Vecchia Gaussian Process Ensembles on Internal Representations of Deep Neural Networks

Felix Jimenez
University of Wisconsin–Madison

Matthias Katzfuss
University of Wisconsin–Madison

Abstract

For regression tasks, standard Gaussian processes (GPs) provide natural uncertainty quantification (UQ), while deep neural networks (DNNs) excel at representation learning. Deterministic UQ methods for neural networks have successfully combined the two and require only a single pass through the neural network. However, current methods necessitate changes to network training to address feature collapse, where unique inputs map to identical feature vectors. We propose an alternative solution, the deep Vecchia ensemble (DVE), which allows deterministic UQ to work in the presence of feature collapse, negating the need for network retraining. DVE comprises an ensemble of GPs built on hidden-layer outputs of a DNN, achieving scalability via Vecchia approximations that leverage nearest-neighbor conditional independence. DVE is compatible with pretrained networks and incurs low computational overhead. We demonstrate DVE's utility on several datasets and carry out experiments to understand the inner workings of the proposed method.

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have achieved remarkable success in various tasks such as image recognition, natural language processing, and speech recognition. However, despite their excellent performance, these models have certain limitations, such as their lack of uncertainty quantification (UQ). Much of UQ for DNNs is based on a Bayesian approach that models network weights as random variables (Neal 1996) or involves ensembles of networks (Lakshminarayanan, Pritzel, and Blundell 2017), both requiring large memory and several evaluations of each test point. Deterministic single-forward-pass methods (Amersfoort et al. 2020; J. Liu et al. 2020; Mukhoti et al. 2023) have emerged to avoid the need to evaluate multiple networks. Deterministic methods require changes to the network training procedure to generate well-structured feature spaces for use with an outside model, often a Gaussian process.

Gaussian processes (GPs) provide natural UQ, but they are known to scale poorly with large datasets, which is addressed by myriad GP approximations. One such method is the Vecchia approximation (Vecchia 1988; Katzfuss and Guinness 2021), which uses nearest-neighbor conditioning sets to exploit conditional independence among the data. However, like all GP based methods, the Vecchia approximation requires that the input space is well structured. Therefore, naively applying a Vecchia GP with a deterministic method wouldn't address feature collapse.

The primary contribution of our paper is the introduction of the deep Vecchia ensemble (DVE), which leverages outputs from multiple internal DNN layers for deterministic uncertainty quantification without modifying network training. A high-level summary of our approach is given in Figure 1. Our method does not require changes to the standard DNN training regimen, unlike existing deterministic techniques. It offers interpretability by identifying training points similar to the query at test time, potentially enhancing performance in scenarios with limited data. Moreover, the DVE framework allows us to differentiate between aleatoric and epistemic uncertainty.

The remainder of the paper is organized as follows: We begin in Section 2 with a review of related concepts and define *intermediate representations*. In Section 3, we contextualize our approach within existing work on UQ for neural networks and GP approximation. The proposed methodology is detailed in Section 4. We then apply our model to pretrained neural networks

Proceedings of the 28[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

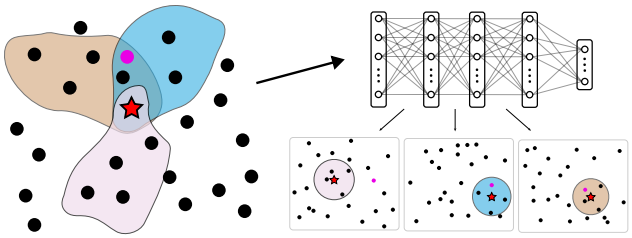

Figure 1: **Different layers imply different nearest neighbors**. Left: The input (red star) has different nearest-neighbor conditioning sets based on the metrics induced by the layers of the DNN with the magenta point being in two of the three conditioning sets. The brown, blue, and pink shaded areas denote the regions in input space that will be mapped to a hypersphere in the first, second, and third intermediate spaces, respectively. The conditioning sets derived from the different regions may overlap, as in the blue and brown region, or be disjoint from the others as in the pink region. Right: The labeled training data are propagated through the network and intermediate feature maps are stored. For a red test point, we assess uncertainty by considering and weighting instances in the training data that are similar to the test sequence in one or more of the feature maps.

in Section 5. This is followed by experiments in Section 6 that explore how our model works. In Section 7, we discuss potential application areas for our method, the limitations of our approach, and future research directions.

2 PRELIMINARIES

2.1 Modeling Functions with Gaussian Processes

Suppose we have a dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$, where $\boldsymbol{x}_i \in \mathcal{X}$, $y_i = f(\boldsymbol{x}_i) + \epsilon_i$, $\epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$, and $f : \mathcal{X} \to \mathbb{R}$. Our goal is to make predictions at n_p test locations $\boldsymbol{X}^* = \{\boldsymbol{x}_j^*\}_{j=1}^{n_p}$ in a probabilistic fashion. A common approach is to assign the latent function f a zero-mean GP prior with covariance function $K_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}')$ parameterized by $\boldsymbol{\theta}$. Letting $\boldsymbol{y}^* = \{f(\boldsymbol{x}_j^*) + \epsilon_j^*\}_{j=1}^{n_p}$, $p(\boldsymbol{y}^*, \boldsymbol{y})$ will be jointly multivariate normal, which implies:

$$p(\boldsymbol{y}^*|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{K}_{\boldsymbol{\theta}^*}^* + \sigma^2 \boldsymbol{I}_{n_p}), \tag{1}$$

where $\boldsymbol{\mu}^*$ and $\boldsymbol{K}_{\boldsymbol{\theta}^*}^*$ depend on $\mathcal{D}$ and $\boldsymbol{\theta}^*$ is selected to maximize the marginal log-likelihood:

$$p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f})d\boldsymbol{f} = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}_n). \tag{2}$$

Both Equations (1) and (2) have time and space complexities of $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively.

2.2 Approximating Gaussian Processes Using Vecchia

The joint distribution of a collection of random variables, $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$, can be decomposed as:

$$p(\boldsymbol{y}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{y}_{h(i)}),\ h(i) = \{1, \ldots, i-1\}$$

Vecchia's approximation (Vecchia 1988) conditions the i^{th} observation on a set indexed by $g(i)$, where $g(i) \subset h(i)$ with $|g(i)| \leq m$ for $m \ll n$. The approximate joint distribution is then, $p(\boldsymbol{y}) \approx \prod_{i=1}^n p(y_i|\boldsymbol{y}_{g(i)})$. This approximation reduces the in-memory space complexity of Equation (2) to $\mathcal{O}(nm)$. An additional approximation utilizing mini-batches (Jimenez and Katzfuss 2023; J. Cao et al. 2022) of size n_b results in an in-memory space complexity of $\mathcal{O}(n_b m^2)$ and a time complexity of $\mathcal{O}(n_b m^3)$.

For prediction, let $\tilde{\boldsymbol{y}} = (\boldsymbol{y}, \boldsymbol{y}^*)$, where $\boldsymbol{y}^* = \{y_{n+1}^*, \ldots, y_{n+n_p}^*\}$ are test points ordered after the observed data. The Vecchia approximation of the conditional distribution for $\boldsymbol{y}^*$ is, $p(\boldsymbol{y}^*|\boldsymbol{y}) \approx \prod_{j=n+1}^{n+n_p} p(y_j^*|\tilde{\boldsymbol{y}}_{g(j)})$, which reduces the time and space complexity of Equation (1) to $\mathcal{O}(n_p m)$.

For the i^{th} data point, the m ordered nearest neighbors $g(i)$ are typically determined based on a distance metric, such as Euclidean distance, between input variables $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. To compute these ordered nearest neighbors, it is assumed that the data has been sorted according to some ordering procedure, and for each i, the m nearest neighbors that precede i in the ordering are selected.

2.3 Ensembling Gaussian Processes

Consider L Gaussian processes meant to model the same function which we want to combine to form a single estimate. While we allow each GP to have its own hyperparameters, product-of-expert GP ensembles (Tresp 2000; Y. Cao and Fleet 2014; Deisenroth and Ng 2015; Rullière et al. 2018) typically share hyperparameters $\boldsymbol{\theta}$ between the GPs. We are assuming $\boldsymbol{\theta}_k$ contains all the hyperparameters in both the GP kernel and the likelihood.

The prediction of y^* at a location $\boldsymbol{x}^*$ can be formed by using a generalized product of experts (gPoE) (Y. Cao and Fleet 2014), an extension of product of experts (PoE) (Hinton 2002), which combines the predictions of L models by using a weighted product of the predictive densities. Letting $p_k(y^*)$ denote the k^{th} model's

predictive density for y^*, the combined distribution for y^* is given by, $p(y^*) \propto \prod_{k=1}^{L} p_k^{\beta_k(\boldsymbol{x}^*)}(y^*)$.

The function $\beta_k(\boldsymbol{x}^*)$ in the exponent acts as a weight for the k^{th} model's prediction based on the input $\boldsymbol{x}^*$ such that $\sum_{k=1}^{L} \beta_k(\boldsymbol{x}^*) = 1$. When each of the k predictions is a Gaussian distribution with mean $\mu_k(\boldsymbol{x}^*)$ and variance $\sigma_k(\boldsymbol{x}^*)$, the combined distribution is also Gaussian. The resulting mean and variance are given by $\mu(\boldsymbol{x}^*) = \sigma^2(\boldsymbol{x}^*) \sum_{k=1}^{L} \beta_k(\boldsymbol{x}^*)\sigma_k^{-2}(\boldsymbol{x}^*)\mu_k(\boldsymbol{x}^*)$ and $\sigma^2(\boldsymbol{x}^*) = (\sum_{k=1}^{L} \beta_k(\boldsymbol{x}^*)\sigma_k^{-2}(\boldsymbol{x}^*))^{-1}$, respectively. For details on computing β_k, see Appendix A.

Although we have focused on the combined predictive distribution for the observed y^*, in the case of a Gaussian likelihood the same results hold for the combined distribution of the latent function f^* (with the variance terms changed to reflect we are working with the noiseless latent function). For details on combining predictions in f or y space, see Appendix A.6.

2.4 Extracting Information from Intermediate Representations

Consider a finite sequence of functions $f_1, \ldots, f_{L+1}$ such that $f_1 : \mathbb{R}^d \to \mathcal{A}_1$, $f_2 : \mathcal{A}_1 \to \mathcal{A}_2$, $\ldots$, $f_L : \mathcal{A}_{L-1} \to \mathcal{A}_L$, $f_{L+1} : \mathcal{A}_L \to \mathbb{R}$, where each $\mathcal{A}_i \subset \mathbb{R}^{d_i}$.

The k^{th} **intermediate representation** of a point $\boldsymbol{x} \in \mathbb{R}^d$ with respect to the sequence $f_1, \ldots, f_{L+1}$ is defined to be $\boldsymbol{e}_k(\boldsymbol{x}) := (f_k \circ f_{k-1} \circ \cdots \circ f_1)(\boldsymbol{x})$. The k^{th} **intermediate space** $\mathcal{A}_k$ is the image of $(f_k \circ f_{k-1} \circ \cdots \circ f_1)(\cdot)$.

A **composite model** $\mathcal{M}$ is the composition $M(\boldsymbol{x}) := (f_{L+1} \circ f_{L-1} \circ \cdots \circ f_2 \circ f_1)(\boldsymbol{x})$. A feed-forward network with L layers is an example of a composite model, and the output of each layer in the network, excluding the final layer, results in an intermediate space.

2.5 Aleatoric and Epistemic Uncertainty

Consider a mapping $h : \mathcal{X} \to \mathcal{Y}$ and a tunable model $f_\theta : \mathcal{X} \to \mathcal{Y}$ aiming to approximate this mapping. Given a dataset $\mathcal{D}_n = \{x_i, y_i\}_{i=1}^n$ generated by h, we encounter two types of uncertainties in machine learning. *Aleatoric uncertainty* originates from the inherent noise in the process h, such as variability in data generation, and is irreducible. In contrast, *epistemic uncertainty* stems from the model f_θ, arising due to either model misspecification or insufficient data to accurately replicate h. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by refining the model or expanding $\mathcal{D}_n$. We aim to estimate the total uncertainty in predictions from f_θ, and to distinguish between aleatoric and epistemic components.

3 RELATED WORK

In this section, we review work on UQ for DNNs, relevant GP approximations, and methods that make use of the internal representations of a neural network. We finish by briefly mentioning the relation of our method to a few other related techniques.

For UQ with DNNs popular methods include ensembles of DNNs (Lakshminarayanan, Pritzel, and Blundell 2017) and Bayesian neural networks (BNNs) (Neal 1996). Ensemble methods train multiple DNNs on different data subsets, while BNNs treat network weights as random variables and use Bayesian inference to estimate a posterior distribution for these weights. Predictions in BNNs involve averaging over this posterior, thus accounting for weight uncertainty. Due to the computational complexity, approximations such as sampling or posterior approximation techniques are often employed (Neal 1996; Ritter, Botev, and Barber 2018; T. Chen, Fox, and Guestrin 2014; Gal and Ghahramani 2016; Maddox et al. 2019). Since our approach is based on a single network there is potential for integration our work with these existing frameworks.

Deterministic uncertainty quantification (UQ) methods offer an alternative to Bayesian neural networks (BNNs) and ensembles by estimating uncertainty through distances in feature space (Amersfoort et al. 2020; J. Liu et al. 2020; Van Amersfoort et al. 2021; Mukhoti et al. 2023). This requires the network's penultimate layer, $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, to be both sensitive to input perturbations and smooth, often enforced via a bi-Lipschitz constraint. A well-structured feature space enables the construction of a distance-aware predictive distribution $p(y|\boldsymbol{x})$ that performs well for both in-distribution and out-of-distribution (OOD) data. However, deep networks often suffer from feature collapse, where distinct inputs map to nearly identical features, limiting the reliability of uncertainty estimates (Amersfoort et al. 2020).

To address feature collapse, deterministic UQ methods modify network training using either two-sided gradient penalties (Amersfoort et al. 2020) or spectral normalization (SN) (J. Liu et al. 2020; Van Amersfoort et al. 2021; Mukhoti et al. 2023). Deterministic Uncertainty Quantification (DUQ) enforces a two-sided gradient penalty to train radial basis networks (Amersfoort et al. 2020). Spectral Normalized Gaussian Process (SNGP) instead applies SN during training and approximates a Gaussian process output layer using random Fourier features (RFF) and a Laplace approximation to the posterior of coefficients (J. Liu et al. 2020; J. Z. Liu et al. 2023). Similarly, Deterministic Uncertainty Quantification (DUQ) (Mukhoti et al. 2023) incorporates SN but uses Gaussian discriminant analysis for OOD detection and network logits for in-distribution uncer-

tainty estimates. Deterministic Uncertainty Estimation (DUE) (Van Amersfoort et al. 2021) also relies on SN, enforcing a bi-Lipschitz on the feature extractor used in deep kernel learning (DKL) (Wilson et al. 2016). While these methods improve uncertainty estimation, they assume control over network training, preventing their application to pretrained models.

Our approach, in contrast, builds on the study of neural collapse, a phenomenon closely related to feature collapse. In neural collapse, a classifier maps unique inputs to identical points based on class label. Recently it has been observed that neural collapse typically occurs at an intermediate layer of a network near the output (Rangamani et al. 2023). Given this insight, we develop a distance-aware predictive distribution using networks not strictly adhering to bi-Lipschitz smoothness.

Our distance-aware predictive distribution heavily relies on approximating GPs. We briefly review relevant GP approximations. The Vecchia approximation (Vecchia 1988; Katzfuss and Guinness 2021) enhances computational efficiency through ordered conditional approximations using nearest neighbors. Inducing-point methods (Hensman, Fusi, and N. D. Lawrence 2013; Titsias 2009), on the other hand, introduce a low-rank covariance matrix structure. Ensembling techniques improve scalability by distributing data across independent GPs, with ongoing research (Cohen et al. 2020; Rullière et al. 2018) dedicated to addressing and overcoming their inherent challenges.

A few notable methods utilize intermediate representations of neural networks. These include deep kernel learning (Wilson et al. 2016), which inputs the penultimate layer's output into a GP without addressing feature collapse—a key aspect for uncertainty quantification as illustrated in our S-curve example (Section 6.1). Deep k-nearest neighbors (DKNN) (Papernot and McDaniel 2018) assesses test point conformity for uncertainty measurement using intermediate representations and neighbor labels, rather than distance. Differing from the deep GP (Damianou and N. D. Lawrence 2013), our approach feeds network layer outputs into Vecchia GPs instead of cascading outputs from one GP to another. A particularly related approach uses Gaussian process probes (GPP) (Z. Wang et al. 2023), which define a prior distribution over classifiers that take activations as inputs. Conditioning on activation-label pairs yields a posterior over these classifiers, enabling assessment of both aleatoric and epistemic uncertainty. While similar to our work, GPP work on a single layer at a time. Combining our approach with GPP could extend DVE to classification tasks while allowing GPP to process multiple layers simultaneously.

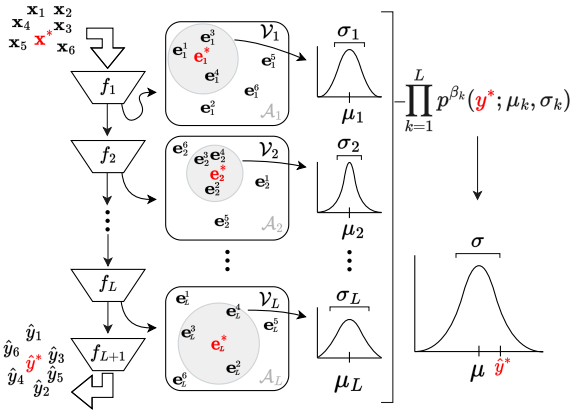

Figure 2: **The DVE pipeline is conceptually simple.** The network maps inputs $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_n$, to intermediate spaces, where we compute nearest neighbors. For example, in $\mathcal{A}_1$, the neighbors for x^* are $\boldsymbol{e}_1^1, \boldsymbol{e}_1^3, \boldsymbol{e}_1^4$ (for simplicity, we define $\boldsymbol{e}_k^i = \boldsymbol{e}_k(\boldsymbol{x}_i)$). For each layer, we define a Vecchia GP, denoted by $\mathcal{V}_1, \ldots, \mathcal{V}_L$, which estimates a distribution for the response y^*. Estimates are combined in a product-of-experts fashion to yield a single distribution parameterized by $\hat{\mu}$ and $\hat{\sigma}$, which improves upon the network's point prediction $\hat{y}^*$.

4 DEEP VECCHIA ENSEMBLE

We are provided with a dataset $\mathcal{D}$ comprising n input-output pairs, $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$, and a composite model $\mathcal{M} : \mathcal{X} \to \mathbb{R}$ that is trained to predict y_i given the corresponding $\boldsymbol{x}_i$. Initially, we map the inputs $\{\boldsymbol{x}_i\}_{i=1}^n$ to their respective intermediate representations, $\{\boldsymbol{e}_k(\boldsymbol{x}_i)\}_{i=1}^n$, where $k = 1, \ldots, L$. We generate a collection of conditioning sets, $\mathcal{G}_k$, for each set of intermediate representations, $\{\boldsymbol{e}_k(\boldsymbol{x}_i)\}_{i=1}^n$, by computing ordered nearest neighbors based on the Euclidean distance between the intermediate representations. The resulting tuple for each intermediate space $(\{\boldsymbol{e}_k(\boldsymbol{x}_i), \boldsymbol{y}_i\}_{i=1}^n, \mathcal{G}_k)$ is combined with an automatic relevance determination (ARD) kernel $K_{\boldsymbol{\theta}_k}(\cdot, \cdot)$ of the appropriate dimension to form an ensemble of deep Vecchia GPs.

At test time, the test point will be mapped to its intermediate spaces and each Vecchia GP will provide a predictive mean and variance. The predictions are then combined via a product of experts to form a single mean and covariance estimate. Figure 2 provides an overview of the proposed model.

4.1 Dataset Extraction from a Neural Network

To begin, we describe how we extract a sequence of datasets from a pretrained neural network and

the data it was trained on. Suppose the first layer of our pretrained network is of the form $f_1(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)})$, with, $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d\times d_1}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_1}$ and $\sigma_1 : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$. Then $\boldsymbol{x}$'s first intermediate representation, $\boldsymbol{e}_1(\boldsymbol{x}) := f_1(\boldsymbol{x})$, will lie in a subset of $\mathbb{R}^{d_1}$, that is $\boldsymbol{e}_1(\boldsymbol{x}) \in \mathcal{A}_1 \subset \mathbb{R}^{d_1}$. Similarly, $\boldsymbol{x}$'s second intermediate representation is given by, $\boldsymbol{e}_2(\boldsymbol{x}) := (f_2 \circ f_1)(\boldsymbol{x})$, where $f_2(\boldsymbol{e}_1(\boldsymbol{x})) = \sigma(\boldsymbol{e}_1(\boldsymbol{x})\boldsymbol{W}^{(2)} + \boldsymbol{b}^{(2)})$ with $\boldsymbol{W}^{(2)} \in \mathbb{R}^{d_1\times d_2}$, $\boldsymbol{b}_2 \in \mathbb{R}^{d_2}$ and $\sigma_2 : \mathbb{R}^{d_2} \to \mathbb{R}^{d_2}$. We can define the remaining $L-2$ intermediate representations in a similar way.

By combining the intermediate representations with the responses, we can generate L datasets $\{\mathcal{D}_k\}_{k=1}^L$, where $\mathcal{D}_k = \{\boldsymbol{e}_k(\boldsymbol{x}_i), y_i\}_{i=1}^n$. We assume the same random ordering of observations for all L datasets, so y_i will always refer to the same response for every $\mathcal{D}_k$.

An example of generating L datasets is visualized in Figure 3 for the bike UCI dataset. Each of the panels contains one dataset to which we fit a Vecchia GP.

Why do we need intermediate layers? Any last-layer method (Amersfoort et al. 2020; J. Liu et al. 2020; Mukhoti et al. 2023) needs to modify training to address feature collapse (Amersfoort et al. 2020). With multiple layers, distinct points in the input space will be distinct for at least one layer, as neural collapse tends to occur later in the network (Rangamani et al. 2023). The heterogeneity of inputs and corresponding responses for different layers also benefits prediction and UQ.

4.2 Conditioning-Set Selection

Figure 3 suggests that the function $\boldsymbol{e}_k(\boldsymbol{x}) \mapsto y$ is smooth for some value of k; therefore, we choose to estimate each $\boldsymbol{e}_k(\boldsymbol{x}) \mapsto y$ using a GP. For computational purposes, we approximate each of the GPs using Vecchia with nearest neighbors computed in the corresponding intermediate space. Doing so gives rise to several conditioning sets for each $\boldsymbol{x}_i$.

Letting $g_k(i)$ denote the conditioning set for observation i from intermediate space k, we consider the **collection of conditioning sets**, given by $\mathcal{G}_k := \{g_k(1), \ldots, g_k(n)\}$. An $L+1$ layer network will have L collections of conditioning sets, $\mathcal{G}_1, \ldots, \mathcal{G}_L$.

By using several collections of conditioning sets, we utilize the unique information present in the panels titled "First Intermediate Space" through "Fifth Intermediate Space" in Figure 3. This approach can lead to better predictions compared to using just one collection of conditioning sets.

Why not use inducing-point GPs? Our primary contribution is the use of intermediate-layer outputs for deterministic UQ, and inducing-point GPs could be used as a replacement for the Vecchia approximation in this situation.

What are the benefits of using the Vecchia approximation in particular? The primary benefits are interpretability and distinguishing between aleatoric and epistemic uncertainty. Interpretability is achieved by examining test-point conditioning sets, revealing the training examples deemed relevant by the network. Using nearest neighbors also allows for precise UQ: aleatoric uncertainty, reflected in the Gaussian likelihood's noise term, indicates data variability, while epistemic uncertainty is gauged by the test point's proximity to training instances. Notably, in ambiguous cases—such as Section 6.3—the conditioning sets pinpoint the source of uncertainty, a nuance that inducing points lack, as they cannot differentiate whether uncertainty stems from data sparsity, proximity to inducing inputs, or conflicting data interpretations.

4.3 Ensemble Building from Conditioning Sets

Each $\mathcal{G}_k$ represents a hypothesis about the conditional independence structure of the data. For instance, with time series data produced by an AR(2) process, the i^{th} observation should depend on the two preceding observations in time, while for an AR(1) process, the conditioning set is limited to the immediately preceding data point. However, in general, it is unclear which $\mathcal{G}_k$ is most appropriate.

Since we do not know which $\mathcal{G}_k$ is most suitable, we propose an ensemble of Vecchia GPs, each of which is based on one of the collections of conditioning sets. With each Vecchia GP independently fit to its corresponding dataset (e.g. $\mathcal{D}_k$), we combine the predictions from each member of the ensemble in a product-of-experts fashion. We combine our predictions in y-space; details on this choice are given in Appendix A.6. Additional details on training and prediction are provided in Appendix A.

5 APPLICATION TO PRETRAINED MODELS

We conduct experiments using pretrained networks across various tasks, comparing DVE to the original network and adjusting the choice of baselines depending on the task requirements. For the chemical property prediction task, we compare DVE with the base model and last-layer Gaussian process (GP) methods, as retraining the networks for these tasks is not feasible. In contrast, for the UCI regression tasks, we benchmark against ensembles, Monte Carlo (MC) dropout, and deep kernel learning, since retraining the networks is

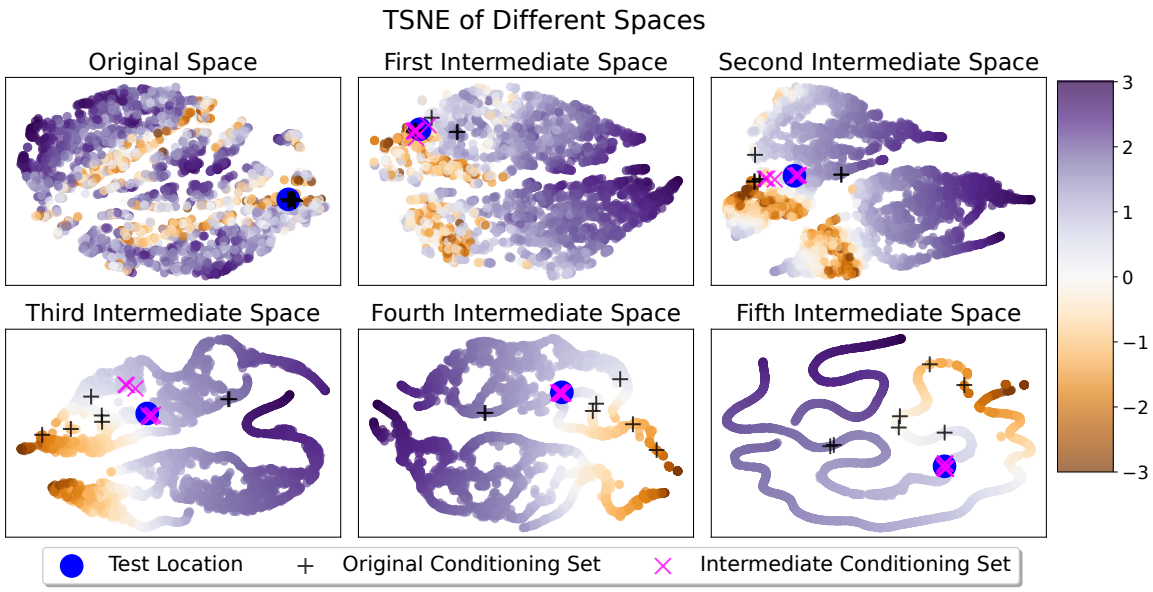

Figure 3: **Sensitivity and smoothness vary by layer**. The 2D TSNE projection of input points in the original space of the bike UCI dataset (top left panel) and intermediate spaces (remaining five panels) with the color denoting the response value. The blue dot is a fixed test point that has been propagated through the network. The black crosses denote the ordered nearest neighbors of the blue dot in the original input space, and their corresponding intermediate representations. The magenta crosses denote the ordered points nearest to the blue point within each intermediate space.

straightforward in this case. We measure performance using root-mean-square error (RMSE) and negative-log likelihood (NLL).

5.1 UCI Benchmarks

We train a feed-forward neural network on three train-validation-test splits of UCI regression tasks. After training, we construct the deep Vecchia ensemble (DVE) as in Section 4 and we make predictions for the test set using the method in Appendix A. Additionally, we train the stochastic variational inference GP (SVI-GP) of Hensman, Fusi, and N. D. Lawrence (2013), the deep Gaussian process (Deep GP) of (Damianou and N. D. Lawrence 2013), and the scaled Vecchia GP (Vecchia) of (Katzfuss, Guinness, and E. Lawrence 2022) separately on the training data for prediction on the test set.

Table 1 compares NLL (left column under each dataset) and RMSE (right column under each dataset) for different methods on UCI datasets. The bold value in each column is the best performing (lower is better). The value in parentheses is the sample standard deviation of the mean of the metric over the three different data splits. The last line of the table specifies the sample size (n) and dimension (d) for each dataset.

We can see that either DVE or the deep GP have the best NLL across the datasets, but the DVE is easier to work with given it is built on a pretrained network. Compared to the neural network, DVE is able to provide uncertainty estimates that are better than existing GP approximations, and DVE even improves upon the neural network's point predictions (in terms of RMSE) for three of the five datasets. For the full version of Table 1 and additional baseline comparisons using an alternative network see Appendix B.2.

5.2 Chemical Property Prediction

We applied our method to the Chemformer, as described by R. Irwin et al. (2022), initially pretrained on a 100 million SMILES string subset from the ZINC-15 dataset (Sterling and J. J. Irwin 2015) (details in Section 2.1 of R. Irwin et al. (2022)). This model was further fine-tuned on three MoleculeNet property prediction datasets—ESOL, FreeSolvation, and Lipophilicity—as used in R. Irwin et al. (2022). Details on data processing, fine-tuning, and deep Vecchia ensemble construction are provided in Appendix B.4.

Table 3 presents RMSE and NLL results. "Chemformer" refers to the original fine-tuned network. "DVE (m = n)" denotes the deep Vecchia ensemble with an exact GP (which is equivalent to a Vecchia approximation with $m = n$), while "DVE (m = 200)" and "DVE (m = 400)" use the Vecchia approximation with 200 and

Table 1: **DVE is consistently competitive with baselines**. NLL ↓ (left columns) and RMSE ↓ (right columns) on UCI benchmarks.

Method	Elevators		KEGG		bike		Protein	
DVE	**0.350 (0.010)**	0.345 (0.004)	-1.021 (0.030)	0.087 (0.002)	**-3.484 (0.054)**	**0.002 (0.000)**	**0.646 (0.029)**	**0.494 (0.014)**
Neural Net	N/A	**0.340 (0.003)**	N/A	**0.085 (0.002)**	N/A	0.006 (0.000)	N/A	0.514 (0.042)
Vecchia	0.525 (0.015)	0.414 (0.006)	-0.987 (0.026)	0.090 (0.002)	-0.239 (0.010)	0.183 (0.001)	0.851 (0.003)	0.571 (0.002)
SVI-GP	0.455 (0.017)	0.381 (0.007)	-0.912 (0.033)	0.098 (0.003)	-1.140 (0.047)	0.073 (0.004)	1.083 (0.001)	0.713 (0.001)
Deep GP (3)	0.366 (0.029)	0.346 (0.011)	**-1.036 (0.036)**	0.086 (0.004)	-2.806 (0.100)	0.004 (0.001)	0.938 (0.025)	0.597 (0.018)
(n,d)	(16.6K,18)		(48.8K,20)		(17.4K,17)		(45.7K,9)	

Table 2: **DVE improves performance of Chemformer**. RMSE and NLL for property prediction.

Dataset	Model	RMSE ↓	NLL ↓
ESOL	Chemformer	0.57 (0.03)	N/A
	DVE ($m = n$)	0.35 (0.01)	-2.02 (0.16)
	DVE ($m = 200$)	**0.35** (0.01)	**-2.21** (0.05)
FreeSolvation	Chemformer	2.23 (0.15)	N/A
	DVE ($m = n$)	**1.30** (0.09)	-1.45 (0.10)
	DVE ($m = 200$)	1.32 (0.09)	**-1.49** (0.10)
Lipophilicity	Chemformer	0.73 (0.02)	N/A
	DVE ($m = n$)	**0.60** (0.01)	-0.60 (0.11)
	DVE ($m = 400$)	0.61 (0.01)	**-0.63** (0.09)

400 nearest neighbors, respectively. Both the exact and Vecchia GP use the exact GP's hyperparameters to focus on the impact of a reduced conditioning set.

In all three datasets, both the exact and approximate GP in DVE reduced RMSE compared to the network alone. The exact GP showed only marginal RMSE improvement over the Vecchia approximation, which generally yielded better NLL. For the smaller ESOL and FreeSolvation datasets, we used exact nearest neighbors, while for the larger Lipophilicity dataset, we employed approximate nearest neighbors with product quantization Jegou, Douze, and Schmid 2010; Johnson, Douze, and Jégou 2017. The choice between exact and approximate nearest neighbors did not significantly impact the results.

6 EXPERIMENTS

In this section we explore DVE properties with the goal of understanding how our procedure works. We begin by looking at the intermediate representations of a network and then we check what the conditioning sets of a neural network look like for correct and incorrect predictions.

6.1 Exploring Intermediate Spaces

To investigate how DVE uses different layers we work with a feed-forward network fit to samples from the S-curve dataset (Tenenbaum, Silva, and Langford 2000). Figure 4 shows results and a basic description is given in the caption. For additional details, see Appendix B.3.

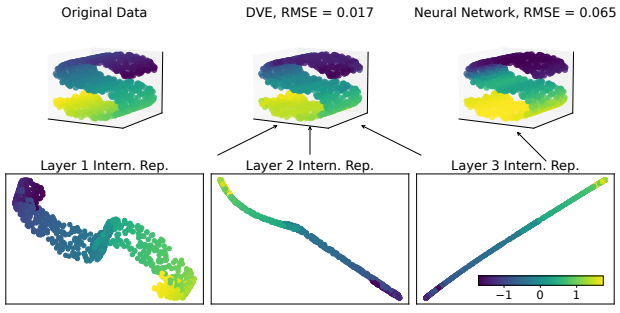

Figure 4: **Ensembling layers can outperform last layer in terms of MSE**. Top left: Data from the S-curve dataset. Top middle: Predictions generated by DVE. Top right: Predictions from the neural network. In all panels, the color denotes the response value. Arrows emphasize that DVE predictions use all three layers, whereas the final network prediction uses only the last layer.

After the first layer the green and yellow points are separated but these points collide by the final layer. DVE preserves this distinction, unlike the network. The DVE predictions closely match the original dataset, with a lower MSE of 0.018 compared to the neural network's 0.065.

Similar results hold for different networks. For example, with four hidden layers with (10, 5, 2, 2) units, the MSE dropped from 1.8e-1 to 4.2e-5. Changing the number of units per layer to (16, 8, 4, 4) showed the MSE drop from 7e-3 to 3.9e-5.

6.2 Interpreting Internal Representations

DVE can be seen from two perspectives: ensembling Vecchia GPs, and a method for UQ in DNNs. Below we relate both view points to the results in this section.

From the Vecchia ensemble perspective, we can see the intermediate representations in Figure 4 as providing three different distance metrics that can be used to compute possibly unique conditioning sets. The requirements to make this connection are formalized in Proposition 1.

Proposition 1. *Consider a sequence of injective func-*

tions $f_1, .., f_L, f_{L+1}$ *and a sequence of metric spaces* $\{(M_i, d_i)\}_{i=1}^{L+1}$ *such that* $f_i : \mathcal{M}_i \to \mathcal{M}_{i+1}$ *for* $i = 1, \ldots, L$. *Then* $\{(M_i, \tilde{d}_i)\}_{i=2}^{L+1}$ *defines a sequence of metric spaces, where* $\tilde{d}_i(x, y) := d_i(\boldsymbol{e}_i(\boldsymbol{x}), \boldsymbol{e}_i(\boldsymbol{x}'))$ *for any* $\boldsymbol{x}, \boldsymbol{x}' \in M_1$, *with* $\boldsymbol{e}_i(\boldsymbol{x}) := f_i \circ f_{i-1} \circ \cdots \circ f_1(\boldsymbol{x})$.

Requiring the functions be injective ensures unique internal representations for unique input points, but continuity is not required. Discontinuous layers are useful when modeling functions with high Lipschitz constants or discontinuities. In such cases, predicting values using Euclidean nearest neighbors in $\mathbb{R}^d$ can be inaccurate, but the function may be continuous with respect to a different distance metric on $\mathbb{R}^d$. The difference in continuity is reminiscent of what we observed in Figure 3, where the function became progressively smoother for later layers. Using multiple spaces allows for competing hypothesis about what distance metric is best.

From the UQ perspective, we can see the sequence of distance metrics as measuring how far a test point is away from instances in the training data. Of course, if a test point is close to existing training data with respect every distance metric, then the ensemble of Vecchia GPs will provide an estimate with small uncertainty. For data far from the training set, the internal representations will not match those of the training data and the prediction will be close to the GP prior.

If a network is experiencing feature collapse, will the sequence of functions be injective? In short, no, the entire sequence of functions may not be injective. However, there is likely a subsequence of the original sequence that remains injective. The DVE can handle non-injective functions effectively because of the posterior variance weighting we use. Specifically, when functions are non-injective, the posterior variance is dominated by output-scale noise, which accommodates the differing function values for unique inputs. As a result, the DVE assigns less weight to the non-injective functions, allowing it to approximate the two perspectives mentioned earlier. This property, along with the ablation studies in Appendix C.2, highlights the strong appeal of posterior variance weighting.

6.3 Inspecting Conditioning Sets

In this experiment, we train a feed-forward network on MNIST and generate embeddings from train data using two hidden layers. Then we select two test set instances—one correctly labeled and one incorrectly labeled by the network—and analyzed their embeddings.

As Figure 5 illustrates, the incorrectly labeled example's nearest neighbors in the first intermediate space are all nines, while in the second space, they are a mix of nines and fives. Conversely, the correctly labeled example's nearest neighbors are consistently eights in both spaces. The incorrectly labeled example is notably more distant from its nearest neighbors, being twice as far in the first embedding and five times in the second. This highlights how the network differently processes correctly and incorrectly classified examples, a phenomenon leveraged in our method. The ensemble mean acts as a weighted average of training labels, indicating layer agreement or providing a nuanced prediction when layers disagree.

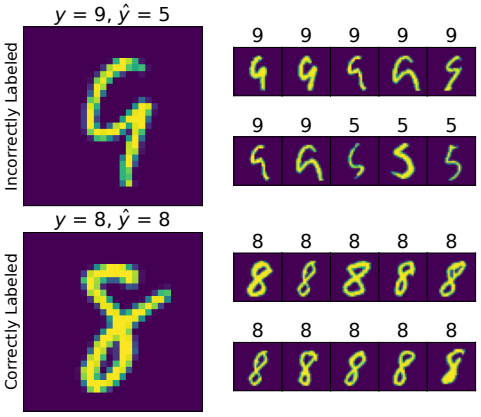

Figure 5: **Layerwise conditioning sets relate to aleatoric uncertainty**. Top left: A test image of a digit 9 incorrectly classified as a 5 by the neural network. Top right: Nearest-neighbor images for the misclassified 9 at the third (top row) and fourth (bottom row) layers of the network, with the top row showing neighbors of the correct class (9) and the bottom row indicating emergence of the incorrect class (5). Bottom half: For a correctly labeled test image of an 8 (bottom left), the nearest-neighbor images at both the third (top row) and fourth (bottom row) layers (bottom right) consistently show the correct class (8).

7 CONCLUSION

We introduced DVE for deterministic UQ with pretrained neural networks that does not require altering network training. The effectiveness of DVE was demonstrated across various tasks, including benchmark regression and chemical property prediction. Additionally, analyses of internal representations and conditioning sets offer valuable insights into the method's functionality.

Unlike existing deterministic methods (J. Liu et al. 2020; Amersfoort et al. 2020; Mukhoti et al. 2023), our

method works with any existing neural network without changing training, expanding the scope of deterministic UQ methods. The proposed method is relevant for latent-space Bayesian optimization (Gómez-Bombarelli et al. 2018), where it could streamline the process by eliminating the need for additional latent-space structuring, as seen in Grosnit et al. (2021).

Our model has three main limitations. Firstly, it requires the original training data, which may not always be accessible. A potential workaround is to fit the network alongside an ensemble of inducing-point GPs or use a smaller validation set for the deep Vecchia ensemble. Secondly, our approach combines predictions in y-space, limiting us to Gaussian likelihoods. This can be addressed by combining predictions in f-space and establishing a method to merge the noise terms from each Vecchia GP's likelihoods (see Appendix C.2.1). Finally, we utilize a DNN with fixed weights, whereas many BNN methods treat weights as random variables. Integrating DVE into BNNs could involve representing intermediate representations as distributions, using an error-in-variables Vecchia GP that combines Bachoc et al. (2023)'s kernel with Kang and Katzfuss (2023)'s correlation strategy.

References

Akiba, Takuya et al. (2019). "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631.

Amersfoort, Joost van et al. (2020). "Uncertainty estimation using a single deep deterministic neural network". In: *International conference on machine learning*. PMLR, pp. 9690–9700.

Bachoc, François et al. (2023). "Gaussian Processes on Distributions based on Regularized Optimal Transport". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 4986–5010.

Cao, Jian et al. (2022). "Scalable Gaussian-process regression and variable selection using Vecchia approximations". In: *The Journal of Machine Learning Research* 23.1, pp. 15799–15828.

Cao, Yanshuai and David J Fleet (2014). "Generalized product of experts for automatic and principled fusion of Gaussian process predictions". In: *arXiv preprint arXiv:1410.7827*.

Chen, Tianqi, Emily Fox, and Carlos Guestrin (2014). "Stochastic gradient hamiltonian monte carlo". In: *International conference on machine learning*. PMLR, pp. 1683–1691.

Cohen, Samuel et al. (2020). "Healing products of Gaussian process experts". In: *International Conference on Machine Learning*. PMLR, pp. 2068–2077.

Damianou, Andreas and Neil D Lawrence (2013). "Deep gaussian processes". In: *Artificial intelligence and statistics*. PMLR, pp. 207–215.

Deb, K. et al. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE Transactions on Evolutionary Computation* 6.2, pp. 182–197. DOI: 10.1109/4235.996017.

Deisenroth, Marc and Jun Wei Ng (2015). "Distributed gaussian processes". In: *International Conference on Machine Learning*. PMLR, pp. 1481–1490.

Gal, Yarin and Zoubin Ghahramani (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR, pp. 1050–1059.

Gardner, Jacob R et al. (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration". In: *Advances in Neural Information Processing Systems*.

Gómez-Bombarelli, Rafael et al. (2018). "Automatic chemical design using a data-driven continuous representation of molecules". In: *ACS central science* 4.2, pp. 268–276.

Grosnit, Antoine et al. (2021). "High-dimensional Bayesian optimisation with variational autoencoders and deep metric learning". In: *arXiv preprint arXiv:2106.03609*.

Hensman, James, Nicolo Fusi, and Neil D Lawrence (2013). "Gaussian processes for big data". In: *arXiv preprint arXiv:1309.6835*.

Hinton, Geoffrey E (2002). "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8, pp. 1771–1800.

Irwin, Ross et al. (2022). "Chemformer: a pre-trained transformer for computational chemistry". In: *Machine Learning: Science and Technology* 3.1, p. 015022.

Jegou, Herve, Matthijs Douze, and Cordelia Schmid (2010). "Product quantization for nearest neighbor search". In: *IEEE transactions on pattern analysis and machine intelligence* 33.1, pp. 117–128.

Jimenez, Felix and Matthias Katzfuss (2023). "Scalable bayesian optimization using vecchia approximations of gaussian processes". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 1492–1512.

– (2025). "Probabilistic Skip Connections for Deterministic Uncertainty Quantification in Deep Neural Networks". In: *arXiv preprint arXiv:2501.04816*.

Johnson, Jeff, Matthijs Douze, and Hervé Jégou (2017). "Billion-scale similarity search with GPUs". In: *arXiv preprint arXiv:1702.08734*.

Kang, Myeongjong and Matthias Katzfuss (2023). "Correlation-based sparse inverse Cholesky factorization for fast Gaussian-process inference". In: *Statistics and Computing* 33.3, p. 56.

Kang, Myeongjong, Florian Schäfer, et al. (2024). "Asymptotic properties of Vecchia approximation for Gaussian processes". In: *arXiv preprint arXiv:2401.15813*.

Katzfuss, Matthias and Joseph Guinness (2021). "A General Framework for Vecchia Approximations of Gaussian Processes". In: *Statistical Science* 36.1, pp. 124–141. DOI: 10.1214/19-STS755. URL: https://doi.org/10.1214/19-STS755.

Katzfuss, Matthias, Joseph Guinness, and Earl Lawrence (2022). "Scaled Vecchia approximation for fast computer-model emulation". In: *SIAM/ASA Journal on Uncertainty Quantification* 10.2, pp. 537–554.

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in neural information processing systems* 30.

Liu, Jeremiah et al. (2020). "Simple and principled uncertainty estimation with deterministic deep learning via distance awareness". In: *Advances in Neural Information Processing Systems* 33, pp. 7498–7512.

Liu, Jeremiah Zhe et al. (2023). "A simple approach to improve single-model deep uncertainty via distance-awareness". In: *Journal of Machine Learning Research* 24.42, pp. 1–63.

Maddox, Wesley J et al. (2019). "A simple baseline for bayesian uncertainty in deep learning". In: *Advances in neural information processing systems* 32.

Mukhoti, Jishnu et al. (2023). "Deep deterministic uncertainty: A new simple baseline". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 24384–24394.

Neal, Radford M (1996). *Bayesian learning for neural networks*. Vol. 118. Lecture Notes in Statistics. Springer New York, NY.

Papernot, Nicolas and Patrick McDaniel (2018). "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning". In: *arXiv preprint arXiv:1803.04765*.

Rangamani, Akshay et al. (2023). "Feature learning in deep classifiers through intermediate neural collapse". In: *International Conference on Machine Learning*. PMLR, pp. 28729–28745.

Ritter, Hippolyt, Aleksandar Botev, and David Barber (2018). "A scalable laplace approximation for neural networks". In: *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*. Vol. 6. International Conference on Representation Learning.

Rullière, Didier et al. (2018). "Nested Kriging predictions for datasets with a large number of observations". In: *Statistics and Computing* 28.4, pp. 849–867.

Salimbeni, Hugh and Marc Deisenroth (2017). "Doubly stochastic variational inference for deep Gaussian processes". In: *Advances in neural information processing systems* 30.

Stein, Michael L (2011). "2010 Rietz lecture: When does the screening effect hold?" In: *The Annals of Statistics* 39.6, pp. 2795–2819.

– (2015). "When does the screening effect not hold?" In: *Spatial Statistics* 11, pp. 65–80.

Sterling, Teague and John J Irwin (2015). "ZINC 15–ligand discovery for everyone". In: *Journal of chemical information and modeling* 55.11, pp. 2324–2337.

Tenenbaum, Joshua B, Vin de Silva, and John C Langford (2000). "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500, pp. 2319–2323.

Titsias, Michalis (2009). "Variational learning of inducing variables in sparse Gaussian processes". In: *Artificial intelligence and statistics*. PMLR, pp. 567–574.

Tresp, Volker (2000). "A Bayesian committee machine". In: *Neural computation* 12.11, pp. 2719–2741.

Van Amersfoort, Joost et al. (2021). "On feature collapse and deep kernel learning for single forward pass uncertainty". In: *arXiv preprint arXiv:2102.11409*.

Vecchia, Aldo V (1988). "Estimation and model identification for continuous spatial processes". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 50.2, pp. 297–312.

Wang, Zi et al. (2023). "Gaussian process probes (GPP) for uncertainty-aware probing". In: *Advances in neural information processing systems* 36, pp. 63573–63594.

Wilson, Andrew Gordon et al. (2016). "Deep kernel learning". In: *Artificial intelligence and statistics*. PMLR, pp. 370–378.

Yadan, Omry (2019). *Hydra - A framework for elegantly configuring complex applications*. Github. URL: https://github.com/facebookresearch/hydra.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes/No/Not Applicable] **Yes.**
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes/No/Not Applicable] **Yes.**
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes/No/Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes/No/Not Applicable] **Yes**
   (b) Complete proofs of all theoretical results. [Yes/No/Not Applicable] **Yes**
   (c) Clear explanations of any assumptions. [Yes/No/Not Applicable] **Yes**

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes/No/Not Applicable] **Yes**
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes/No/Not Applicable] **Yes**
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes/No/Not Applicable] **Yes**
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes/No/Not Applicable] **Yes**

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes/No/Not Applicable] **Yes**
   (b) The license information of the assets, if applicable. [Yes/No/Not Applicable] **Yes**
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes/No/Not Applicable] **Yes**
   (d) Information about consent from data providers/curators. [Yes/No/Not Applicable] **Yes**
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes/No/Not Applicable] **Yes**

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Yes/No/Not Applicable] **NA**
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Yes/No/Not Applicable] **NA**
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Yes/No/Not Applicable] **NA**

Supplementary Materials

Supplementary Material Contents:

A METHOD DETAILS

In this section, we provide details on the Deep Vecchia Ensemble (DVE) method proposed in Section 4. Specifically, Appendix A.1 provides details on our choice of covariance function, Appendix A.2 outlines the process of fitting the DVE, Appendix A.3 details the steps for predicting with the DVE, Appendix A.4 discusses the complexity of our proposed method, Appendix A.5 presents the proof of Proposition 1, and Appendix A.6 discusses how to combine the predictions from the layer-wise GPs of DVE.

A.1 Covariance Function

For each GP, we use an ARD Matérn 5/2 kernel, introducing $(d+1)$ parameters from the kernel that must be estimated. We select the Matérn kernel because it has a natural connection to the screening effect (Stein 2011), where spatial predictions primarily depend on observations closest to the prediction location. Briefly, the Matérn covariance function satisfies conditions that ensure an asymptotic screening effect holds (Stein 2011; Kang, Schäfer, et al. 2024), which is not true for the squared-exponential kernel (Stein 2015). While empirical evidence suggests the squared-exponential kernel performs well with the Vecchia approximation (Kang, Schäfer, et al. 2024), we chose the Matérn covariance due to its strong theoretical connection to the screening effect.

A.2 Training

For an $L+1$-layer network trained on a dataset with N observations, we begin by generating L datasets, where the inputs are the intermediate representations of the training inputs $\boldsymbol{x}$, and the responses are the same responses from the training data. That is, for each $k = 1, \ldots, L$, we have $\mathcal{D}_k = \{\boldsymbol{e}_k(\boldsymbol{x}i), y_i\}i = 1^N$. We will then independently fit a Vecchia Gaussian process to each dataset, $\mathcal{D}_k$.

This implies that for an $L+1$-layer network, we obtain an ensemble of L Vecchia GPs, each with its own set of parameters $\boldsymbol{\theta}_k = (\sigma_k^2, \boldsymbol{\lambda}_k, \tau_k)$, where σ_k^2 denotes the output variance for the kernel, $\boldsymbol{\lambda}_k \in \mathbb{R}_+^{d_k}$ represents the lengthscales for the ARD kernel, and τ_k^2 is the variance in the Gaussian likelihood. To estimate $\boldsymbol{\theta}_k$ for each Vecchia GP, we independently apply mini-batch gradient descent within type-II maximum likelihood, optimizing the Vecchia marginal log-likelihood using the corresponding dataset $\mathcal{D}_k$

$$\hat{\boldsymbol{\theta}}_k = \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\boldsymbol{y}_{g_k(i)}) \tag{3}$$

$$= \arg\max_{\boldsymbol{\theta}\in\Theta} \sum_{i=1}^{n} \log \mathcal{N}\left(y_i \mid \mu_i^{(k)}, \sigma_i^{(k)} + \tau_k^2 I\right). \tag{4}$$

The mean $\mu_i^{(k)}$ and the variance $\Sigma_i^{(k)}$ are given by,

$$\mu_i^{(k)} = \boldsymbol{K}_{i,\boldsymbol{g}} \left(\boldsymbol{K}_{\boldsymbol{g},\boldsymbol{g}} + \tau_k^2 I\right)^{-1} \boldsymbol{y}_{g_k(i)}$$
$$\sigma_i^{(k)} = \boldsymbol{K}_{i,i} - \boldsymbol{K}_{i,\boldsymbol{g}} \left(\boldsymbol{K}_{\boldsymbol{g},\boldsymbol{g}} + \tau_k^2 I\right)^{-1} \boldsymbol{K}_{\boldsymbol{g},i},$$

with $\boldsymbol{K}_{i,\boldsymbol{g}}$ denoting the covariance between $\boldsymbol{e}_k(\boldsymbol{x}_i)$ and $\boldsymbol{E}_{g_k(i)}$ where $\boldsymbol{E}_{g_k(i)} = \{\boldsymbol{e}_k(\boldsymbol{x}_j)\}_{j\in g_k(i)}$.

The conditional distribution $p_{\boldsymbol{\theta}_k}(y_i|\boldsymbol{y}_{g_k(i)})$ can be seen as using nearby observation, $\boldsymbol{y}_{g_k(i)}$, to predict y_i. This conditional distribution depends on the parameters $\boldsymbol{\theta}_k$, the intermediate representation, and the conditioning set. This model allows us to effectively capture the unique complexities present in each intermediate space of the neural network. Since we are optimizing the layer-wise Vecchia GPs independently, we ignore the hierarchical structure of the intermediate representations. Training the models independently is fundamentally different from a deep Gaussian process (Damianou and N. D. Lawrence 2013) as we do not feed the outputs of one GP into the next GP. However, by performing our training in this way, we can more easily work with existing pretrained models. Therefore, our procedure can be viewed as augmenting both the Vecchia GP and the pretrained model.

A.3 Prediction

For a test point $\boldsymbol{x}^*$, we compute the intermediate representation, $\boldsymbol{e}_k^* := \boldsymbol{e}_k(\boldsymbol{x}^*)$, and find the m nearest neighbors of $\boldsymbol{e}_k^*$ in the training dataset $\mathcal{D}_k$. We then compute the posterior mean μ_k^* and variance σ_k^* of f^*, where $y^*|f^* \sim \mathcal{N}(f^*, \sigma_k^2)$. We repeat this for each of the L layers. We then combine the L estimates into a single estimate as given below:

$$\mu(\boldsymbol{x}^*) = \sigma^2(\boldsymbol{x}^*) \sum_{k=1}^{L} \beta_k(\boldsymbol{e}_k^*)(\sigma_k^*)^{-2}\mu_k^* \tag{5}$$

$$\sigma^2(\boldsymbol{x}^*) = \sum_{k=1}^{L} \beta_k(\boldsymbol{e}_k^*)(\sigma_k^*)^{-2} \tag{6}$$

We chose this combining strategy to account for prediction uncertainty. Specifically, our approach implicitly considers the degree of agreement between the collections of conditioning sets for $\boldsymbol{x_i}$ at each layer. If there is high agreement across the layers, we expect the test point is well within the support of our training data and can make confident predictions. In contrast, if there is little agreement, we infer that $\boldsymbol{x_i}$ is a relatively novel observation and our predictive uncertainty should reflect this. The relative closeness of the test point and its conditioning set within each intermediate space also influences our confidence in the predictions. For instance, if our model has seen many data points similar to $\boldsymbol{x_i}$ at every layer, we would be more confident in our predictions than if our model had never encountered such points, even if the conditioning sets are consistent across all layers. Our product of experts combining strategy captures these behaviors effectively.

As mentioned in Section 2.3, we have the choice to combine the predictions in either f-space or y-space, and we investigate the performance of both approaches in Appendix A.6. We also investigate the impact of different ways of computing the β_k's in Appendix A.6. In short, we recommend combining methods in y-space using the differential entropy to compute the β_k's.

A.3.1 Layerwise Moments

Computing layerwise moments is the same for each layer so we focus on a general layer, k. The conditional mean and conditional variance for layer k for test point j is given by

$$\mu_i^{(k)} = \boldsymbol{K}_{i,\boldsymbol{g}} \left(\boldsymbol{K}_{\boldsymbol{g},\boldsymbol{g}} + \tau_k^2 I\right)^{-1} \boldsymbol{y}_{g_k(i)}$$
$$\sigma_i^{(k)} = \boldsymbol{K}_{i,i} - \boldsymbol{K}_{i,\boldsymbol{g}} \left(\boldsymbol{K}_{\boldsymbol{g},\boldsymbol{g}} + \tau_k^2 I\right)^{-1} \boldsymbol{K}_{\boldsymbol{g},i},$$

again with $\boldsymbol{K}_{i,\boldsymbol{g}}$ denoting the covariance between $\boldsymbol{e}_k(\boldsymbol{x}_i)$ and $\boldsymbol{E}_{g_k(i)}$ where $\boldsymbol{E}_{g_k(i)} = \{\boldsymbol{e}_k(\boldsymbol{x}_j)\}_{j \in g_k(i)}$.

A.4 Computational Complexity

Below we detail the runtime complexity of training and prediction using the DVE. We let m denote the conditioning set size and L denote the number of layers being ensembled.

A.4.1 Training

Since our approach trains the GP for each layer independently, it inherits the favorable complexity characteristics of the original Vecchia approximation. After computing the nearest neighbors for the training sets using an approximate nearest neighbor (ANN) method, the complexity for each training step is $\mathcal{O}(n_b m^3)$ per layer, where n_b is the number of training mini-batches and m is the size of the conditioning set. For the entire ensemble, the combined complexity for each training step is, $\mathcal{O}(L n_b m^3)$, where L is the number of layers used by the ensemble. To optimize efficiency, we precompute and store the intermediate representations for each layer on disk, allowing the base network to be queried only once per training point.

A.4.2 Prediction

For prediction, we pass a test point through the network once to compute its intermediate representations and then use ANN search at each layer to identify the conditioning sets. The prediction complexity after finding the approximate nearest neighbors is similarly $\mathcal{O}(L n_p m^3)$, where n_p is the number of test points.

A.4.3 Nearest Neighbors

The complexity associated with nearest neighbors can be adjusted by tuning the hyperparameters of the ANN algorithm so we excluded it when discussing the training and test complexity. Specifically, we use an Inverted File Index (IVF) approach for computing nearest neighbors, where the data are divided into N_{list} clusters. Queries involve searching the nearest N_{probe} clusters to compute nearest neighbors. Increasing N_{list} improves speed at the cost of accuracy, while increasing N_{probe} enhances accuracy but reduces speed. This trade-off allows control over the complexity of the nearest neighbor computation. We have observed that large N_{list} values and small N_{probe} values can still produce high-quality results. For a more thorough analysis of ANN's impact on Vecchia GPs, we refer to Jimenez and Katzfuss (2023).

A.5 Proof of Proposition 1

Let $f : M_1 \to M_2$ be an injective function where (M_1, d_1) and (M_2, d_2) denote metric spaces. Now define $\tilde{d}_1(\boldsymbol{x}, \boldsymbol{x}') := d_2(f(\boldsymbol{x}), f(\boldsymbol{x}'))$ and let $\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z} \in M_1$.

Note that $\tilde{d}_1(\boldsymbol{x}, \boldsymbol{x}') = d_2(f(\boldsymbol{x}), f(\boldsymbol{x}')) = d_2(f(\boldsymbol{x}'), f(\boldsymbol{x})) = \tilde{d}_1(\boldsymbol{x}', \boldsymbol{x})$.

Additionally, $\tilde{d}_1(\boldsymbol{x}, \boldsymbol{x}') = d_2(f(\boldsymbol{x}), f(\boldsymbol{x}')) \geq 0$ for any $\boldsymbol{x}, \boldsymbol{x}' \in M_1$.

Now suppose $\tilde{d}_1(\boldsymbol{x}', \boldsymbol{x}) = 0 \Leftrightarrow d_2(f(\boldsymbol{x}'), f(\boldsymbol{x})) = 0 \Leftrightarrow f(\boldsymbol{x}') = f(\boldsymbol{x}) \Leftrightarrow \boldsymbol{x}' = \boldsymbol{x}$.

Finally, observe $\tilde{d}_1(\boldsymbol{x}, \boldsymbol{x}') = d_2(f(\boldsymbol{x}), f(\boldsymbol{x}')) \leq d_2(f(\boldsymbol{x}), f(\boldsymbol{z})) + d_2(f(\boldsymbol{z}), f(\boldsymbol{x}')) = \tilde{d}_1(\boldsymbol{x}, \boldsymbol{z}) + \tilde{d}_1(\boldsymbol{z}, \boldsymbol{x}')$.

This establishes that $\tilde{d}_1$ is a proper distance metric on M_1. Now because the composition of injective functions is injective, then each $\tilde{d}_i$ will be a proper distance metric, which proves the result.

A.6 Combining Predictions

This section details how to combine the predictions from the Vecchia GPs fitted to different intermediate spaces. We begin with our recommendation, followed by a discussion of the key components. Experimental results supporting this can be found in Appendix C.2.

A.6.1 Recommendation

We recommend combining predictions in y-space using the "Posterior Variance" method in Table 7, without the softmax function. This is effective when ensemble members have different noise terms, and no clear method exists for combining them. We also suggest exploring f-space combination by finding a principled approach to handling noise terms, especially for extending results to non-Gaussian likelihoods.

A.6.2 Combining Strategies

When combining predictions from the ensemble of Deep Vecchia GPs, we must decide whether to combine in y-space (as in J. Liu et al. (2020)) or f-space (as in Cohen et al. (2020) and Deisenroth and Ng (2015)), how to compute the unnormalized weights $\beta_k(\boldsymbol{x}^*)$, and whether to use a softmax function for normalizing the weights. If using softmax, a temperature parameter T also needs to be selected.

A.6.3 Weighting Schemes for GP Ensembles

There are multiple methods for computing β_k. For example, the softmax-based approach from Cohen et al. (2020) sets $\beta_k(\boldsymbol{x}^*) \propto \exp(-T\psi_k(\boldsymbol{x}^*))$, where T controls sparsity and $\psi_k(\boldsymbol{x}^*)$ measures the model's confidence. Without softmax, $\beta_k \propto \psi_k(\boldsymbol{x}^*)$, which aligns with product of experts methods. Various choices of $\psi_k(\boldsymbol{x}^*)$ lead to different weighting schemes, such as: $\psi_k(\boldsymbol{x}^*) = 0$ (standard PoE), $\psi_k(\boldsymbol{x}^*) = \sigma_k^2(\boldsymbol{x}^*)$ (higher weight for lower variance), and $\psi_k(\boldsymbol{x}^*) = \frac{1}{2}(\log\sigma^2 - \log\sigma_k^2(\boldsymbol{x}^*))$ (differential entropy weighting from Y. Cao and Fleet (2014)). Another option is to use the Wasserstein distance between the prior and posterior Cohen et al. (2020).

B EXPERIMENT DETAILS

This section contains details for the experiments we conducted. This includes details on computing infrastructure, hyperparameter, data splits, baseline implementations and choices specific to each experiment.

B.1 Environment and Hyperparameters

All experiments were designed to run efficiently with minimal resources on a machine running Ubuntu 22.04, equipped with an AMD Ryzen 9 5950X, a single NVIDIA 3090 Ti, and 64 GB of RAM. Larger hyperparameter sweeps were conducted on an HTCondor cluster, which consists of various node types and GPUs, providing additional computational power.

Hyperparameters were selected by minimizing either RMSE or jointly minimizing RMSE and NLL on the validation set. For each experiment and model combination, hyperparameter optimization was performed using a budget of 20-200 (depending on the problem/time). For single objective optimization we used a Tree Parzen Estimator as implemented in the Optuna (Akiba et al. 2019) plugin for the Python package Hydra (Yadan 2019). Likewise, for multiobjective optimization we used the Optuna implementation of NSGA-II (Deb et al. 2002). The hyperparameters chosen were those that achieved the highest value of validation RMSE in the single objective tasks. For the multiobjective tasks, we preferred hyperparameters that resulted in better NLL, but never to the extend that RMSE significantly decreased. For any multistage process, we optimized the hyperparameters at each stage independently and set them for any subsequent search.

Table 3: The feed forward neural network from the UCI regression tasks.

Network Parameters	
Neurons Per Hidden Layer	512, 128, 64, 32, 16
Activation	SELU
Training Hyperparameters	
Hyperparameters Optimized	learning rate, α-drop out, weight decay

B.2 UCI Regression

Below we provide details on the UCI regression experiment. We begin with the network that is used in Table 1 and then provide details on the deep GP baseline.

B.2.1 Network and Training

The network used is identical for each UCI regression problem presented in the main text and the weights and activation function are given in Table 3. For each dataset we chose a batchsize and number of epochs such that the training time was about the same for each task. For three random seeds we then performed train-validation-test splits with 64%, 16% and 20%, respectively. Using the first split we chose around 200 random configurations of the hyperparameters in the bottom of Table 3 to train the network on the training data. We evaluated the MSE on the validation set and chose the the hyperparameters that gave the lowest MSE on the validation set. The weights, splits and hyperparameters were then stored for each of the three random seeds.

B.2.2 Deep GP Details

For the deep GP we used three layer networks. Each hidden layer had dimension d_x, where d_x is the dimension of the input for that problem. For example, $d_x = 8$ for kin40k.

To approximate the posterior of the deep GP we used the method of Salimbeni and Deisenroth (2017), as implemented in GPyTorch (Gardner et al. 2018). This inducing point based approach uses mini-batches as well as Monte Carlo to estimate the ELBO. In all instances, we used 100 inducing points per layer and five samples from the variational distribution to esimate the log-likelihood term in the ELBO. We used an RBF automatic relevance determination (ARD) kernel for each GP.

During training we used a batch size of 1024 for every dataset and set the number of epochs so that training time was between 20-50 minute from start to finish. We used an Adam optimizer with an initial learning rate chosen through a simple grid search for fixed epoch and batch size. The learning rate was decreased during training using cosine annealing with a final learning rate set to be 1e-9.

The inputs and outputs were scaled to have unit variance and zero mean. During training we observed that the validation RMSE would often plateau for long periods and then suddenly improve. To avoid this we found it was useful to initialize the inducing points by drawing from a standard normal and setting every value whose magnitude was less than one to zero. Additionally, we used a Gamma(9,0.5) prior for all the lengthscales (we have used the shape/rate parameterization of the Gamma distribution). During evaluation, we centered and sclaed the test data using moments calculated on the training data.

B.3 S-Curve

In this experiment we use a feedforward network with three hidden layers that each have two units followed by a SeLU activation function. We utilized the S-curve dataset comprised of 1000 samples (Tenenbaum, Silva, and Langford 2000) which can be seen as a 1D curve that is embedded in 3D space. This dataset is illustrated in the top left panel of Figure 4. Each hidden layer of the network was made up of two units that were followed by the SeLU activation function. The three intermediate representations can be visualized by plotting them directly. We colored the points by the response value in the experiment, and the resulting plot is displayed in the three panels at the bottom of the figure. The top row's remaining two panels show the predictions (and RMSE)

from the Deep Vecchia Ensemble and the neural network, respectively. The Deep Vecchia Ensemble utilized all three intermediate representations to form predictions, while the network only used the final intermediate representation. Using the Deep Vecchia Ensemble resulted in an improvement in RMSE.

B.4 Chemical Property Prediction

Below we provide details on data Processing and fine-tuning for the chemical property prediction tasks in Section 5.2.

B.4.1 Data Processing

In this experiment we use 10 train, validation, and test splits. For each of the 10 train, validation and test splits we followed the same process. To be precise we followed the data processing steps described in Section 2.3 of R. Irwin et al. (2022) as closely as possible. Below is a bulleted list of the steps taken to prepare the data for training. The steps are nearly identical to those presented in R. Irwin et al. (2022) but with a few extra details provided.

- Convert each SMILES string into its canonical form and find all instances which occur in more than one of the three datasets and place these into the training set.
- Split the remaining observations such that the train, validation and test splits make up 75%, 10% and 15% of the dataset, respectively.
- For each dataset, scale the target value such that the training set has values in zero to one. Scale the validation and test target values by the same amounts.
- Up-sample the ESOL and FreeSolvation datasets by a factor of 2 and 3, respectively. This is only done in the training set used for fine-tuning.

B.4.2 Fine-Tuning

For fine-tuning we used the finetuning procedure given in the Chemformer github repo: https://github.com/MolecularAI/Chemformer. We used the default hyper-parameters and chose the model weights such that the validation loss was minimized. The results obtained are similar to those reported in Table 3 of R. Irwin et al. (2022), except for the Free Solvation results. Consistently, for all 10 splits the Free Solvation RMSE we obtained was higher than that reported in the original Chemformer paper.

B.4.3 Changing SMILES and Forming the Ensemble

In this problem the SMILES strings are combined with a token that denotes the property being predicted. As an example "`C=CC(C)C`" would become "`<FreeS>|C=CC(C)C`" for a SMILES string from the FreeSolvation dataset. Then the processed strings are fed through several transformer encoder layers. Now since each SMILES string is potentially a different length, we take the intermediate representation of each encoder layer to be the length 512 vector whose position corresponds to the property being predicted. We now have a network with L layers with well defined intermediate representations and we can proceed to construct a deep Vecchia ensemble as described in Section 4.

C ADDITIONAL EXPERIMENTAL RESULTS

In this section we present additional experimental results which supplement experiments in the main text or present experiments not found in the main text.

C.1 UCI Regression

This section contains experimental results that relate to the UCI regression tasks presented in Section 5.1. This includes more results related to Table 1, additional baselines (e.g. NN-ensembles) and other ablation studies.

Table 4: **DVE is consistently competitive with baselines**. NLL ↓ (left columns) and RMSE ↓ (right columns) on UCI benchmarks using SELU activation. Upper half represents single pass methods while the lower half represents multiple pass methods.

Dataset (n,d)	Method	NLL ↓	RMSE ↓
kin40k (40K,8)	DVE	-1.445 (0.011)	0.057 (0.001)
	Neural Net	N/A	0.069 (0.000)
	Vecchia	-0.028 (0.015)	0.209 (0.015)
	SVI-GP	-0.093 (0.008)	0.068 (0.005)
	Deep GP (3)	**-1.664 (0.027)**	**0.030 (0.001)**
Elevators (16.6K,18)	DVE	**0.350 (0.010)**	0.345 (0.004)
	Neural Net	N/A	**0.340 (0.003)**
	Vecchia	0.525 (0.015)	0.414 (0.006)
	SVI-GP	0.455 (0.017)	0.381 (0.007)
	Deep GP (3)	0.366 (0.029)	0.346 (0.011)
KEGG (48.8K,20)	DVE	-1.021 (0.030)	0.087 (0.002)
	Neural Net	N/A	**0.085 (0.002)**
	Vecchia	-0.987 (0.026)	0.090 (0.002)
	SVI-GP	-0.912 (0.033)	0.098 (0.003)
	Deep GP (3)	**-1.036 (0.036)**	0.086 (0.004)
bike (17.4K, 17)	DVE	**-3.484 (0.054)**	**0.002 (0.000)**
	Neural Net	N/A	0.006 (0.000)
	Vecchia	-0.239 (0.010)	0.183 (0.001)
	SVI-GP	-1.140 (0.047)	0.073 (0.004)
	Deep GP (3)	-2.806 (0.100)	0.004 (0.001)
protein (45.7K,9)	DVE	**0.646 (0.029)**	**0.494 (0.014)**
	Neural Net	N/A	0.514 (0.042)
	Vecchia	0.851 (0.003)	0.571 (0.002)
	SVI-GP	1.083 (0.001)	0.713 (0.001)
	Deep GP (3)	0.938 (0.025)	0.597 (0.018)

C.1.1 Complete UCI Results for SELU Network

Table 4 is the complete version of Table 1 from the main text, with the protein dataset included. It is worth noting that the results in Table 4 are for a network using a SELU activation and the results in Table 5 are for a network using ReLU activations.

C.1.2 Additional UCI Baselines

This experiment aims to answer the following:

- Does DVE work if we change the network?
- How does DVE compare with Monte Carlo dropout, deep ensembles, deep kernel learning?

To answer these questions we repeat the UCI regression task on a subset of the datasets from Table 4 but we replace the SELU activation with a ReLU activation and compare with these alternative baselines. NLL and RMSE results for this experiment are shown in Table 5. In the Table "MC Dropout (20)" denotes that we use 20 passes through the network for each observation. Likewise, "Ensemble (5)" denotes an ensemble built using five networks. The subscript for DKL indicates the spectral norm bound used, while the absence of a subscript denotes no spectral normalization. As before the metrics are computed using centered and scaled responses, so the values in Table 5 and directly comparable to the values in Table 4.

We can see from Table 5 that simply adding the DVE on an existing network gives results competitive to ensembles. Stated differently, ensembling layers of a single network can be as effective as ensembling different networks. However, with DVE we don't have to worry about inducing diversity among the models as we simply take the best deterministic network we can find, decompose the layers and fit the intermediate GPs. Furthermore, adding spectral normalization alone to the feature extractor of our DKL variant improves RMSE and NLL compared to the baseline DKL variant. However, DKL-SN does not outperform DVE in terms of NLL on these examples. This outcome is expected, as the network lacks residual connections, preventing the feature extractor from behaving like the constrained feature extractor in a model such as DUE (Van Amersfoort et al. 2021). The constrained feature extractor of DUE is crucial for fully leveraging DKL, whereas DVE allows us to bypass this requirement.

Table 5: **Adding DVE on top of a network consistently achieves competitive performance compared to baselines**. RMSE and NLL for UCI regression tasks using ReLU activation with various baselines for comparison.

Dataset	Model	NLL ↓	RMSE ↓
kin40k	Neural Net	N/A	0.044 (0.001)
	DVE	**-1.810 (0.007)**	0.046 (0.001)
	DKL	-1.511 (0.005)	0.044 (0.001)
	DKL-SN$_{1.0}$	-1.400 (0.013)	0.054 (0.001)
	DKL-SN$_{2.0}$	-1.544 (0.002)	**0.042 (0.000)**
	DKL-SN$_{3.0}$	-1.533 (0.007)	0.043 (0.001)
	DKL-SN$_{4.0}$	-1.535 (0.003)	**0.042 (0.000)**
	Ensemble (5)	-1.716 (0.026)	**0.042 (0.002)**
	MC Dropout (20)	-1.653 (0.019)	0.052 (0.001)
bike	Neural Net	N/A	0.014 (0.000)
	DVE	**-4.469 (0.060)**	0.004 (0.001)
	DKL	-2.851 (0.056)	0.009 (0.001)
	DKL-SN$_{1.0}$	-1.178 (0.353)	0.051 (0.004)
	DKL-SN$_{2.0}$	-3.001 (0.017)	**0.003 (0.000)**
	DKL-SN$_{3.0}$	-2.989 (0.017)	0.005 (0.000)
	DKL-SN$_{4.0}$	-2.936 (0.033)	0.007 (0.001)
	Ensemble (5)	-3.516 (0.234)	0.010 (0.000)
	MC Dropout (20)	-2.543 (0.090)	0.016 (0.001)

Table 6: **DVE's performance matches or exceeds that of the best individual layer-wise model in the ensemble, sidestepping the need to identify the optimal layer.**. RMSE and NLL for UCI regression tasks using ReLU activation, comparing DVE to the individual layer-wise models that are combined to form DVE.

Dataset	Model	NLL ↓	RMSE ↓
kin40k	DVE	**-1.810 (0.007)**	**0.046 (0.001)**
	Vecchia GP$_1$	-0.857 (0.085)	0.090 (0.004)
	Vecchia GP$_2$	-1.575 (0.039)	0.053 (0.001)
	Vecchia GP$_3$	-1.723 (0.022)	0.048 (0.002)
	Vecchia GP$_4$	-1.726 (0.068)	0.047 (0.003)
bike	DVE	-4.469 (0.060)	0.004 (0.001)
	Vecchia GP$_1$	-4.361 (0.153)	**0.003 (0.001)**
	Vecchia GP$_2$	-4.423 (0.083)	0.005 (0.002)
	Vecchia GP$_3$	**-4.504 (0.049)**	0.004 (0.002)
	Vecchia GP$_4$	-4.454 (0.117)	0.003 (0.001)

C.1.3 DVE Outperforms Individual Layers

This experiment compares the performance of the individual layer-wise models used to construct DVE with DVE itself, which ensembles these layers. We use the same DVE model as in Table 5, but now we compare it to the layer-wise Vecchia GPs placed at intermediate layers.

The results are presented in Table 6, where Vecchia GP$_1$ refers to the Vecchia GP model that utilizes the first intermediate representation (i.e., the first hidden layer). We see that DVE performs favorably compared to the individual layer-wise models across both datasets. Notably, the best-performing individual layer can vary depending on whether RMSE or NLL is considered. In contrast, DVE consistently delivers strong performance for both metrics while sidestepping the challenge of selecting the optimal layer.

C.1.4 Intermediate Layers Preserve Information

This experiment examines how intermediate representations preserve information relevant to estimating epistemic uncertainty, even when that information is lost in the final layer. We build on the binary classification task from Amersfoort et al. (2020), where two-dimensional points are classified based on the sign of their first coordinate. A network trained on this task should ignore perturbations in the second coordinate, treating out-of-distribution points similarly to training data. However, in active learning, we often want to identify and label such points to reduce epistemic uncertainty, creating a tension between classification accuracy and sensitivity to novel points.

We reuse the implementation from Jimenez and Katzfuss (2025), including their data generation and plotting code, with slight modifications to the network architecture. The network consists of layers with dimensions [8, 8, 4, 4, 4, 4], residual connections, and Leaky-ReLU activations throughout. The dataset contains $N = 2000$

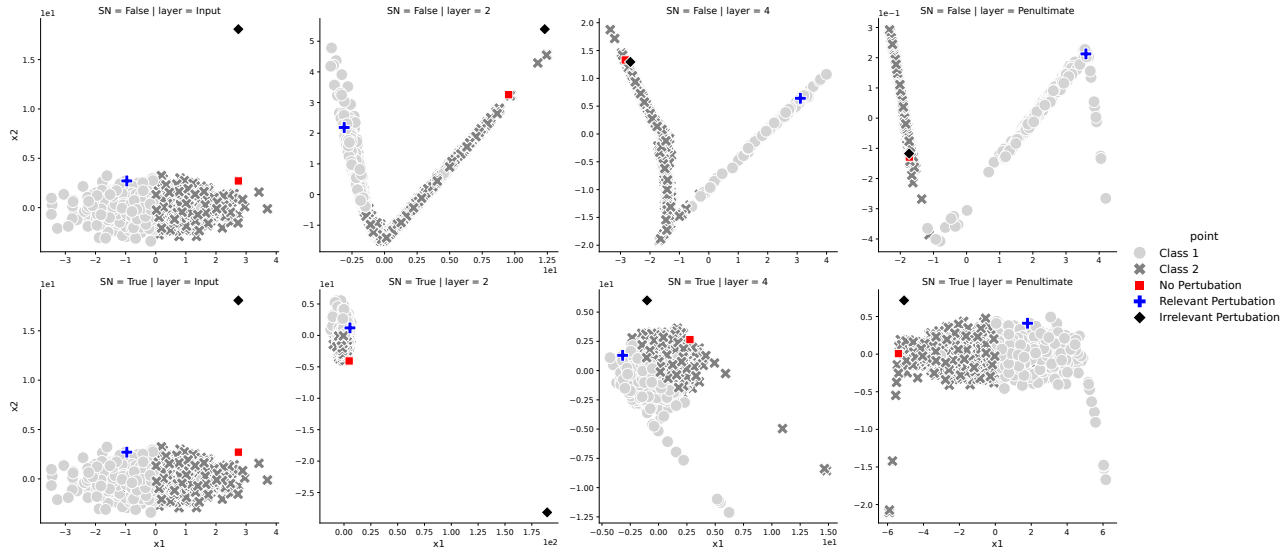

Figure 6: **Without spectral normalization, intermediate layers are sensitive to perturbations irrelevant to the class label**, as shown in the second panel from the right in the top row. This sensitivity persists even though later layers in the same network collapse the representation of the irrelevant perturbation to resemble in-distribution data. In contrast, with spectral normalization, the perturbation is preserved throughout the network (bottom row). However, DVE can leverage the earlier layers, enabling us to work with networks that are not trained with spectral normalization.

samples of two-dimensional points ($d = 2$) drawn from a Gaussian distribution with variance $\sigma = 0.3$. The labels are determined by the sign of the first coordinate ($y = \text{sign}(x_1)$), with label noise introduced by flipping labels with probability $\epsilon_p = 0.005$. The test set is generated similarly but without noise.

We train two networks, one with spectral normalization and one without, and examine their intermediate representations by projecting them onto the first two principal components. We visualize the projections using the full test set and two perturbations of a test point: one class-relevant (blue) and one irrelevant (red), which perturbs the second coordinate.

The results, shown in Figure 6, demonstrate that without spectral normalization, the irrelevant perturbation collapses to resemble in-distribution points, whereas with spectral normalization, the point remains distinguishable from in-distribution data even in the penultimate layer. Early layers in the network without spectral normalization (e.g., layer 1) preserve information about the perturbation, suggesting they help estimate epistemic uncertainty.

A similar observation was made in Jimenez and Katzfuss (2025), which showed that intermediate representations, especially from the penultimate layer of networks with spectral normalization, can distinguish in- and out-of-distribution data.

C.2 Combining Layer-Wise Predictions

In this section, we empirically investigate how to optimally combine the predictions from the layer-wise GPs in the DVE. As discussed in Appendix A.6, there are several design choices involved in combining the predictions from the layer-wise Vecchia GPs, and this study serves to guide us toward our final recommendation.

To assess the impact of each design choice, we measure the DVE's validation RMSE and NLL on two datasets from the UCI regression tasks: bike and kin40k. In each setting, we begin by fitting the layer-wise GPs to the same training data that was used to train the network. We then use the validation set to generate GP predictions for each layer and then combine them using all meaningful combinations of the options listed in Table 7. For example, we combine Wasserstein weighting in f-space using softmax, with the temperature set to 3. For each combination, then form the combined prediction and measure performance. Training is performed on the training set, and results are evaluated on the validation set.

Table 7: Different options for combining Deep Vecchia Ensemble predictions. Weigthing formulas correspond to unnormalized weights.

Component	Options
Weighting	Uniform $\beta_k(\boldsymbol{x}_*) = 1$ Posterior Variance $\beta_k(\boldsymbol{x}_*) = 1/\sigma_j^2(\boldsymbol{x}_*)$ Differential Entropy $\beta_k(\boldsymbol{x}_*) = 0.5(\sigma_{**}^2 - \sigma_j^2(\boldsymbol{x}_*))$ Wasserstein Dist. $\mu_j(\boldsymbol{x}_*)^2 + (\sigma_{**}^2 - \sigma_j^2(\boldsymbol{x}_*))^2$
Combining space	f-space / y-space
softmax	True / False
Temperature (T)	1, 3, 5

The RMSE and NLL values for both bike and kin40k when combining without softmax are shown in Figure 7. The RMSE and NLL values when using softmax for bike and kin40k are presented in Figure 9 and Figure 8, respectively. Below, we analyze the results in relation to various concepts and discuss how our observations compare to existing literature on combining model predictions.

C.2.1 Combining in f-Space or y-Space

From Figure 7, Figure 9 and Figure 8 we see that the NLL is lower when combining in y-space. This holds for every combination of combining method, combining space and temperature value. While the story is not as clear when considering RMSE, the results are comparable when combining in y-space or f-space. Therefore, we suggest combining in y-space when using a simple average of the noise terms from each Vecchia GP. However, these results will certainly change if the noise terms from each Vecchia GP are combined in a more principled way, and especially if the noise terms influence the weight being assigned to each GP in the ensemble. A complete investigation of combining in f-space for the deep Vecchia ensemble would likely result in further improvement over what is presented here.

These results are unique because, for PoE models relying on disjoint partitions of the data, it has been shown that combining in f-space can be preferable. Additionally, for non-Gaussian likelihoods it necessary to combine in f-space as combining in y-space may result in intractable inference (Cohen et al. 2020). However, the effect of combining in y-space vs. f-space is different here than what was observed in Cohen et al. (2020), as we do not use GPs on disjoint partitions of the data. Additionally, each Vecchia GP likelihood has its own value for the observation noise. For simplicity we take an average of these observation noise terms, but a more principled approach would likely improve the results of combining in f-space. We do not address this issue here and simply compare combining in y-space to combining in f-space with a simple average of observation noise terms.

C.2.2 Softmax or No Softmax

For the bike dataset, the NLL and RMSE values for "Posterior Variance" in Figure 7 are as good as any any NLL and RMSE value given in Figure 9 (considering the error bars of each plot). This implies there is no value of T for the softmax that outperforms no softmax on the bike dataset. The same observation holds when comparing kin40k results without softmax (Figure 7) to those using softmax (Figure 8). Furthermore, without the use of a softmax function we do not need to choose the value of the hyperparameter T. Therefore, for our work we did not use the softmax function. We believe that similar results may may hold for other datasets when using the deep Vecchia ensemble and combining in f-space as we have. Again these results may change if the Veccchia GP noise terms are combined differently, but we do not investigate that here.

Figure 7: The RMSE and NLL (lower is better) for Deep Vecchia ensemble using different combining methods in both f-space (red) and y-space (blue) without using the temperature endowed softmax. The left column shows the results on bike and the right columns show the results on kin40k. The vertical bars on each plot repersent one standard deviation (from three repeats). In terms of NLL, combing in y-space has better results for every combining method. The results for RMSE are not as clear.

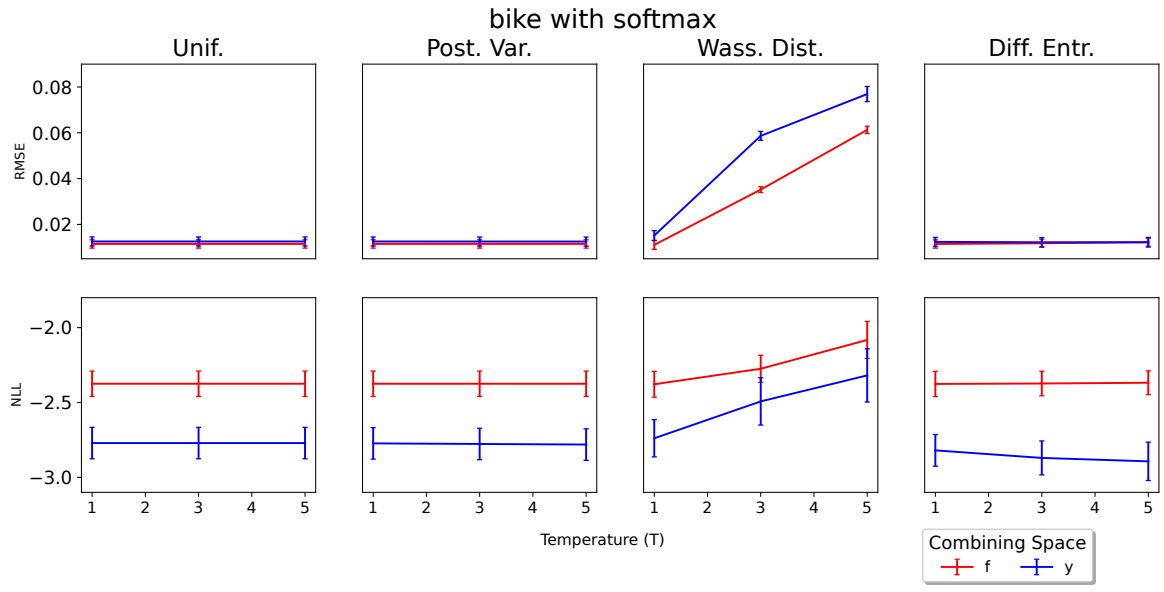

Figure 8: The RMSE and NLL vs temperature (T) for Deep Vecchia ensemble using different combining methods in both f-space (red) and y-space (blue). The results shown are all for the **bike** dataset. The vertical bars on each plot represent one standard deviation (from three repeats). It appears that increasing the value of T had meaningful change on the NLL of Wasserstein distance and differential entropy, but not on posterior variance or uniform weighting. For RMSE only Wasserstein distance changed as we varied T.

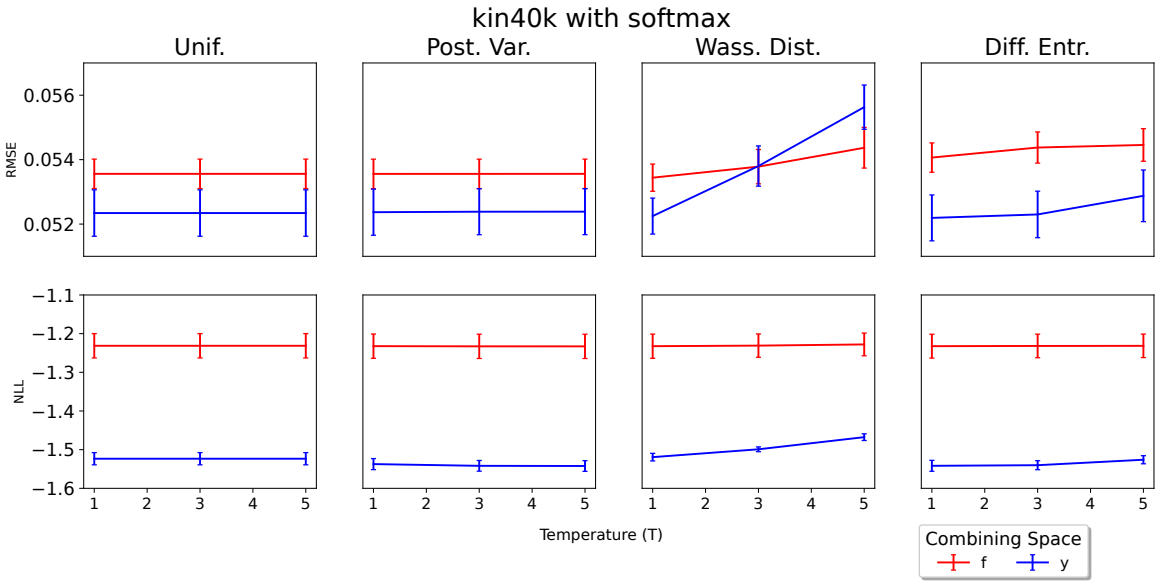

Figure 9: The RMSE and NLL vs temperature (T) for Deep Vecchia ensemble using different combining methods in both f-space (red) and y-space (blue). The results shown are all for the kin40k dataset. The vertical bars on each plot represent one standard deviation (from three repeats). Just as in Figure 9 NLL changed for Wasserstein distance and differential entropy as we varied T, but not for other methods. RMSE for Wasserstein distance and differential entropy were the only two that changed as we increased T.