife, its distribution and mean-field approximation. *CoRR*, abs/2006.07584, 2020.

[43] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 7576–7586. Curran Associates, Inc., 2018.

[44] G. Pleiss, J. Gardner, K. Weinberger, and A. G. Wilson. Constant-time predictive distributions for gaussian processes. In *International Conference on Machine Learning*, pages 4114–4123. PMLR, 2018.

[45] H. Ritter, A. Botev, and D. Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

[46] A. Y. Foong, Y. Li, J. M. Hernández-Lobato, and R. E. Turner. 'in-between'uncertainty in bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.

[47] M. Humt, J. Lee, and R. Triebel. Bayesian optimization meets laplace approximation for robotic introspection. *arXiv preprint arXiv:2010.16141*, 2020.

[48] K. Shinde, J. Lee, M. Humt, A. Sezgin, and R. Triebel. Learning multiplicative interactions with bayesian neural networks for visual-inertial odometry. *arXiv preprint arXiv:2007.07630*, 2020.

[49] J. Lee, M. Humt, J. Feng, and R. Triebel. Estimating model uncertainty of neural networks in sparse information form. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5702–5713. PMLR, 13–18 Jul 2020.

[50] J. Postels, F. Ferroni, H. Coskun, N. Navab, and F. Tombari. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2931–2940, 2019.

[51] A. Sharma, N. Azizan, and M. Pavone. Sketching curvature for efficient out-of-distribution detection for deep neural networks. *The Conference on Uncertainty in Artificial Intelligence (UAI)*, 2021.

[52] J. Wenger, H. Kjellström, and R. Triebel. Non-parametric calibration for classification. In *International Conference on Artificial Intelligence and Statistics*, pages 178–190. PMLR, 2020.

[53] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

[54] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 895–903, 2011.

[55] F. Meier, P. Hennig, and S. Schaal. Incremental local gaussian regression. In *Advances in Neural Information Processing Systems*, pages 972–980, 2014.

[56] P. Lutz, M. G. Müller, M. Maier, S. Stoneman, T. Tomić, I. von Bargen, M. J. Schuster, F. Steidle, A. Wedler, W. Stürzl, and R. Triebel. ARDEA - An MAV with skills for future planetary missions. *Journal of Field Robotics (JFR)*, 2019.

[57] J. Lee, R. Balachandran, Y. S. Sarkisov, M. De Stefano, A. Coelho, K. Shinde, M. J. Kim, R. Triebel, and K. Kondak. Visual-inertial telepresence for aerial manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1222–1229. IEEE, 2020.

[58] J. Lee, T. Muskardin, C. R. Pacz, P. Oettershagen, T. Stastny, I. Sa, R. Siegwart, and K. Kondak. Towards autonomous stratospheric flight: A generic global system identification framework for fixed-wing platforms. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6233–6240. IEEE, 2018.

[59] D. Feng, A. Harakeh, S. Waslander, and K. Dietmayer. A review and comparative study on probabilistic object detection in autonomous driving. *arXiv preprint arXiv:2011.10671*, 2020.