From Deep Additive Kernel Learning to Last-Layer Bayesian Neural Networks via Induced Prior Approximation

Anonymous Author
Anonymous Institution

Abstract

With the strengths of both deep learning and kernel methods like Gaussian Processes (GPs), Deep Kernel Learning (DKL) has gained considerable attention in recent years. From the computational perspective, however, DKL becomes challenging when the input dimension of the last-layer GP is high. To address this challenge, we propose the Deep Additive Kernel (DAK) model, which incorporates i) an additive structure for the last-layer GP; and ii) induced prior approximation for each GP component. This naturally leads to a last-layer Bayesian neural network (BNN) architecture. The proposed method enjoys the interpretability of DKL as well as the computational advantages of BNN. Empirical results show that the proposed approach outperforms state-of-the-art DKL methods in both regression and classification tasks.

1 INTRODUCTION

Deep Neural Networks (DNNs) (LeCun et al., 2015) are powerful tools capable of capturing intricate patterns in large datasets, and have demonstrated remarkable performance across a wide range of tasks. However, DNNs are prone to overfitting on small datasets, offer limited interpretability and transparency, and lack the ability to provide uncertainty estimation. On the other hand, as a traditional kernel-based method, Gaussian processes (GPs) (Williams and Rasmussen, 2006) are robust against overfitting. In addition, they naturally enable uncertainty quantification and offer greater interpretability and flexibility of incorporating prior knowledge. Damianou and Lawrence (2013) proposed Deep Gaussian Processes (DGPs) by stacking multiple layers of GPs, which introduces the hierarchical structure of deep learning with the probabilistic, non-parametric nature of GPs. Although the deep structure of DGPs allows them to learn features at different levels of abstraction, tuning and optimization of DGPs, particularly for large datasets, can be difficult due to the layered structure, requiring careful consideration of hyperparameters and approximation techniques.

To learn rich, hierarchical representations from data with proper interpretability and uncertainty estimation, Wilson et al. (2016a) introduced Deep Kernel Learning (DKL) by incorporating DNNs into the last-layer kernel methods. This hybrid model enhances both flexibility and scalability, making it well-suited for a variety of real-world tasks, including regression, classification, and active learning, especially in scenarios where uncertainty quantification is critical.

One of the main challenges in DKL arises from the GP layer, which requires $\mathcal{O}(N^3)$ training time for N data points, limiting its scalability for large datasets. To address this issue, several GP approximation methods have been developed, including Random Fourier Features (RFF) (Rahimi and Recht, 2007), Stochastic Variational GP (SVGP) (Titsias, 2009; Hensman et al., 2015), and Kernel Interpolation for Scalable Structured GP (KISS-GP) (Wilson and Nickisch, 2015), and these have been incorporated into DKL models (Wilson et al., 2016b; Xue et al., 2019; Xie et al., 2019). Another challenge in DKL, as identified by Ober et al. (2021), is the tendency to overcorrelate inputs to minimize the complexity penalty term in the marginal likelihood. To address this, Ober et al. (2021) introduced a fully Bayesian approach using Markov Chain Monte Carlo (MCMC), however, this method scales poorly for high-dimensional posteriors. In response, Matias et al. (2024) proposed Amortized Variational DKL (AVDKL), leveraging NN-based representation learning to compute the inducing locations and variational parameters using input-dependent sparse GPs

Preliminary work. Under review by AISTATS 2025. Do not distribute.

(Jafrasteh et al., 2022), thus attenuating the overcorrelation of NN output features. However, these DKL models still directly combine approximated GPs with NNs via inducing points, which remains inefficient due to the dependency of inducing variables when computing the Evidence Lower Bound (ELBO).

In this work, we propose the **D**eep **A**dditive **K**ernel (DAK) model, which embeds hierarchical features learnt from NN into additive GPs and models the last-layer GP as a Bayesian neural network (BNN) layer (MacKay, 1992; Harrison et al., 2024) with a sparse kernel activation via an induced prior approximation on designed grids (Ding et al., 2024). The proposed methodology jointly trains the variational parameters of the last layer and the deterministic parameters of the feature extractor by maximizing the variational lower bound. This hybrid architecture enjoys the interpretability of DKL as well as the computational advantages of BNN, and also leads to a closed-form ELBO and predictive distribution for regression tasks, bypassing the Monte Carlo sampling during inference and training. Our contributions are as follows:

- We introduce a DAK model that reinterprets the deep additive kernel learning as a last-layer BNN via induced prior approximation. The proposed DAK enjoys the interpretability of DKL as well as the computational advantages of BNN, and is flexible to generalize to various DL tasks.
- We derive closed-form expressions of both the predictive distribution in inference and the ELBO during training for regression tasks, eliminating the need for sampling and reducing computational complexity to linear in the size of the induced grid.
- In experimental studies, we demonstrate that the proposed DAK model outperforms state-of-the-art DKL methods in both regression and classification tasks, while also mitigating the overfitting issue.

The remainder of this paper is organized as follows: Section 2 introduces the background on GPs, additive GPs, and DKL. In Section 3, we present the proposed DAK model and discuss its computational complexity compared to other state-of-the-art DKL models, followed by related work in Section 4. We present the experimental results in Section 5, and conclude the paper in Section 6.

2 PRELIMINARIES

GPs. A GP $f(\cdot) \sim \mathcal{GP}(\mu(\cdot), k(\cdot,\cdot))$ is completely specified by its mean function $\mu(\cdot)$ and covariance (kernel) function $k(\cdot,\cdot)$. Given a dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, where $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^N$ are training points and $\mathbf{y} = (y_1, \ldots, y_N)^\top$ is the corresponding observation where $y_i \in \mathbb{R}$, the standard procedure of GPs assumes $\mu(\cdot)$ is constant function, assigned to zero, and considers $y_i = f(\mathbf{x}_i) + \epsilon_i$ with Gaussian noise $\epsilon_i \sim \mathcal{N}(0, \sigma_f^2)$ for $i = 1, \ldots, N$. The predictive posterior $\mathbf{f}_* := f(\mathbf{X}^*)$ at N_* test points $\mathbf{X}^* := \{\mathbf{x}_i^* \in \mathbb{R}^D\}_{i=1}^{N_*}$ can also be expressed in closed form as a Gaussian distribution:

$$\mathbf{f}_* | \mathbf{X}, \mathbf{y}, \mathbf{X}^* \sim \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \tag{1}$$

$$\boldsymbol{\mu}_* = \mathbf{K}_{\mathbf{X}^*,\mathbf{X}} \left(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_f^2 \mathbf{I}\right)^{-1} \mathbf{y}, \tag{2}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{\mathbf{X}^*,\mathbf{X}^*} - \mathbf{K}_{\mathbf{X}^*,\mathbf{X}} \left(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_f^2 \mathbf{I}\right)^{-1} \mathbf{K}_{\mathbf{X},\mathbf{X}^*}, \tag{3}$$

where $\mathbf{K}_{\mathbf{X},\mathbf{X}'}$ denotes the kernel matrix with $[\mathbf{K}_{\mathbf{X},\mathbf{X}'}]_{ij} := k(\mathbf{x}_i, \mathbf{x}'_j)$. The common challenges of GPs are the $\mathcal{O}(N^3)$ computational complexity of inference and the "curse of dimensionality" with high-dimensional data. We refer the readers to Williams and Rasmussen (2006) for more details.

Additive GPs. Duvenaud et al. (2011) introduced the additive GP model, which allows additive interactions of all orders, ranging from first-order interactions all the way to D-th order interactions. In this work, we consider the simplest case where the additive GP is restricted to the first order with the same base kernel for each unit $d \in \{1, \ldots, D\}$:

$$f(\mathbf{x}) = \sigma \sum_{d=1}^{D} g_d(x_d) + \mu(\mathbf{x}), \tag{4}$$

where x_d is the d-th feature of the point $\mathbf{x} \in \mathbb{R}^D$, $g_d(x_d) \sim \mathcal{GP}(0, k_d(x_d, x'_d))$ is the *centered* base GP unit. The resulting $f(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$ is a GP specified by mean function $\mu(\mathbf{x})$ and additive kernel $k_{\text{add}}^{[D]}(\mathbf{x}, \mathbf{x}') = \sigma^2 \sum_{d=1}^{D} k_d(x_d, x'_d)$, where σ^2 is the variance assigned to all first-order interactions. Delbridge et al. (2020) showed that additive kernels projecting to a low dimensional setting can match or even surpass the performance of kernels operating in the original space. However, an additive kernel alone offers no computational advantage, as the cubic complexity remains. As we will show shortly, the additive GP perspective allows us to apply an induced prior approximation, which reduces computational costs and leads to a last-layer Bayesian representation.

Deep Kernel Learning. The performance of GPs is limited by the choice of kernel. DKL (Wilson et al., 2016a) attempts to solve this problem by DNNs, of which the structural properties can model high-dimensional data and large datasets well. DKL first learns feature transformations $h_\psi(\cdot)$ of DNNs with the parameters ψ. The outputs of DNNs are then used as inputs to a GP layer $\mathcal{GP}(\mu_\theta(\cdot), k_\theta(\cdot,\cdot))$ with

the parameters θ resulting in the effective kernel $k_{\text{DKL}}(\mathbf{x}, \mathbf{x}') = k_\theta(h_\psi(\mathbf{x}), h_\psi(\mathbf{x}'))$ to learn uncertainty representation provided by GPs. The complete set of parameters $\{\psi, \theta\}$ in DNNs and GPs are trained jointly by maximizing the marginal log-likelihood (MLL).

3 DAK: Deep Additive Kernel

In GPs, the kernel function defines the similarity between data points, playing a crucial role in the model's predictions. However, choosing the right kernel for complex, high-dimensional data can be challenging. DKL addresses this by learning the kernel function directly from data using a DNN. In practice, the outputs of DNNs can still be too complex to scale GPs, e.g. multi-task regression, or image recognition. Therefore, we present the Deep Additive Kernel as a last-layer BNN that can efficiently reduce GPs to single-dimensional ones without loss in performance.

Model. We present the construction of the DAK using the proposed additive GPs. Let $h_\psi : \mathbb{R}^D \to \mathbb{R}^P$ be a neural network, and we consider a total of P *centered* one-dimensional base GPs $g_p \sim \mathcal{GP}(0, k_p(\cdot,\cdot))$ with Laplace kernel $k_p(x_p, x_p') = \exp\left(-|x_p - x_p'|/\theta_p\right)$ for $p = 1, \ldots, P$. The forward pass of deep additive model $f(\cdot) : \mathbb{R}^D \to \mathbb{R}$ with Laplace kernel for each base GP at N data points $\mathbf{X}$ is described as follows:

$$f(\mathbf{X}) = \sum_{p=1}^{P} \sigma_p g_p\Big(\underbrace{h_\psi^{[p]}(\mathbf{X})}_{:=\mathbf{H}_p \in \mathbb{R}^N} \Big) + \mu, \tag{5}$$

where $\mathbf{H}_p := h_\psi^{[p]}(\mathbf{X}) \in \mathbb{R}^N$ is the p-th feature vector of neural network representations $h_\psi(\mathbf{X}) \in \mathbb{R}^{N \times P}$, and $\mu \in \mathbb{R}$ is a constant prior mean placed on $f(\cdot)$. The resulting deep additive kernel can be written as:

$$\begin{aligned} k_{\text{DAK}}(\mathbf{x}, \mathbf{x}') &= k_{\text{add}}^{[P]}(h_\psi(\mathbf{x}), h_\psi(\mathbf{x}')) \\ &= \sum_{p=1}^{P} \sigma_p^2 k_p\Big(\underbrace{h_\psi^{[p]}(\mathbf{x})}_{h_p \in \mathbb{R}}, \underbrace{h_\psi^{[p]}(\mathbf{x}')}_{h_p' \in \mathbb{R}} \Big). \end{aligned} \tag{6}$$

The corresponding coefficients $\{\sigma_p\}_{p=1}^P$ help learn a correlated contribution of each base kernel, and high-dimensional features of neural network $h_\psi(\mathbf{X})$ can be solved in a single-dimensional space $\mathbb{R}$ to achieve scalability. The GP hyperparameters can be inherently optimized with DNN parameters ψ without additional computational cost in our method. The details of reparameterization and optimization tricks are deferred when we discuss **Inference** and **Training** shortly.

Induced Prior Approximation. An essential ingredient of the proposed method is to apply a reduced-rank approximation to the prior, i.e., the base GPs. This approximation is referred to as the induced prior approximation (Ding et al., 2024), given by

$$\begin{aligned} \tilde{g}_p(\cdot) &:= \mathbf{K}_{(\cdot),\mathbf{U}} \mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1} g_p(\mathbf{U}) \\ &= \underbrace{\mathbf{K}_{(\cdot),\mathbf{U}} \left[\mathbf{L}_{\mathbf{U}}^\top\right]^{-1}}_{:=\phi(\cdot) \in \mathbb{R}^{1 \times M}} \underbrace{\mathbf{L}_{\mathbf{U}}^{-1} g_p(\mathbf{U})}_{:=\mathbf{z}_p \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_M)} \\ &= \phi(\cdot)\, \mathbf{z}_p, \end{aligned} \tag{7, 8}$$

where $\mathbf{U} = \{u_i \in \mathbb{R}\}_{i=1}^M$ denotes the induced grids (see details in Appendix A), $\mathbf{L}_{\mathbf{U}} \in \mathbb{R}^{M \times M}$ is a lower triangular matrix derived by the Cholesky decomposition of $\mathbf{K}_{\mathbf{U},\mathbf{U}} = \mathbf{L}_{\mathbf{U}} \mathbf{L}_{\mathbf{U}}^\top$, and $\mathbf{z}_p = \mathbf{L}_{\mathbf{U}}^{-1} g_p(\mathbf{U}) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_M)$ are i.i.d. standard Normal random variables. The main idea of induced prior approximation (eq. (7)) is to use the GP regression (see eq. (2)) to reconstruct the prior GP. According to the theory of GP regression (Yakowitz and Szidarovszky, 1985; Wang et al., 2020), $\tilde{g}_p$ converges to the original prior as $\mathbf{U}$ becomes dense in its domain.

The standard Cholesky decomposition of a dense matrix requires $O(M^3)$ time. However, by leveraging the Markov property of the Laplace kernel, the decomposed matrix $\mathbf{L}_{\mathbf{U}}^{-1}$ becomes sparse if $\mathbf{U}$ is designed by a one-dimensional dyadic point set with increasing order (Ding et al., 2024). As a consequence, the complexity of Cholesky decomposition can be reduced to $\mathcal{O}(M)$. Details of obtaining the sparse Cholesky factors are deferred to Appendix A.

Applying the sparsely induced GP approximation in eq. (8) together with the deep additive model in eq. (5), we obtain the final DAK:

$$\begin{aligned} \tilde{f}(\mathbf{X}) &= \sum_{p=1}^{P} \sigma_p \tilde{g}_p\left(h_\psi^{[p]}(\mathbf{X})\right) + \mu \\ &= \sum_{p=1}^{P} \sigma_p \Big(\phi(\mathbf{H}_p)\mathbf{z}_p\Big) + \mu, \end{aligned} \tag{9}$$

where $\phi(\mathbf{H}_p) := \mathbf{K}_{\mathbf{H}_p,\mathbf{U}} \left[\mathbf{L}_{\mathbf{U}}^\top\right]^{-1} \in \mathbb{R}^{1 \times M}$ denotes the kernel activation of p-th base GP, μ is the mean of additive GP, and $\mathbf{z}_p$'s are random weights with i.i.d. normal prior distribution $\mathbf{z}_p \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_M)$, for all $p = 1, \ldots, P$.

The proposed DAK in eq. (9) results in deep additive kernel learning which mathematically possesses the form of a last-layer BNN (MacKay, 1992) but with a kernel activation $\phi(\cdot) := \mathbf{K}_{(\cdot),\mathbf{U}} \left[\mathbf{L}_{\mathbf{U}}^\top\right]^{-1}$ followed by a Gaussian forward layer with i.i.d. Normal prior weights. Figure 1 illustrates the module architecture of DAK. Thanks to the mathematical equivalence to BNN and reparameterization tricks on learnable parameters, all modules of our model can be implemented either as a deterministic or a Bayesian NN

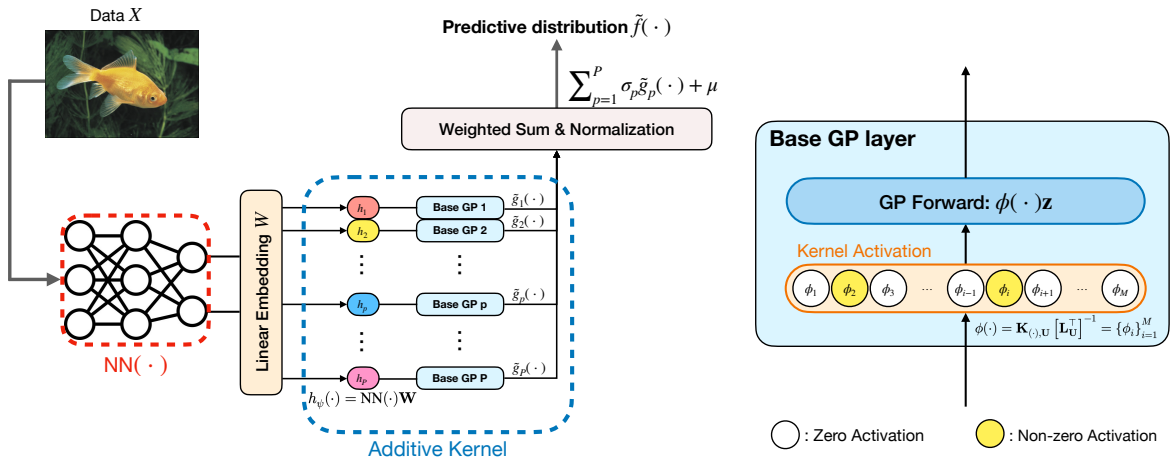


Figure 1: Model architecture of Deep Additive Kernel (DAK). DAK consists of a feature extractor $\mathrm{NN}(\cdot)$ with a linear embedding layer $\mathbf{W}$, an additive kernel with GP layers $\tilde{g}_p(\cdot)$ for $p = 1, \ldots, P$, and a weighted sum layer. The embedded features learned by DNN are decomposed as first-order components and fed to base GP layers, each consisting of a kernel activation and a GP forward layer. Each kernel activation is designed by a one-dimensional dyadic point set on an induced grid with sparse non-zero activated neurons.

layer. This results in a single NN with hybrid layers, which allows us to straightforwardly take advantage of parallelized GPU computing using the readily available PyTorch package (Paszke et al., 2019).

Inference. To obtain the predictive distribution, we perform Variational Inference (VI) to estimate the posterior using DAK in eq. (9). With a motivation to naturally model the deep additive kernel learning as a last-layer BNN, and taking the mean field assumptions, we select a variational family of independent but not identical Gaussian weights $\mathbf{Z} := [\mathbf{z}_1, \ldots, \mathbf{z}_P]$ and additive Gaussian bias μ, denoted by $\Theta_{\text{var}} := \left\{\{\mathbf{z}_p\}_{p=1}^{P}, \mu\right\}$. The variational Gaussian weights are parameterized as $\mathbf{z}_p \sim \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$ for $p = 1, \ldots, P$, and the variational bias is parameterized as $\mu \sim \mathcal{N}(m_\mu, \sigma_\mu^2)$. We denote the reparameterization of Θ_{var} as $\boldsymbol{\eta} := \left\{\{\mathbf{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}\}_{p=1}^{P}, \{m_\mu, \sigma_\mu\}\right\}$.

Note that $\mathbf{S}_{\mathbf{z}_p} \in \mathbb{R}^{M \times M}$ is a diagonal covariance matrix due to the mean field assumption of $\mathbf{z}_p$, where M is decided by the size of induced interpolation grids defined in eq. (8). The variational distribution is given by $q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) = q(\mu) \prod_{p=1}^{P} q(\mathbf{z}_p) = \mathcal{N}(m_\mu, \sigma_\mu^2) \prod_{p=1}^{P} \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$, and the prior of Θ_{var} is denoted by $p(\Theta_{\text{var}})$.

The other deterministic parameters consist of DNN parameters ψ and additive GP hyperparameters $\{\boldsymbol{\sigma}\} := [\sigma_1, \ldots, \sigma_P]^\top$, denoted as $\boldsymbol{\theta} := \{\psi, \boldsymbol{\sigma}\}$. We treat GP lengthscales as fixed parameters specified during initialization and apply an additional linearly embedding layer to DNN: $h_\psi(\cdot) := \mathrm{NN}(\cdot)\mathbf{W} : \mathbb{R}^D \to \mathbb{R}^P$, where $\mathrm{NN}(\cdot) : \mathbb{R}^D \to \mathbb{R}^{D_w}$ represents any DNN, and $\mathbf{W} \in \mathbb{R}^{D_w \times P}$ is the linear embedding. The lengthscales can be inherently optimized by learning the NN weights $\mathbf{W}$ without loss in performance, which is encoded in the DNN parameters ψ. Further details are provided in Appendix B.

Given a data point $\mathbf{x} \in \mathbb{R}^D$, the forward pass of predictive distribution is given by

$$\tilde{f}_{\mathbf{x}} := \tilde{f}(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\eta}) = \sum_{p=1}^{P} \sigma_p \Big(\phi\big(h_\psi^{[p]}(\mathbf{x})\big) \mathbf{z}_p \Big) + \mu, \quad (10)$$

where the complete set of parameters is $\Theta := \{\boldsymbol{\theta}, \boldsymbol{\eta}\} = \left\{\psi, \boldsymbol{\sigma}, \{\mathbf{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}\}_{p=1}^{P}, \{m_\mu, \sigma_\mu\}\right\}$, consisting of the deterministic parameters $\boldsymbol{\theta} := \{\psi, \boldsymbol{\sigma}\}$ and the variational parameters $\boldsymbol{\eta} := \left\{\{\mathbf{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}\}_{p=1}^{P}, \{m_\mu, \sigma_\mu\}\right\}$. In Appendix C, we derive an analytical form for the predictive distribution $\tilde{f}$.

Training. Vanilla DKL optimizes the marginal log-likelihood $\log \Pr(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$ which involves intractable integral of non-conjugate likelihoods in some tasks such as classification. We apply the framework of stochastic variational inference to fit GPs via the variational distribution $q_{\boldsymbol{\eta}}(\Theta_{\text{var}})$. Consequently, during training, we optimize the variational lower bound, as formulated in Hensman et al. (2015), to achieve efficient and scalable inference:

$$\log \Pr(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \geq \sum_{\mathbf{x}, y \in \mathbf{X}, \mathbf{y}} \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})} \left[\log \Pr\big(y|\tilde{f}_{\mathbf{x}}\big) \right] - \mathrm{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}}) \right]. \quad (11)$$

The details of the training are derived in Appendix D. The resulting objective is known as Evidence Lower

Table 1: Computational complexity of DKL models for N training points in one iteration. $\hat{M}$ is the number of inducing points in SVGP and KISS-GP, while M is the size of induced grids in DAK, $M < \hat{M}$. S is the number of Monte Carlo samples, B is the size of mini-batch, D_w is the dimension of the NN outputs in DKL, P is the dimension after the linear embedding of NN features. DAK-MC refers to DAK using Monte Carlo approximation, while DAK-CF refers to DAK using closed-form inference and ELBO.

	Inference	**Training** (per iteration)
NN + SVGP	$\mathcal{O}(\hat{M}^2 N)$	$\mathcal{O}(SD_w\hat{M}B + \hat{M}^3)$
NN + KISS-GP	$\mathcal{O}(D_w\hat{M}^{1+\frac{1}{D_w}})$	$\mathcal{O}(SD_w\hat{M}B + D_w\hat{M}^{\frac{3}{D_w}})$
DAK-MC (ours)	$\mathcal{O}(SM)$	$\mathcal{O}(SPMB + PM)$
DAK-CF (ours)	$\mathcal{O}(M)$	$\mathcal{O}(PMB + PM)$

Bound (ELBO):

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\eta}) := \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \Pr(\mathbf{y}|\tilde{f}_{\mathbf{X}})\right] - \text{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}})\right]. \quad (12)$$

The first term of ELBO, the expected log-likelihood, can be estimated by Monte Carlo methods. To avoid the potential computing cost of sampling, we also derive a sampling-free analytical ELBO with a closed form for regression tasks. The details of derivation are deferred to Appendix E.

Computational Complexity. We summarize the computational complexity of our proposed DAK model compared to other state-of-the-art GP and DKL methods in Table 1. On the one hand, the number of inducing points $\hat{M}$ in SVGP and KISS-GP may need to be chosen quite large in more complex or multi-task GPs since it is associated with the GP input dimensionality, while at competitive performances the size of induced grids M in DAK can be small due to the first-order additive structure. On the other hand, the dimension of the embedding layer P is usually smaller than the dimension of NN outputs D_w. Further discussion is deferred to Appendix F.

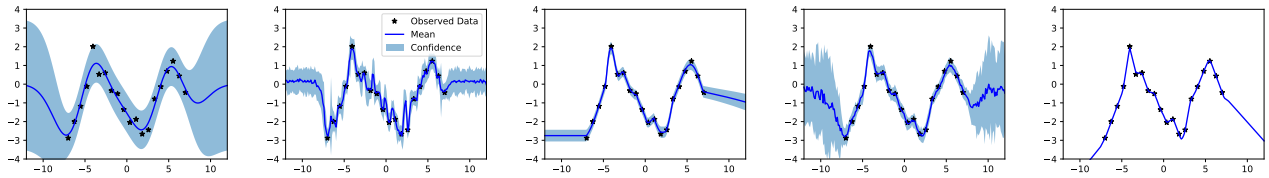
Remarks. We highlight several key aspects of the proposed model: **1)** The proposed model adds base GP components directly, rather than adding kernels as described in (Duvenaud et al., 2011). While they are mathematically equivalent, using an additive kernel alone does not lead to any computational advantage over any standard kernel, where the cubic computational complexity persists. In contrast, the additive GP led us to apply the induced approximation technique, which naturally leads to reduced computation and the last-layer Bayesian interpretation. **2)** The induced prior approximation in our model differs from the standard inducing points approximation (Titsias, 2009) often used for GPs. In our approach, we approximate the prior using a fixed set of induced grids, resulting in a BNN representation. In contrast, the standard inducing point methods treat the inducing points as variational parameters for optimization, which does not lead to a BNN representation. **3)** In the forward pass $f(\cdot)$ as defined in eq. (10), we chose to place a constant prior on the mean μ on $f(\cdot)$, which also naturally facilitates the construction of the last-layer BNN.

4 RELATED WORK

Given the motivation of integrating the power of deep networks with the interpretability and theoretical foundations of kernel methods, it is encouraging to see some contributions on such combinations across a range of contexts.

Variations of DKL. Several recent studies have explored variations of DKL models. To handle large datasets and diverse tasks, Wilson et al. (2016b) extended the vanilla DKL (Wilson et al., 2016a) to Stochastic Variational DKL (SV-DKL) by leveraging Stochastic Variational GP (SVGP) (Hensman et al., 2015) and KISS-GP (Wilson and Nickisch, 2015). However, the kernel interpolation on a Cartesian grid does not scale in the high-dimensional space. Xue et al. (2019) and Xie et al. (2019) integrated deep kernel models with Random Fourier Features (RFF) (Rahimi and Recht, 2007), an efficient method for approximating GP kernel feature mappings. Some other work has focused on developing models with specialized kernel structures, such as recurrent kernels (Al-Shedivat et al., 2017) and compositional kernels (Sun et al., 2018). More recently, Ober et al. (2021) investigated overfitting in DKL models based on marginal likelihood maximization and proposed a fully Bayesian approach to mitigate this issue. Matias et al. (2024) introduced amortized variational DKL, which uses DNNs to learn variational distributions over inducing points in an amortized manner, thereby reducing overfitting by locally smoothing predictions.

GPs and NNs. The connection between GPs and NNs was first established by Neal (1994), who showed that the function defined by a single-layer NN with infinite width, random independent zero-mean weights, and biases is equivalent to a GP. This equivalence was later generalized to arbitrary NNs with infinite-width or infinite-depth layers in (Lee et al., 2017; Cutajar et al., 2017; Matthews et al., 2018; Yang, 2019; Dutordoir et al., 2021; Gao et al., 2023). However, these studies focus primarily on fully connected NNs, which are not suitable for all practical applications. Garriga-Alonso et al. (2018) and Novak et al. (2018) extended the equivalence to Convolutional Neural Networks (CNNs) (LeCun et al., 1989), which are widely



(a) Exact GP. (b) DGP. (c) Exact DKL. (d) DAK. (e) DNN.

Figure 2: Results on toy dataset. (a)–(d) show the predictive posterior of the exact GP, DGP, exact DKL and proposed DAK model, respectively, on the noisy data generated by 1D GP with zero-mean and covariance function $k(x, x') = \exp(-(x - x')^2)$. We plot the predictive mean and ± 2 standard deviations together with the observed data. (e) shows the NN fit with the same training data.

used in image recognition. Hybrid models combining GPs and NNs have also been investigated. Bradshaw et al. (2017) proposed a hybrid GPDNN that feeds CNN features into a GP, while Zhang et al. (2024) introduced GP Neural Additive Models (GP-NAM), a class of GAMs that use GPs approximated by RFF and a single-layer NN. However, GPDNN uses the standard GP, while we approximate GP via induced prior approximation. GP-NAM applies the additive model with RFF approximation but lacks Bayesian inference. Harrison et al. (2024) introduced a sampling-free Bayesian last-layer architecture but did not establish a connection between this architecture and DKL.

5 EXPERIMENTS

In this section, we evaluate the proposed DAK model on multiple real datasets for both regression tasks in Section 5.2 and classification tasks in Section 5.3. We compare its performance to several baselines, including a neural network without GP integration (NN), DKL with SVGP (Titsias, 2009; Hensman et al., 2015) as the GP approximation (NN+SVGP), and SV-DKL (Wilson et al., 2016b). Additionally, in Section 5.1, we present a toy example demonstrating how the proposed model mitigates the out-of-sample overfitting issue highlighted by Ober et al. (2021), in contrast to the NN model. Source code is available at the following anonymized link [1].

5.1 Toy Example: GP

We first evaluate a toy example of 1-Dimensional (1D) GP regression on synthetic data with zero-mean and SE kernel $k(x, x') = \exp\left(-(x - x')^2\right)$. The training set consists of 20 noisy data points in the range of $[-7, 7]$, while the test set consist of 100 data points in $[-12, 12]$. We consider a two-layer MLP with layer-width $[64, 32]$ as the feature extractor, letting $P = 2$ as the number of *base* GPs. To maintain a fair comparison, we use the same training recipe: full-batch training, a learning rate 0.01, and a number of optimization steps 1000. The only differences are the choice of last layer and the loss function.

Figure 2a – 2d shows the predictive posterior of the exact GP, two-layer DGP, exact DKL, and proposed DAK. We observe that DAK has good both in-sample and out-of-sample predictions which is close to exact GP posterior, while DGP is unbiased but over-confident outside the training data. Exact DKL is biased and over-confident on out-of-sample predictions, which were also observed and investigated by Ober et al. (2021). We also compare DAK with the DNN that shares the same feature extractor and model depth. It is evident that the DNN fit in Figure 2e suffers from "overfitting": the fit shows significant extrapolation beyond the training data.

5.2 UCI Regression

We benchmark the regression task on six datasets from the UCI repository (Dua and Graff, 2017): three smaller datasets with fewer than 10K samples (Wine, Gas, Parkinsons) and three larger datasets with 40K to 50K samples (Kin40K, Protein, KEGG). All models use the same neural network architecture: a fully connected network with two hidden layers of 64 and 32 neurons, respectively. Training is performed using the Adam optimizer over 100 epochs with a learning rate of 0.001, weight decay of 0.0005, and batch size of 512. The noise variance is set to $\sigma_f^2 = 0.01$.

For NN+SVGP and SV-DKL, we use SE kernels and 64 inducing points initialized with uniformly distributed random variables of a fixed seed. In the proposed DAK models, the induced grid $\mathbf{U}$ from eq. (8) consists of $M = 7$ equally spaced points over the interval $(0, 1)$ for each base GP, i.e., $\mathbf{U} = \{1/8, 2/8, \ldots, 7/8\}$. The NN outputs of both SV-DKL and our DAK model, which utilize an additive GP structure, are embedded and normalized into 16 base GPs over the interval $[0, 1]$. That is, the number of projections P in eq. (5) is set to 16. For the DAK model, we evaluate two methods: DAK-CF, using the closed-form inference from Appendix C and the closed-form

[1] `https://anonymous.4open.science/r/dak2bnn-v1/`

Table 2: Comparison of regression performance on UCI datasets using 5-fold cross-validation with a batch size of 512 and a fully connected neural network architecture of $D \to 64 \to 32 \to D_w$, where the output features D_w are 16, 64, and 256 respectively. The best results are highlighted in **bold**. Our models, DAK-MC (using Monte Carlo approximation) and DAK-CF (using closed-form inference and ELBO), are highlighted with gray background.

Dataset (N, D)	Model	NN out features = 16			NN out features = 64			NN out features = 256		
		RMSE ↓	NLPD ↓	Time (s) ↓	RMSE ↓	NLPD ↓	Time (s) ↓	RMSE ↓	NLPD ↓	Time (s) ↓
Gas (2565, 128)	NN	2.377 ± 2.399	4.749 ± 6.594	**2.345 ± 0.003**	2.107 ± 1.212	3.092 ± 2.229	**2.362 ± 0.029**	1.196 ± 0.712	1.730 ± 0.711	**2.346 ± 0.004**
	NN+SVGP	0.502 ± 0.171	1.121 ± 0.106	4.727 ± 0.009	0.625 ± 0.148	1.206 ± 0.119	4.724 ± 0.004	0.743 ± 0.183	1.322 ± 0.143	4.718 ± 0.009
	SV-DKL	0.589 ± 0.161	1.303 ± 0.227	28.189 ± 0.490	0.499 ± 0.183	1.235 ± 0.256	27.956 ± 0.078	0.534 ± 0.189	1.207 ± 0.209	28.400 ± 0.185
	DAK-MC	**0.405 ± 0.061**	**0.886 ± 0.048**	8.887 ± 0.007	0.353 ± 0.046	**0.881 ± 0.053**	8.844 ± 0.005	0.351 ± 0.019	**0.871 ± 0.027**	8.831 ± 0.017
	DAK-CF	0.412 ± 0.134	0.928 ± 0.100	7.400 ± 0.004	**0.350 ± 0.020**	0.898 ± 0.040	7.398 ± 0.009	**0.342 ± 0.033**	0.895 ± 0.046	7.410 ± 0.009
Parkinsons (5875, 20)	NN	2.692 ± 1.302	3.503 ± 2.630	**2.693 ± 0.027**	2.288 ± 0.712	2.724 ± 1.158	**2.589 ± 0.050**	2.252 ± 0.757	2.720 ± 1.188	**2.885 ± 0.026**
	NN+SVGP	3.481 ± 1.906	4.606 ± 4.606	5.516 ± 0.117	3.238 ± 2.419	4.940 ± 5.494	5.467 ± 0.097	3.676 ± 3.287	6.791 ± 8.612	5.945 ± 0.090
	SV-DKL	2.608 ± 1.023	2.745 ± 1.097	35.804 ± 1.590	2.817 ± 1.670	3.193 ± 2.109	33.042 ± 0.306	2.896 ± 2.055	3.206 ± 1.906	32.882 ± 0.060
	DAK-MC	1.983 ± 1.154	**2.699 ± 1.819**	11.596 ± 0.260	1.949 ± 0.912	**2.575 ± 1.121**	13.085 ± 0.055	1.846 ± 0.974	2.420 ± 1.212	11.820 ± 0.296
	DAK-CF	**1.801 ± 1.013**	2.848 ± 1.810	9.071 ± 0.083	**1.788 ± 0.997**	2.801 ± 1.853	9.073 ± 0.048	**1.466 ± 1.093**	**2.308 ± 1.892**	8.166 ± 0.071
Wine (1599, 11)	NN	**0.728 ± 0.055**	1.233 ± 0.057	**2.350 ± 0.011**	**0.725 ± 0.057**	1.231 ± 0.054	**2.343 ± 0.007**	**0.712 ± 0.062**	1.214 ± 0.062	**2.367 ± 0.021**
	NN+SVGP	0.739 ± 0.069	1.243 ± 0.067	4.713 ± 0.010	0.801 ± 0.130	1.325 ± 0.130	4.725 ± 0.008	0.893 ± 0.180	1.418 ± 0.147	4.718 ± 0.016
	SV-DKL	0.899 ± 0.110	1.443 ± 0.135	27.224 ± 0.170	0.947 ± 0.109	1.460 ± 0.112	27.136 ± 0.023	0.879 ± 0.130	1.434 ± 0.158	27.332 ± 0.101
	DAK-MC	0.756 ± 0.068	1.164 ± 0.075	8.820 ± 0.038	0.751 ± 0.055	1.163 ± 0.060	8.821 ± 0.045	0.727 ± 0.065	**1.140 ± 0.070**	8.765 ± 0.016
	DAK-CF	0.736 ± 0.042	**1.162 ± 0.049**	7.376 ± 0.018	0.728 ± 0.064	**1.153 ± 0.067**	7.352 ± 0.010	0.720 ± 0.057	1.147 ± 0.061	7.415 ± 0.029
Kin40K (40000, 8)	NN	0.109 ± 0.016	0.752 ± 0.004	**16.017 ± 0.114**	0.100 ± 0.012	0.749 ± 0.003	**16.912 ± 0.112**	0.087 ± 0.016	0.746 ± 0.004	**18.048 ± 0.025**
	NN+SVGP	0.104 ± 0.013	0.751 ± 0.004	40.749 ± 0.508	0.142 ± 0.017	0.763 ± 0.006	41.239 ± 0.304	0.123 ± 0.016	0.757 ± 0.006	43.111 ± 1.482
	SV-DKL	0.096 ± 0.024	0.750 ± 0.010	230.284 ± 7.049	0.092 ± 0.018	0.748 ± 0.006	228.137 ± 7.406	0.097 ± 0.025	0.750 ± 0.009	226.606 ± 4.231
	DAK-MC	0.096 ± 0.042	0.746 ± 0.010	74.787 ± 0.383	0.090 ± 0.028	0.744 ± 0.006	77.349 ± 5.409	0.090 ± 0.031	0.744 ± 0.007	75.586 ± 0.313
	DAK-CF	**0.073 ± 0.015**	**0.742 ± 0.005**	54.174 ± 0.322	**0.069 ± 0.018**	**0.741 ± 0.005**	60.339 ± 0.161	**0.068 ± 0.015**	**0.741 ± 0.004**	55.148 ± 0.165
Protein (45730, 9)	NN	4.678 ± 7.804	25.091 ± 52.135	**24.910 ± 0.221**	0.906 ± 0.304	1.421 ± 0.232	**19.433 ± 0.093**	1.403 ± 1.408	2.358 ± 2.349	**28.447 ± 0.133**
	NN+SVGP	0.773 ± 0.003	1.278 ± 0.002	47.081 ± 0.643	0.773 ± 0.003	1.278 ± 0.002	38.986 ± 0.485	0.773 ± 0.003	1.278 ± 0.002	52.924 ± 1.144
	SV-DKL	0.769 ± 0.003	1.275 ± 0.001	262.274 ± 8.216	0.770 ± 0.003	1.276 ± 0.002	265.182 ± 3.680	0.769 ± 0.003	1.275 ± 0.001	264.678 ± 4.450
	DAK-MC	**0.622 ± 0.047**	**1.017 ± 0.043**	85.729 ± 1.035	**0.612 ± 0.038**	**1.008 ± 0.035**	86.594 ± 0.827	**0.608 ± 0.042**	**1.004 ± 0.039**	86.245 ± 0.088
	DAK-CF	0.631 ± 0.021	1.024 ± 0.019	68.762 ± 0.379	0.625 ± 0.027	1.018 ± 0.026	69.766 ± 0.190	0.618 ± 0.029	1.012 ± 0.026	68.783 ± 0.222
KEGG (48827, 20)	NN	0.132 ± 0.021	0.759 ± 0.006	**21.856 ± 0.168**	0.124 ± 0.009	0.758 ± 0.004	**20.989 ± 0.132**	0.127 ± 0.008	0.766 ± 0.019	**23.143 ± 0.710**
	NN+SVGP	0.128 ± 0.005	0.758 ± 0.001	43.788 ± 1.377	0.129 ± 0.013	0.759 ± 0.002	40.996 ± 0.280	0.127 ± 0.009	0.758 ± 0.002	46.882 ± 1.290
	SV-DKL	0.134 ± 0.011	0.766 ± 0.013	269.598 ± 5.733	0.139 ± 0.031	0.769 ± 0.025	271.083 ± 7.502	0.152 ± 0.024	0.780 ± 0.025	275.013 ± 2.443
	DAK-MC	0.126 ± 0.010	**0.748 ± 0.003**	90.473 ± 0.144	0.124 ± 0.010	**0.748 ± 0.003**	92.507 ± 0.964	0.123 ± 0.009	**0.748 ± 0.003**	113.738 ± 1.251
	DAK-CF	**0.124 ± 0.007**	**0.748 ± 0.003**	65.070 ± 0.140	**0.122 ± 0.007**	**0.748 ± 0.003**	79.732 ± 2.189	**0.121 ± 0.005**	**0.748 ± 0.003**	78.702 ± 0.631

ELBO from Appendix E, and DAK-MC, using Monte Carlo sampling to approximate the mean and variance during inference and approximate the expected log likelihood term in ELBO during training. The number of Monte Carlo samples is set to 20 for inference and 8 for training, the same for SV-DKL and NN+SVGP. Further details can be found in Appendix G.1.

We assess the performance of each model using training time (in seconds), Root Mean Squared Error (RMSE), and Negative Log Predictive Density (NLPD) with varying neural network output feature sizes of 16, 64, and 256. The results are averaged over 5-fold cross-validation for each dataset. As shown in Table 2, except for the smallest dataset, Wine, our models, DAK-MC and DAK-CF (highlighted in gray), consistently achieve the best RMSE and NLPD performance compared to other models.

Although NN+SVGP takes less time than the DAK models, its performance often degrades, with higher RMSE and NLPD as the size of the NN output features increases. In contrast, the DAK models show improved performance with larger NN output features. This is because SVGP has a fixed number of 64 inducing points, which limits its ability to approximate GPs effectively in high-dimensional spaces, making it less suitable for complex tasks requiring high-dimensional NNs, such as multi-task regression or meta-learning.

Additionally, SV-DKL, which also uses an additive GP layer and KISS-GP to accelerate GP computations, takes significantly more time than our DAK models. This is because SV-DKL treats dependent inducing variables as parameters in VI, which requires more training time, while our DAK models treat the GP layers as sparse BNNs with independent Gaussian weights and biases under the mean-field assumption, leading to more efficient training.

5.3 Image Classification

We next benchmark the classification task on high-dimensional and highly-structured image data, including MNIST (LeCun et al., 1998), CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009). All models share the same neural network head as feature extractors, to which we add either a linear output layer in NN model or corresponding GP output layers in NN+SVGP, SV-DKL, and DAK. In classification, last layer outputs are followed by a Softmax layer to normalize the output to a probability distribution, and we perform Monte Carlo sampling to approximate the Softmax likelihood term in ELBO.

We use the same setting of training for all models (refer to Table 7 in Appendix G.2 for details). For MNIST, we consider a simple CNN as the feature extractor, 20 epochs of training using Adadelta optimizer with initial learning rate 1.0, weight dacay 0.0001, and annealing learning rate scheduler at each step with a factor of 0.7. For CIFAR-10, we perform full-training on a ResNet-18 (He et al., 2016) with GP layers for 200 epochs, while for CIFAR-100, we use a pretrained ResNet-34 as the feature extractor and fine-tune GP layers for 200 epochs since NN+SVGP and SV-DKL struggled to fit. In both CIFAR10/100, we use SGD

Table 3: ELBO, Accuracy, NLL, ECE for DAK, SV-DKL, NN+SVGP, NN on image classification tasks averaged over 3 runs. MNIST uses a simple CNN with 16 output features; CIFAR-10 uses ResNet-18 with 64 output features; CIFAR-100 uses ResNet-34 with 512 output features. The best results are highlighted in **bold**.

	Batch size: 128				Batch size: 1024		
	NN	NN+SVGP	SV-DKL	DAK-MC	NN+SVGP	SV-DKL	DAK-MC
MNIST - ELBO ↑	—	-0.121 ± 0.000	-0.054 ± 0.000	**-0.030 ± 0.001**	-1.997 ± 0.001	-0.305 ± 0.001	**-0.048 ± 0.001**
Top-1 Acc. (%) ↑	99.14 ± 0.00	98.56 ± 0.02	99.16 ± 0.00	**99.26 ± 0.01**	14.79 ± 0.08	96.23 ± 0.02	**99.1 ± 0.00**
NLL ↓	0.026 ± 0.000	0.064 ± 0.001	0.030 ± 0.000	**0.024 ± 0.000**	1.985 ± 0.003	0.288 ± 0.001	**0.028 ± 0.004**
ECE ↓	0.005 ± 0.000	0.012 ± 0.000	0.005 ± 0.000	**0.004 ± 0.000**	0.062 ± 0.002	0.020 ± 0.000	**0.006 ± 0.000**
CIFAR-10 - ELBO ↑	—	-1.038 ± 0.004	-0.017 ± 0.001	**-0.002 ± 0.000**	-1.039 ± 0.004	-0.034 ± 0.021	**-0.002 ± 0.000**
Top-1 Acc. (%) ↑	94.72 ± 0.13	77.35 ± 0.13	93.44 ± 0.28	**94.81 ± 0.13**	16.90 ± 0.29	90.22 ± 1.42	**93.02 ± 0.18**
NLL ↓	**0.252 ± 0.025**	1.790 ± 0.001	0.312 ± 0.033	0.256 ± 0.014	2.270 ± 0.000	0.485 ± 0.061	**0.345 ± 0.001**
ECE ↓	0.040 ± 0.002	0.061 ± 0.003	0.046 ± 0.003	**0.039 ± 0.002**	**0.027 ± 0.002**	0.060 ± 0.004	0.052 ± 0.001
CIFAR-100 - ELBO ↑	—	-4.605 ± 0.000	-0.059 ± 0.000	**-0.003 ± 0.000**	-4.605 ± 0.000	-0.103 ± 0.009	**-0.005 ± 0.001**
Top-1 Acc. (%) ↑	75.88 ± 0.54	1.04 ± 0.01	74.52 ± 0.13	**76.75 ± 0.18**	1.10 ± 0.09	66.54 ± 0.74	**70.38 ± 1.25**
NLL ↓	1.018 ± 0.021	4.605 ± 0.000	1.041 ± 0.007	**1.001 ± 0.027**	4.605 ± 0.000	1.738 ± 0.058	**1.203 ± 0.040**
ECE ↓	0.086 ± 0.002	**0.003 ± 0.001**	0.049 ± 0.002	0.041 ± 0.004	**0.003 ± 0.001**	0.148 ± 0.007	0.056 ± 0.006

Table 4: The number of total trainable parameters and average training time per epoch across different tasks for each model. NN has smaller set of parameters and less training time as is expected. DAK is more scalable than SV-DKL in terms of total trainable parameters and training time per epoch.

Datasets	N	D	C	D_w	# parameters				Epoch time (sec.)			
					NN	NN+SVGP	SV-DKL	DAK-MC	NN	NN+SVGP	SV-DKL	DAK-MC
MNIST	60K	28×28	10	16	1.19M	+2.72M	+0.08M	**+0.01M**	6.18	+16.57	**+4.35**	+4.57
CIFAR-10	50K	3×32×32	10	64	11.17M	+29.58M	+0.30M	**+0.04M**	34.41	+22.92	+7.88	**+5.02**
CIFAR-100	50K	3×32×32	100	512	21.32M	+52.53M	+2.19M	**+0.10M**	42.82	+101.77	+16.61	**+6.24**

optimizer with an initial learning rate of 0.1, weight decay of 0.0001, momentum 0.9, and cosine annealing learning rate scheduler.

For NN+SVGP, we use SE kernel and 512 inducing points, while for SV-DKL, we use 64 inducing points initialized with uniformly distributed random variable within the interval $[-1, 1]^{D_w}$. For the proposed DAK-MC, we use $M = 63$ equally spaced points over the interval $[-1, 1]$ for the induced grid $\mathbf{U}$ in each base GP. Further details can be found in Appendix G.2.

We evaluate all models in terms of Top-1 accuracy, NLLs, ELBOs, and expected calibration error (ECE) over three runs. As shown in Table 3, our DAK-MC (highlighted in gray) achieves the best accuracy, ELBO, and NLL performance compared to other DKL models. Although some ECEs of NN+SVGP and SV-DKL is lower than DAK-MC, their accuracy degrades more with the increment of dimensionality. The failure of SVGP in CIFAR-10/100 demonstrated the necessity of additive structure in high-dimensional multitask DKL. Additionally, we observe that SVDKL struggles more to fit in CIFAR-100, indicating the importance of pre-training in SVDKL, while our proposed DAK does not hurt by the curse of dimensionality because of the computational advantages of the last-layer BNN feature. We also repeated the experiments with a larger batch size of 1024. Our proposed DAK model is more robust than other DKL methods when the batch size and the number of features increases. Further experimental results are provided in Appendix G.3. In Table 4, we measure the total number of trainable parameters and average training time. NN has the smallest set of parameters and the least training time as expected. Among DKL methods, DAK is more efficient than SV-DKL when large-scale neural networks and high-dimensional tasks are applied.

6 CONCLUSION

In this work, we introduced the DAK model, which reinterprets DKL as a hybrid NN, representing the last-layer GP as a sparse BNN layer to address the scalability and training inefficiencies of DKL. The DAK model overcomes limitations of GPs by embedding high-dimenional features from NNs into additive GP layers and leveraging the sparse Cholesky factor of the Laplace kernel on the induced grids, significantly enhancing training and inference efficiency. The proposed model also provides closed-form solutions for both the predictive distribution and ELBO in regression tasks, eliminating the need for costly Monte Carlo sampling. Empirical results show that DAK outperforms state-of-the-art DKL models in both regression and classification tasks. This work also opens new possibilities for improving DKL by establishing the connection between BNNs and GPs.

Possible directions for future work include considering more general GP layers other than the Laplace kernels and the additive structure, and exploring other variational families for training the BNNs.

References

Al-Shedivat, M., Wilson, A. G., Saatchi, Y., Hu, Z., and Xing, E. P. (2017). Learning scalable deep kernels with recurrent structure. *Journal of Machine Learning Research*, 18(82):1–37.

Bradshaw, J., Matthews, A. G. d. G., and Ghahramani, Z. (2017). Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*.

Cutajar, K., Bonilla, E. V., Michiardi, P., and Filippone, M. (2017). Random feature expansions for deep Gaussian processes. In *International Conference on Machine Learning*, pages 884–893. PMLR.

Damianou, A. and Lawrence, N. D. (2013). Deep Gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR.

Delbridge, I., Bindel, D., and Wilson, A. G. (2020). Randomly projected additive Gaussian processes for regression. In *International Conference on Machine Learning*, pages 2453–2463. PMLR.

Ding, L., Tuo, R., and Shahrampour, S. (2024). A sparse expansion for deep Gaussian processes. *IISE Transactions*, 56(5):559–572.

Dua, D. and Graff, C. (2017). UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`.

Dutordoir, V., Hensman, J., van der Wilk, M., Ek, C. H., Ghahramani, Z., and Durrande, N. (2021). Deep neural networks as point estimates for deep Gaussian processes. *Advances in Neural Information Processing Systems*, 34:9443–9455.

Duvenaud, D. K., Nickisch, H., and Rasmussen, C. (2011). Additive Gaussian processes. *Advances in neural information processing systems*, 24.

Gao, T., Huo, X., Liu, H., and Gao, H. (2023). Wide neural networks as Gaussian processes: Lessons from deep equilibrium models. *Advances in Neural Information Processing Systems*, 36:54918–54951.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. *Advances in neural information processing systems*, 31.

Garriga-Alonso, A., Rasmussen, C. E., and Aitchison, L. (2018). Deep convolutional networks as shallow Gaussian processes. *arXiv preprint arXiv:1808.05587*.

Harrison, J., Willes, J., and Snoek, J. (2024). Variational Bayesian last layers. *arXiv preprint arXiv:2404.11599*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hensman, J., Matthews, A., and Ghahramani, Z. (2015). Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR.

Jafrasteh, B., Villacampa-Calvo, C., and Hernandez-Lobato, D. (2022). Input dependent sparse Gaussian processes. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9739–9759. PMLR.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *Technical report, University of Toronto*. `https://www.cs.toronto.edu/~kriz/cifar.html`.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a backpropagation network. *Advances in neural information processing systems*, 2.

LeCun, Y., Cortes, C., and Burges, C. J. (1998). The MNIST database of handwritten digits. `https://yann.lecun.com/exdb/mnist/`.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). Deep neural networks as Gaussian processes. *arXiv preprint arXiv:1711.00165*.

MacKay, D. J. (1992). A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

Matias, A. L. S., Mattos, C. L., Gomes, J. P. P., and Mesquita, D. (2024). Amortized variational deep kernel learning. In *Forty-first International Conference on Machine Learning*.

Matthews, A. G. d. G., Rowland, M., Hron, J., Turner, R. E., and Ghahramani, Z. (2018). Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning*, pages 807–814.

Neal, R. M. (1994). *Bayesian learning for neural networks*. PhD thesis, University of Toronto, Dept. of Computer Science.

Novak, R., Xiao, L., Lee, J., Bahri, Y., Yang, G., Hron, J., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Bayesian deep convolutional networks with many channels are Gaussian processes. *arXiv preprint arXiv:1810.05148*.

Ober, S. W., Rasmussen, C. E., and van der Wilk, M. (2021). The promises and pitfalls of deep kernel learning. In *Uncertainty in Artificial Intelligence*, pages 1206–1216. PMLR.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20.

Sun, S., Zhang, G., Wang, C., Zeng, W., Li, J., and Grosse, R. (2018). Differentiable compositional kernel learning for Gaussian processes. In *International Conference on Machine Learning*, pages 4828–4837. PMLR.

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

Wang, W., Tuo, R., and Jeff Wu, C. (2020). On prediction properties of kriging: Uniform error bounds and robustness. *Journal of the American Statistical Association*, 115(530):920–930.

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

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International conference on machine learning*, pages 1775–1784. PMLR.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016a). Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. (2016b). Stochastic variational deep kernel learning. *Advances in neural information processing systems*, 29.

Xie, J., Liu, F., Wang, K., and Huang, X. (2019). Deep kernel learning via random Fourier features. *arXiv preprint arXiv:1910.02660*.

Xue, H., Wu, Z.-F., and Sun, W.-X. (2019). Deep Spectral Kernel Learning. In *IJCAI*, pages 4019–4025.

Yakowitz, S. and Szidarovszky, F. (1985). A comparison of kriging with nonparametric regression methods. *Journal of Multivariate Analysis*, 16(1):21–53.

Yang, G. (2019). Wide feedforward or recurrent neural networks of any architecture are Gaussian processes. *Advances in Neural Information Processing Systems*, 32.

Zhang, W., Barr, B., and Paisley, J. (2024). Gaussian Process Neural Additive Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16865–16872.

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]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [**Yes**/No/Not Applicable]
   (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]
   (b) Complete proofs of all theoretical results. [**Yes**/No/Not Applicable]
   (c) Clear explanations of any assumptions. [**Yes**/No/Not Applicable]

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]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [**Yes**/No/Not Applicable]
   (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]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [**Yes**/No/Not Applicable]

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]
   (b) The license information of the assets, if applicable. [Yes/No/**Not Applicable**]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes/No/**Not Applicable**]
   (d) Information about consent from data providers/curators. [Yes/No/**Not Applicable**]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes/No/**Not Applicable**]

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**]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Yes/No/**Not Applicable**]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Yes/No/**Not Applicable**]

From Deep Additive Kernel Learning to Last-Layer Bayesian Neural Networks via Induced Prior Approximation: Supplementary Materials

A SPARSE CHOLESKY DECOMPOSITION

In this section, we present the algorithm for constructing the induced grids $\mathbf{U}$ as defined in eq. (8) by using sorted dyadic points, and obtaining the sparse Choleksy decomposition of the Laplace kernel in one dimension, as proposed in (Ding et al., 2024).

A set of one-dimensional level-L dyadic points $\mathbf{X}_L$ in increasing order over the interval $[0, 1]$ is defined as:

$$\mathbf{X}_L := \left\{ \frac{1}{2^L}, \frac{2}{2^L}, \frac{3}{2^L}, \ldots, \frac{2^L - 1}{2^L} \right\}. \tag{13}$$

However, this increasing order does not yield a sparse representation of the Markov kernel $k(\cdot, \cdot)$ on the points $\mathbf{X}_L$, i.e., Cholesky decomposition of the covariance matrix $k(\mathbf{X}_L, \mathbf{X}_L)$ is not sparse. To achieve a sparse hierarchical expansion, we first sort the dyadic points $\mathbf{X}_L$ according to their levels.

Sorted Dyadic Points For level-ℓ dyadic points $\mathbf{X}_\ell$ where $\ell = 1, \ldots, L$, we first define the set $\rho(\ell)$ consisting of odd numbers as follows:

$$\rho(\ell) = \left\{1, 3, 5, \ldots, 2^\ell - 1\right\}. \tag{14}$$

Next, we define the sorted incremental set $\mathbf{D}_\ell$ (with $\mathbf{X}_0 := \varnothing$) as:

$$\mathbf{D}_\ell = \left\{ \frac{i}{2^\ell} : i \in \rho(\ell) \right\} = \mathbf{X}_\ell - \mathbf{X}_{\ell-1}, \quad \ell = 1, \ldots L. \tag{15}$$

Thus, the level-L dyadic points $\mathbf{X}_L$ can be decomposed into disjoint incremental sets $\{\mathbf{D}_\ell\}_{\ell=1}^L$:

$$\mathbf{X}_L = \cup_{\ell=1}^L \mathbf{D}_\ell, \quad \mathbf{D}_i \cap \mathbf{D}_j = \varnothing \text{ for } i \neq j. \tag{16}$$

Therefore, we can define the sorted level-L dyadic points using these incremental sets as:

$$\mathbf{X}_L^{\text{sort}} := \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_L\} = \left\{ \frac{i \in \rho(\ell)}{2^\ell}, \ell = 1, \ldots, L \right\}. \tag{17}$$

For example, the sorted level-3 dyadic points are given by:

$$\mathbf{X}_3^{\text{sort}} = \left\{ \underbrace{\frac{1}{2^1}}_{\mathbf{D}_1}, \underbrace{\frac{1}{2^2}, \frac{3}{2^2}}_{\mathbf{D}_2}, \underbrace{\frac{1}{2^3}, \frac{3}{2^3}, \frac{5}{2^3}, \frac{7}{2^3}}_{\mathbf{D}_3} \right\}. \tag{18}$$

Algorithm We now present the algorithm for computing the inverse of the upper triangular Cholesky factor $[\mathbf{L}_{\mathbf{X}_L^{\text{sort}}}^\top]^{-1}$ of the covariance matrix $k(\mathbf{X}_L^{\text{sort}}, \mathbf{X}_L^{\text{sort}})$ in Algorithm 1, where $\mathbf{L}_{\mathbf{X}_L^{\text{sort}}} \mathbf{L}_{\mathbf{X}_L^{\text{sort}}}^\top = k(\mathbf{X}_L^{\text{sort}}, \mathbf{X}_L^{\text{sort}})$.. The corresponding proof can be found in (Ding et al., 2024). The output of Algorithm 1 is a sparse matrix with $\mathcal{O}(3 \cdot (2^L - 1))$ nonzero entries. Since each iteration of the for-loop only requires solving a 3×3 linear system, which costs $\mathcal{O}(3^3)$ time, the total computational complexity of Algorithm 1 is $\mathcal{O}(2^L - 1)$. This implies that the complexity of computing $\left[\mathbf{L}_{\mathbf{U}}^\top\right]^{-1}$ in eq. (8) is $\mathcal{O}(M)$ when $\mathbf{U}$, the induced grid of size M, consists of sorted dyadic points as defined in eq. (17).

Algorithm 1 Computation of the inverse Cholesky factor for the Markov kernel $k(\cdot,\cdot)$ on sorted one-dimensional level-L dyadic points $\mathbf{X}_L^{\text{sort}}$.

Input: Markov kernel $k(\cdot,\cdot)$, sorted level-L dyadic points $\mathbf{X}_L^{\text{sort}}$
Output: inverse of the upper triangular Cholesky factor $\mathbf{R} := [\mathbf{L}_{\mathbf{X}_L^{\text{sort}}}^\top]^{-1}$, s.t. $\mathbf{L}_{\mathbf{X}_L^{\text{sort}}}\mathbf{L}_{\mathbf{X}_L^{\text{sort}}}^\top = k(\mathbf{X}_L^{\text{sort}}, \mathbf{X}_L^{\text{sort}})$

Initialize $\mathbf{R} \leftarrow \text{zeros}(2^L-1, 2^L-1)$;
Define $k(\pm\infty,\cdot) = k(\cdot,\pm\infty) = 0$;
for $\ell = 1$ **to** L **do**
  for $i \in \rho(\ell) = \{1, 3, \ldots, 2^\ell - 1\}$ **do**
    $x_{\text{mid}} := \frac{i}{2^\ell}$; $x_{\text{left}} := \frac{i-1}{2^\ell}$ **if** $i > 1$ **else** $-\infty$; $x_{\text{right}} := \frac{i+1}{2^\ell}$ **if** $i < 2^\ell - 1$ **else** $+\infty$;
    Get i_{mid}, i_{left}, i_{right}, the indices of the points x_{mid}, x_{left}, x_{right} in the sorted set $\mathbf{X}_L^{\text{sort}}$ respectively;
    Get the coefficients c_1, c_2, c_3 by solving the following linear system:

$$\begin{bmatrix} k(x_{\text{left}}, x_{\text{left}}) & k(x_{\text{left}}, x_{\text{mid}}) & k(x_{\text{left}}, x_{\text{right}}) \\ k(x_{\text{mid}}, x_{\text{left}}) & k(x_{\text{mid}}, x_{\text{mid}}) & k(x_{\text{mid}}, x_{\text{right}}) \\ k(x_{\text{right}}, x_{\text{left}}) & k(x_{\text{right}}, x_{\text{mid}}) & k(x_{\text{right}}, x_{\text{right}}) \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}. \tag{19}$$

    $[c_1, c_2, c_3] := [c_1, c_2, c_3]/\sqrt{c_2}$;
    if $x_{\text{left}} \neq -\infty$, **then** $\mathbf{R}[i_{\text{left}}, i_{\text{mid}}] = c_1$; **if** $x_{\text{right}} \neq +\infty$, **then** $\mathbf{R}[i_{\text{right}}, i_{\text{mid}}] = c_3$;
    $\mathbf{R}[i_{\text{mid}}, i_{\text{mid}}] = c_2$;
  end for
end for

B REPARAMETERIZATION OF KERNEL LENGTHSCALES

Considering the additive Laplace kernel with fixed lengthscale $\tilde{\theta}$ for all base kernels, applying linear projections $\left\{\mathbf{w}_p^\top \mathbf{x}\right\}_{p=1}^P$ on inputs $\mathbf{x} \in \mathbb{R}^D$ will give:

$$\begin{aligned} &\sum_{p=1}^P \sigma_p^2 k_p\left(\mathbf{w}_p^\top \mathbf{x}, \mathbf{w}_p^\top \mathbf{x}'\right) \\ =&\sum_{p=1}^P \sigma_p^2 \exp\left(-\frac{\sum_{d=1}^D |w_{p,d}(x_d - x_d')|}{\tilde{\theta}}\right) \\ =&\sum_{p=1}^P \prod_{d=1}^D \sigma_p^2 \exp\left(-\frac{|x_d - x_d'|}{\tilde{\theta}/|w_{p,d}|}\right) \\ =&\sum_{p=1}^P \prod_{d=1}^D \sigma_p^2 \exp\left(-\frac{|x_d - x_d'|}{\theta_{p,d}}\right), \end{aligned} \tag{20}$$

This still leads to an additive Laplace kernel but with adaptive lengthscale $\theta_{p,d}$ for base kernels. The resulting kernel also retains *sparse* Cholesky decomposition by the properties of Markov kernels so that the complexity of inference is $\mathcal{O}(M)$.

C INFERENCE OF PREDICTIVE DISTRIBUTION

Given an input $\mathbf{x} \in \mathbb{R}^D$, the prediction of the DAK model can be written in the following equation according to eq. (10):

$$\begin{aligned} \tilde{f}_{\mathbf{x}} &= \sum_{p=1}^P \sigma_p\Big(\phi(h_\psi^{[p]}(\mathbf{x}))\mathbf{z}_p\Big) + \mu \\ &= \sum_{p=1}^P \sigma_p\Big(\boldsymbol{\phi}_p^\top \mathbf{z}_p\Big) + \mu, \end{aligned} \tag{21}$$

where $\boldsymbol{\phi}_p^\top := \phi(h_\psi^{[p]}(\mathbf{x})) \in \mathbb{R}^{1\times M}$. We assume the variational distribution over the independent Gaussian weights $\mathbf{z}_p \sim \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$ and the bias $\mu \sim \mathcal{N}(m_\mu, \sigma_\mu^2)$. Then it's straighforward to deduce that

$$\boldsymbol{\phi}_p^\top \mathbf{z}_p + \mu \sim \mathcal{N}\left(\boldsymbol{\phi}_p^\top \boldsymbol{m}_{\mathbf{z}_p} + m_\mu,\ \boldsymbol{\phi}_p^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_p + \sigma_\mu^2\right), \tag{22}$$

$$\sigma_p\left(\boldsymbol{\phi}_p^\top \mathbf{z}_p\right) + \mu \sim \mathcal{N}\left(\sigma_p(\boldsymbol{\phi}_p^\top \boldsymbol{m}_{\mathbf{z}_p}) + m_\mu,\ \sigma_p^2(\boldsymbol{\phi}_p^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_p) + \sigma_\mu^2\right), \tag{23}$$

$$\tilde{f}_{\mathbf{x}} = \sum_{p=1}^{P} \sigma_p\left(\boldsymbol{\phi}_p^\top \mathbf{z}_p\right) + \mu \sim \mathcal{N}\left(\sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_p^\top \boldsymbol{m}_{\mathbf{z}_p}) + m_\mu,\ \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_p^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_p) + \sigma_\mu^2\right). \tag{24}$$

Therefore, we obtain the predictive distribution of the $\tilde{f}(\mathbf{x})$ at the point $\mathbf{x} \in \mathbb{R}^D$ and its mean and variance are given by:

$$\mathbb{E}\left[\tilde{f}_{\mathbf{x}}\right] = \sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_p^\top \boldsymbol{m}_{\mathbf{z}_p}) + m_\mu, \tag{25a}$$

$$\mathrm{Var}\left[\tilde{f}_{\mathbf{x}}\right] = \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_p^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_p) + \sigma_\mu^2. \tag{25b}$$

D TRAINING OF VARIATIONAL INFERENCE

Given the dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ where $\mathbf{X} := \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{y} = (y_1, \ldots, y_N)^\top$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \mathbb{R}$, the prediction $\tilde{f}_{\mathbf{X}} \in \mathbb{R}^N$ of DAK is given by all the parameters $\boldsymbol{\theta} = \{\psi, \boldsymbol{\sigma}\}$, $\boldsymbol{\eta} = \left\{\{\mathbf{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}\}_{p=1}^P, \{m_\mu, \sigma_\mu\}\right\}$ according to eq. (10):

$$\tilde{f}_{\mathbf{X}} := \tilde{f}(\mathbf{X}; \boldsymbol{\theta}, \boldsymbol{\eta}) = \sum_{p=1}^{P} \sigma_p\Big(\phi(h_\psi^{[p]}(\mathbf{X}))\mathbf{z}_p\Big) + \mu, \tag{26}$$

where $\mathbf{z}_p \sim \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$, $p = 1, \ldots, P$, and $\mu \sim \mathcal{N}(m_\mu, \sigma_\mu^2)$ are variational variables Θ_{var} parameterized by $\boldsymbol{\eta}$. The variational distribution is denoted by $q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) = q(\mu)\prod_{p=1}^P q(\mathbf{z}_p) = \mathcal{N}(m_\mu, \sigma_\mu^2)\prod_{p=1}^P \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$, and the variational prior is denoted by $p(\Theta_{\text{var}})$.

We consider the KL divergence between $q_{\boldsymbol{\eta}}(\Theta_{\text{var}})$ and the true posterior $p(\Theta_{\text{var}}|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta})$:

$$\begin{aligned}
&\mathrm{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}}|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta})\right] \\
&= \int q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \log \frac{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}{p(\Theta_{\text{var}}|\mathbf{y}, \mathbf{X}, \boldsymbol{\theta})} d\Theta_{\text{var}} \\
&= \int q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \log \frac{q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}, \Theta_{\text{var}}) p(\Theta_{\text{var}})} d\Theta_{\text{var}} \\
&= \int q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \log \frac{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}{p(\Theta_{\text{var}})} d\Theta_{\text{var}} - \int q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \log p(\mathbf{y}|\tilde{f}_{\mathbf{X}}) d\Theta_{\text{var}} + \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}).
\end{aligned} \tag{27}$$

Using the fact that $\mathrm{KL}[\cdot\|\cdot] \geq 0$, we have

$$\begin{aligned}
\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) &\geq \int q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \log p(\mathbf{y}|\tilde{f}_{\mathbf{X}}) d\Theta_{\text{var}} - \mathrm{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log p(\mathbf{y}|\tilde{f}_{\mathbf{X}})\right] - \mathrm{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}})\right].
\end{aligned} \tag{28}$$

Full-training. Firstly, we present the joint training of $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$. The most common approach optimizes the marginal log-likelihood (the left-hand side of eq. (28)):

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) \tag{29}$$

$$= \arg\max_{\boldsymbol{\theta}} \log \int p\left(y|X, \boldsymbol{\theta}, \Theta_{\text{var}}\right) p(\Theta_{\text{var}}) d\Theta_{\text{var}}, \tag{30}$$

which involves intractable integral in some tasks such as classification. Instead, we optimize the variational lower bound (the right-hand side of eq. (28)):

$$\Theta^* := \arg\max_{\boldsymbol{\theta},\boldsymbol{\eta}} \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\eta}) = \arg\max_{\boldsymbol{\theta},\boldsymbol{\eta}} \left\{ E_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})} \left[\log p(\mathbf{y}|\tilde{f}_{\mathbf{X}}) \right] - \text{KL}\left[q_{\boldsymbol{\eta}}(\Theta_{\text{var}}) \| p(\Theta_{\text{var}}) \right] \right\}. \tag{31}$$

Fine-tuning. An alternative training approach is to firstly pre-train the deterministic parameters of feature extractor by standard neural network training, with mean squared error for regression or cross-entropy for classification as the loss function, and then fine-tune the last layer additive GP with fixed features. The objective function is identical to eq. (12), but $\boldsymbol{\theta}$ is learned during the pre-training step and is no longer optimized during fine-tuning.

E ELBO

E.1 Assumptions

Consider the model $y_i = \tilde{f}(\mathbf{x}_i) + \epsilon_i$ with the i.i.d. noise $\epsilon_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_f^2)$ and $\tilde{f} : \mathbb{R}^D \to \mathbb{R}$ is defined in eq. (10). The training dataset is $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ where $\mathbf{X} := \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{y} = (y_1, \ldots, y_N)^\top$, $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \mathbb{R}$. $\Theta_{\text{var}} := \{\mu, \{\mathbf{z}_p\}_{p=1}^P\}$ are the variational random variables consisting of Gaussian weights and bias of P units, ψ are the parameters of the NN, $\boldsymbol{\sigma} := (\sigma_1, \ldots, \sigma_p)^\top$ are the scale parameters of base GP layers. The variational distributions are $q(\mu) = \mathcal{N}(m_\mu, \sigma_\mu^2)$, $q(\mathbf{z}_p) = \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$ and the variational priors are $p(\mu) = \mathcal{N}(\check{m}_\mu, \check{\sigma}_\mu^2)$, $p(\mathbf{z}_p) = \mathcal{N}(\check{\boldsymbol{m}}_{\mathbf{z}_p}, \check{\mathbf{S}}_{\mathbf{z}_p})$. Note that $\mathbf{S}_{\mathbf{z}_p} \in \mathbb{R}^{M\times M}$ is a diagonal covariance matrix due to the independence of $\mathbf{z}_p$, M is the number of inducing points $\mathbf{U}$ defined in eq. (8), and $\boldsymbol{m}_{\mathbf{z}_p} \in \mathbb{R}^M$, $m_\mu \in \mathbb{R}$, $\sigma_\mu^2 \in \mathbb{R}$. We derive the ELBO in VI to learn the preditive posterior over the variational variables $\Theta_{\text{var}} := \{\mu, \{\mathbf{z}_p\}_{p=1}^P\}$ parameterized by $\boldsymbol{\eta} := \left\{\{\mathbf{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}\}_{p=1}^P, \{m_\mu, \sigma_\mu\}\right\}$, and optimize the deterministic parameters $\boldsymbol{\theta} := \{\psi, \boldsymbol{\sigma}\}$.

E.2 Expected Log Likelihood

Closed Form The *expected log likelihood*, which is the first term in ELBO defined in eq. (12), is given by

$$\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \Pr(\mathbf{y}|\tilde{f}_{\mathbf{X}})\right] &= \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \prod_{i=1}^N p(y_i|\tilde{f}_{\mathbf{x}_i})\right] \\
&= \sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log p(y_i|\tilde{f}_{\mathbf{x}_i})\right] \\
&= \sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \mathcal{N}(\tilde{f}_i,\, \sigma_f^2)\right] \\
&= \sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log\left((2\pi\sigma_f^2)^{-\frac{1}{2}} \exp\left\{-\frac{(y_i-\tilde{f}_i)^2}{2\sigma_f^2}\right\}\right)\right] \\
&= \sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_f^2) - \frac{1}{2\sigma_f^2}(y_i-\tilde{f}_i)^2\right] \\
&= -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_f^2) - \frac{1}{2\sigma_f^2}\sum_{i=1}^N \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[(y_i-\tilde{f}_i)^2\right] \\
&= -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_f^2) - \frac{1}{2\sigma_f^2}\sum_{i=1}^N \left(\left(\mathbb{E}_{q(\Theta_{\text{var}})}\left[(y_i-\tilde{f}_i)\right]\right)^2 + \text{Var}_{q(\Theta_{\text{var}})}\left[(y_i-\tilde{f}_i)\right]\right),
\end{aligned} \tag{32}$$

where

$$\tilde{f}_i = \sum_{p=1}^{P} \sigma_p\Big(\underbrace{\phi(h_\psi^{[p]}(\mathbf{x}_i))}_{:=\boldsymbol{\phi}_{i,p}^\top \in \mathbb{R}^{1\times M}} \mathbf{z}_p\Big) + \mu$$
$$= \sum_{p=1}^{P} \sigma_p\left(\boldsymbol{\phi}_{i,p}^\top \mathbf{z}_p\right) + \mu. \tag{33}$$

Recall that the variational assumptions $q(\mathbf{z}_p) = \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$ and $q(\mu) = \mathcal{N}(m_\mu, \sigma_\mu^2)$, we can infer that

$$\boldsymbol{\phi}_{i,p}^\top \mathbf{z}_p + \mu \sim \mathcal{N}\left(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p} + m_\mu,\ \boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p} + \sigma_\mu^2\right), \tag{34}$$
$$\sigma_p\left(\boldsymbol{\phi}_{i,p}^\top \mathbf{z}_p\right) + \mu \sim \mathcal{N}\left(\sigma_p(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p}) + m_\mu,\ \sigma_p^2(\boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p}) + \sigma_\mu^2\right), \tag{35}$$
$$\tilde{f}_i = \sum_{p=1}^{P} \sigma_p\left(\boldsymbol{\phi}_{i,p}^\top \mathbf{z}_p\right) + \mu \sim \mathcal{N}\left(\sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p}) + m_\mu,\ \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p}) + \sigma_\mu^2\right), \tag{36}$$
$$y_i - \tilde{f}_i \sim \mathcal{N}\left(y_i - \sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p}) - m_\mu,\ \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p}) + \sigma_\mu^2\right). \tag{37}$$

Therefore,

$$\left(\mathbb{E}_{q(\Theta_{\text{var}})}\left[(y_i - \tilde{f}_i)\right]\right)^2 = \left(y_i - \sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p}) - m_\mu\right)^2, \tag{38a}$$

$$\mathrm{Var}_{q(\Theta_{\text{var}})}\left[(y_i - \tilde{f}_i)\right] = \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p}) + \sigma_\mu^2. \tag{38b}$$

By applying eq. (38) to eq. (32), we derive the analytical formula for the expected evidence, expressed as

$$\mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \Pr(\mathbf{y}|\tilde{f}_{\mathbf{X}})\right] = -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_f^2)$$
$$- \frac{1}{2\sigma_f^2}\sum_{i=1}^{N}\left(\left(y_i - \sum_{p=1}^{P} \sigma_p(\boldsymbol{\phi}_{i,p}^\top \boldsymbol{m}_{\mathbf{z}_p}) - m_\mu\right)^2 + \sum_{p=1}^{P} \sigma_p^2(\boldsymbol{\phi}_{i,p}^\top \mathbf{S}_{\mathbf{z}_p} \boldsymbol{\phi}_{i,p}) + \sigma_\mu^2\right). \tag{39}$$

Monte Carlo Approximation For comparison, we provide the equation for computing the Monte Carlo estimate of the ELBO in the paragraph that follows.

$$\mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log \Pr(\mathbf{y}|\tilde{f}_{\mathbf{X}})\right] = \sum_{i=1}^{N} \mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})}\left[\log p(y_i|\tilde{f}_{\mathbf{x}_i})\right]$$
$$\approx \sum_{i=1}^{N}\frac{1}{S}\sum_{s=1}^{S} \log p(y_i|\mathbf{x}_i, \tilde{\Theta}_{\text{var}}^{(s)}, \boldsymbol{\theta})$$
$$= \frac{1}{S}\sum_{i=1}^{N}\sum_{s=1}^{S} \log \mathcal{N}(y_i \mid \tilde{f}_i^{(s)}, \sigma_f^2)$$
$$= \frac{1}{S}\sum_{i=1}^{N}\sum_{s=1}^{S} \log\left((2\pi\sigma_f^2)^{-\frac{1}{2}} \exp\left\{-\frac{(y_i - \tilde{f}_i^{(s)})^2}{2\sigma_f^2}\right\}\right)$$
$$= \frac{1}{S}\sum_{i=1}^{N}\sum_{s=1}^{S}\left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_f^2) - \frac{1}{2\sigma_f^2}(y_i - \tilde{f}_i^{(s)})^2\right)$$
$$= -\frac{N}{2}\log(2\pi) - \frac{N}{2}\log(\sigma_f^2) - \frac{1}{2\sigma_f^2}\sum_{i=1}^{N}\frac{1}{S}\sum_{s=1}^{S}(y_i - \tilde{f}_i^{(s)})^2, \tag{40}$$

where S is the number of Monte Carlo samples, $\{\tilde{\mu}^{(s)}, \{\tilde{\mathbf{z}}_p^{(s)}\}_{p=1}^P\} := \tilde{\Theta}_{\text{var}}^{(s)}$ are the s-th Monte Carlo samplings over the variational parameters Θ_{var} and $\tilde{\Theta}_{\text{var}}^{(s)} \sim q_{\boldsymbol{\eta}}(\Theta_{\text{var}})$, $\tilde{f}_i^{(s)}$ is given as follows:

$$\begin{aligned}\tilde{f}_i^{(s)} &:= \tilde{f}(\mathbf{x}_i; \tilde{\Theta}_{\text{var}}^{(s)}, \boldsymbol{\theta}) \\ &= \sum_{p=1}^{P} \sigma_p \Big(\underbrace{\phi(h_\psi^{[p]}(\mathbf{x}_i))}_{:=\boldsymbol{\phi}_{i,p}^\top \in \mathbb{R}^{1\times M}} \tilde{\mathbf{z}}_p^{(s)} \Big) + \tilde{\mu}^{(s)} \\ &= \sum_{p=1}^{P} \sigma_p \left(\boldsymbol{\phi}_{i,p}^\top \tilde{\mathbf{z}}_p^{(s)} \right) + \tilde{\mu}^{(s)}. \end{aligned} \tag{41}$$

Therefore, we plug eq. (41) into eq. (40) and get the the Monte Carlo estimate of the ELBO written in the following formula:

$$\mathbb{E}_{q_{\boldsymbol{\eta}}(\Theta_{\text{var}})} \left[\log \Pr(\mathbf{y} | \tilde{f}_{\mathbf{X}}) \right] \approx -\frac{N}{2} \log(2\pi) - \frac{N}{2} \log(\sigma_f^2) - \frac{1}{2\sigma_f^2} \sum_{i=1}^{N} \frac{1}{S} \sum_{s=1}^{S} \left(y_i - \sum_{p=1}^{P} \sigma_p \left(\boldsymbol{\phi}_{i,p}^\top \tilde{\mathbf{z}}_p^{(s)} \right) - \tilde{\mu}^{(s)} \right)^2, \tag{42}$$

$$\tilde{\mathbf{z}}_p^{(s)} \sim \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p}), \qquad \tilde{\mu}^{(s)} \sim \mathcal{N}(m_\mu, \sigma_\mu^2). \tag{43}$$

E.3 KL Divergence

Since we place Gaussian assumptions over the variational parameters Θ_{var}, the *KL divergence*, which is the second term in ELBO defined in eq. (12), is then given by

$$\begin{aligned}\text{KL}\left[q(\Theta_{\text{var}}) \| p(\Theta_{\text{var}})\right] &= \text{KL}\left[q(\mu, \{\mathbf{z}_p\}_{p=1}^P) \| p(\mu, \{\mathbf{z}_p\}_{p=1}^P)\right] \\ &= \text{KL}\left[q(\mu) \| p(\mu)\right] + \sum_{p=1}^{P} \text{KL}\left[q(\mathbf{z}_p) \| p(\mathbf{z}_p)\right], \end{aligned} \tag{44}$$

$$\text{KL}\left[q(\mu) \| p(\mu)\right] = \frac{1}{2} \left(\frac{\sigma_\mu^2}{\check{\sigma}_\mu^2} + \frac{(m_\mu - \check{m}_\mu)^2}{\check{\sigma}_\mu^2} - \log\left(\frac{\sigma_\mu^2}{\check{\sigma}_\mu^2}\right) - 1 \right), \tag{45}$$

$$\text{KL}\left[q(\mathbf{z}_p) \| p(\mathbf{z}_p)\right] = \frac{1}{2} \sum_{i=1}^{M} \left(\frac{[\mathbf{S}_{\mathbf{z}_p}]_{ii}}{[\check{\mathbf{S}}_{\mathbf{z}_p}]_{ii}} + \frac{([\boldsymbol{m}_{\mathbf{z}_p}]_i - [\check{\boldsymbol{m}}_{\mathbf{z}_p}]_i)^2}{[\check{\mathbf{S}}_{\mathbf{z}_p}]_{ii}} - \log\left(\frac{[\mathbf{S}_{\mathbf{z}_p}]_{ii}}{[\check{\mathbf{S}}_{\mathbf{z}_p}]_{ii}}\right) - 1 \right), \tag{46}$$

where $[\mathbf{S}_{\mathbf{z}_p}]_{ii}$ is the (i,i)-th element of the diagonal covariance matrix $\mathbf{S}_{\mathbf{z}_p} \in \mathbb{R}^{M\times M}$, $[\boldsymbol{m}_{\mathbf{z}_p}]_i$ is the i-th element of the mean vector $\boldsymbol{m}_{\mathbf{z}_p} \in \mathbb{R}^M$, the approximated posteriors are $q(\mu) = \mathcal{N}(m_\mu, \sigma_\mu^2)$, $q(\mathbf{z}_p) = \mathcal{N}(\boldsymbol{m}_{\mathbf{z}_p}, \mathbf{S}_{\mathbf{z}_p})$ and the priors are $p(\mu) = \mathcal{N}(\check{m}_\mu, \check{\sigma}_\mu^2)$, $p(\mathbf{z}_p) = \mathcal{N}(\check{\boldsymbol{m}}_{\mathbf{z}_p}, \check{\mathbf{S}}_{\mathbf{z}_p})$.

E.4 Limitations of the Closed-Form ELBO

The closed-form ELBO is only applicable to regression problems. In classification, applying the softmax function to $\tilde{f}(\mathbf{x}; \boldsymbol{\theta}, \boldsymbol{\eta})$ results in a non-analytic predictive distribution, meaning the ELBO must still be computed via Monte Carlo sampling during training. Similarly, the closed-form expressions for the predictive mean and variance, as provided in eq. (25) in Appendix C, are not applicable to classification but only apply to regression problems.

F COMPUTATIONAL COMPLEXITY

In this section, we discuss the computational complexity of various DKL models compared to the proposed DAK method, focusing on the GP layer as the most computationally demanding component. Table 5 shows the computational complexity of our model compared to other state-of-the-art GP and DKL methods.

Table 5: Computational complexity of the DKL models for N training points. The reported training complexity is for one iteration. $\hat{M}$ is the number of inducing points in SVGP and KISS-GP, while M is the size of induced grids in DAK, $M < \hat{M}$. S is the number of Monte Carlo samples, B is the size of mini-batch, D_w is the dimension of the NN outputs in DKL, P is the dimension of the outputs after applying linear transformations to the NN outputs in the proposed DAK model. DAK-MC refers to the DAK model using Monte Carlo approximation, while DAK-CF refers to the DAK model using closed-form inference and ELBO.

	Inference	**Training** (per iteration)
NN + SVGP	$\mathcal{O}(\hat{M}^2 N)$	$\mathcal{O}(SD_w MB + \hat{M}^3)$
NN + KISS-GP	$\mathcal{O}(D_w \hat{M}^{1+\frac{1}{D_w}})$	$\mathcal{O}(SD_w MB + D_w \hat{M}^{\frac{3}{D_w}})$
DAK-MC (ours)	$\mathcal{O}(SM)$	$\mathcal{O}(SPMB + PM)$
DAK-CF (ours)	$\mathcal{O}(M)$	$\mathcal{O}(PMB + PM)$

Inference Complexity. In inference based on induced approximation, computing the multiplication of the inverse of the covariance matrix $k(\mathbf{U}, \mathbf{U})$ and a vector takes $\mathcal{O}(\hat{M}^2 N)$ time for $\hat{M}$ inducing points $\mathbf{U}$ and N training points when using SVGP. This cost is reduced by KISS-GP to $\mathcal{O}(D\hat{M}^{1+\frac{1}{D}})$ by decomposing the covariance matrix into a Kronecker product of D one-dimensional covariance matrices of the inducing points: $k(\mathbf{U}, \mathbf{U}) = \bigotimes_{d=1}^{D} k(\mathbf{U}^{[d]}, \mathbf{U}^{[d]})$. Despite the significant reduction on complexity, it requires inducing points $\mathbf{U}$ arranged on a Cartesian grid of size $\hat{M} = \prod_{d=1}^{D} \hat{M}_d$, where $\hat{M}_d$ is the number of inducing points in the d-th dimension. In high-dimensional spaces, fixing $\hat{M}$ leads to very small $\hat{M}_d$ per dimension, which can degrade model performance. To address this, we propose the DAK model via sparse finite-rank approximation, which employs an additive Laplace kernel for GPs. The inverse Cholesky factor $\mathbf{L}_{\mathbf{U}}^{\top}$ for one-dimensional induced grids $\mathbf{U}$ of size M, where $M < \hat{M}$, as defined in eq. (8), is sparse and can be computed in $\mathcal{O}(M)$ time.

Training Complexity. In training, VI requires computing the ELBO as described in eq. (12), which consists of two terms: the *expected log likelihood* and the *KL divergence* between the variational distributions and priors.

1) The *expected log likelihood* is usually approximated via Monte Carlo sampling at a cost of $\mathcal{O}(SN_\Theta N)$, where S is the number of Monte Carlo samples, N_Θ is the total number of variational parameters Θ_{var}, and N is the number of training points. This complexity can be reduced to $\mathcal{O}(SN_\Theta B)$ by applying stochastic variational inference with a mini-batch of size $B \ll N$. For DKL models using SVGP and KISS-GP, Θ_{var} are inducing variables, and the expectation does not have a closed form, requiring Monte Carlo sampling. In contrast, in the proposed DAK model, $\Theta_{\text{var}} = \{\{\mathbf{z}_p\}_{p=1}^{P}, \mu\}$ consists of independent Gaussian weights $\mathbf{z}_p \in \mathbb{R}^M$ and bias μ. This allows us to derive an analytical form for this term, as shown in eq. (39) in Appendix E, reducing the computational cost to $\mathcal{O}(N_\Theta B) = \mathcal{O}(PMB)$ when using a mini-batch of size B.

2) The *KL divergence* between two Gaussian distributions can be computed in closed form. This leads to a linear time complexity of $\mathcal{O}(N_\Theta)$ if the parameters Θ_{var} are independent, or cubic time $\mathcal{O}(N_\Theta^3)$ if they are fully correlated. In SVGP and KISS-GP, Θ_{var} represents fully correlated Gaussian distributed inducing variables, so computing the KL divergence takes $\mathcal{O}(\hat{M}^3)$ for SVGP. In KISS-GP, this can be reduced to $\mathcal{O}(D\hat{M}^{\frac{3}{D}})$ using fast eigendecomposition of Kronecker matrices. In the DAK model, the weights $\{\mathbf{z}_p\}_{p=1}^{P}$ as defined in eq. (8) are independent Gaussian random variables, allowing the KL divergence to be computed in $\mathcal{O}(N_\Theta) = \mathcal{O}(PM)$ time, where P is the number of base GP layers.

G EXPERIMENTAL DETAILS

In this section, we provide additional details regarding the experiments.

G.1 Benchmarks for Regression

Experiment Setup For all models, the NN architecture is a fully connected NN with rectified linear unit (ReLU) activation function (Nair and Hinton, 2010) and two hidden layers containing 64 and 32 neurons, respectively, structured as $D \to 64 \to 32 \to D_w$, where D is the input feature size (also the size of input $\mathbf{X}$) and D_w is the output feature size. The models are evaluated with $D_w = 16$, 64, and 256, respectively. The number

of Monte Carlo samples is set to 8 during training and 20 during inference.

The NN is a deterministic model, and we use the negative Gaussian log-likelihood as the loss function to quantify the uncertainty of the NN outputs and compute the NLPD.

For NN+SVGP, the inducing points are set to the size of 64 in D_w dimension. We implement the `ApproximateGP` model in GPyTorch (Gardner et al., 2018), defining the inducing variables as variational parameters, and use `VariationalELBO` in GPyTorch to perform variational inference and compute the loss.

SV-DKL is originally designed for classification, so for a fair comparison in regression tasks, we modify it by first applying a linear embedding layer $\mathbf{W} : \mathbb{R}^{D_w} \to \mathbb{R}^P$ with $P = 16$ and normalizing the outputs to the interval $[0, 1]$ for each base GP, similar to the DAK model. To adapt the additive GP layer for regression, we remove the softmax function from the model in eq. (1) of (Wilson et al., 2016b). Given training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, the model is modified as follows:

$$p(\mathbf{y}_i|\mathbf{f}_i, A) = \mathcal{A}(\mathbf{f}_i)^\top \mathbf{y}_i \tag{47}$$

where $\mathbf{f}_i \in \mathbb{R}^P$ is a vector of independent GPs followed by a linear mixing layer $\mathcal{A}(\mathbf{f}_i) = A\mathbf{f}_i$, with $A \in \mathbb{R}^{C\times P}$ as the transformation matrix. Here, $C = 1$ for single-task regression. For each p-th GP ($1 \le p \le P$) in the additive GP layer, the corresponding inducing variables $\mathbf{u}_p$ are set to the size of 64 and treated as variational parameters for training. We use the `GridInterpolationVariationalStrategy` model with `LMCVariationalStrategy` in GPyTorch to perform KISS-GP with variational inducing variables, augmented by a linear mixing layer.

Both DAK-MC and DAK-CF use the same additive GP layer size as SV-DKL, with $P = 16$, and employ fixed induced grids $\mathbf{U} = \{1/8, 2/8, \ldots, 7/8\}$ of size 7 for each base GP, which is much smaller than that of SV-DKL.

Metrics Let $\{\mathbf{x}_t, y_t\}_{t=1}^T$ represent a test dataset of size T, where μ_t and σ_t^2 are the predictive mean and variance. We evaluate model performance using two common metrics: Root Mean Squared Error (RMSE) and Negative Log Predictive Density (NLPD).

RMSE is widely used to assess the accuracy of predictions, measuring how far predictions deviate from the true target values. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \mu_t)^2}. \tag{48}$$

NLPD is a standard probabilistic metric for evaluating the quality of a model's uncertainty quantification. It represents the negative log likelihood of the test data given the predictive distribution. For GPs, NLPD is calculated as:

$$\text{NLPD} = -\sum_{t=1}^{T} \log p(y_t = \mu_t|\mathbf{x}_t) \tag{49}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\Big[\frac{(y_t - \mu_t)^2}{2\sigma_t^2} + \frac{1}{2}\log(2\pi\sigma_t^2)\Big]. \tag{50}$$

Both RMSE and NLPD are widely used in the GP regression literature, where smaller values indicate better model performance.

Computing Infrastructure The experiments for regression were run on Macbook Pro M1 with 8 cores and 16GB RAM.

G.2 Benchmarks for Classification

We use PyTorch (Paszke et al., 2019) baseline of NN models, GPyTorch (Gardner et al., 2018) baseline of SVGP and SV-DKL models. In classification tasks, we apply a softmax likelihood to normalize the output digits to probability distributions. The NN is a deterministic model trained via negative log-likelihood loss, while DKL

and DAK models are trained via ELBO loss. The setting of all training tasks are described in Table 6 and Table 7.

SVGP is originally designed for single-output regression. To make it fit for multi-output classification, we used `IndependentMultitaskVariationalStrategy` in GPyTorch to implement the multi-task `ApproximateGP` model, and use `VariationalELBO` with `SoftmaxLikelihood` in GPyTorch to perform variational inference and compute the loss.

For SV-DKL, we employed the same `VariationalELBO` with `SoftmaxLikelihood` as the variational loss objective. `GridInterpolationVariationalStrategy` is applied within `IndependentMultitaskVariationalStrategy` to perform additive KISS-GP approximation. For each KISS-GP unit, we used 64 variational inducing points initialized on a grid of size $[-1, 1]$.

For DAK, we implemented DAK-MC using Monte Carlo estimation given the intractable softmax likelihood. We employed fixed induced grids $\mathbf{U} = \{-31/32, -30/32, \ldots, 30/32, 31/32\}$ of size 63 for each base GP component.

Table 6: Model architectures for image classification on MNIST, CIFAR-10 and CIFAR-100.

Model	Hyper-parameter	MNIST	CIFAR-10	CIFAR-100
NN+SVGP	Feature extractor	CNN	ResNet-18	ResNet-34
	NN out features D_w	128	512	512
	Embedding features P	16	64	128
	# inducing points $\hat{M}$	512	512	512
	# epochs	20	200	200
	Training strategy	Full-training	Full-training	Fine-tuning
SV-DKL	Feature extractor	CNN	ResNet-18	ResNet-34
	NN out features D_w	128	512	512
	Embedding features P	16	64	128
	# inducing points $\hat{M}$	64	64	64
	Grid bounds	[-1,1]	[-1,1]	[-1,1]
	# epochs	20	200	200
	Training strategy	Full-training	Full-training	Fine-tuning
DAK	Feature extractor	CNN	ResNet-18	ResNet-34
	NN out features D_w	128	512	512
	Embedding features P	16	64	128
	# induced interpolation M	63	63	63
	# epochs	20	200	200
	Training strategy	Full-training	Full-training	Full-training

MNIST We used a CNN implemented in PyTorch as the feature extractor: `Conv2d`(1,32,3) $\rightarrow$ `Conv2d`(32,64,3) $\rightarrow$ `MaxPool2d`(2) $\rightarrow$ `Dropout`(0.25) $\rightarrow$ `Linear`(9216,128) $\rightarrow$ `Dropout`(0.5). To make a fair comparison, for both SV-DKL and DAK, we applied an embedding module through a linear layer that transform 128 output features into $P = 16$ base GP channels.

CIFAR-10 We used a ResNet-18 as the feature extractor followed by a linear embedding layer that compressed the 512 output features into $P = 64$ base GP channels.

CIFAR-100 We used a pretrained ResNet-34 as the feature extractor for SV-DKL and fine-tuned GP output layers since SV-DKL struggled to fit using full-training. For proposed DAK, we used full-training. The number of base GP channels is selected as $P = 128$.

Metrics We evaluate model performance using four common metrics: Top-1 accuracy, ELBO, Negative Log Likelihood (NLL), and Expected Calibration Error (ECE).

ECE is a metric used to quantify the degree of "calibration" of a probabilistic model in UQ, specifically for classification problems. It is defined as the weighted average of the absolute difference between the model's predicted probability (confidence) and the actual outcome (accuracy) over several bins of predicted probability.

Table 7: Details of training optimizer for image classification on MNIST, CIFAR-10 and CIFAR-100.

Optimization	MNIST	CIFAR-10	CIFAR-100
Optimizer	Adadelta	SGD	SGD
Initial lr.	1.0	0.1	0.1
Weight decay	0.0001	0.0001	0.0001
Scheduler	StepLR	CosineAnnealingLR	CosineAnnealingLR
Data Augmentation	MNIST	CIFAR-10	CIFAR-100
RandomCrop	-	size=32, padding=4	size=32, padding=4
HorizontalFlip	-	p=0.5	p=0.5

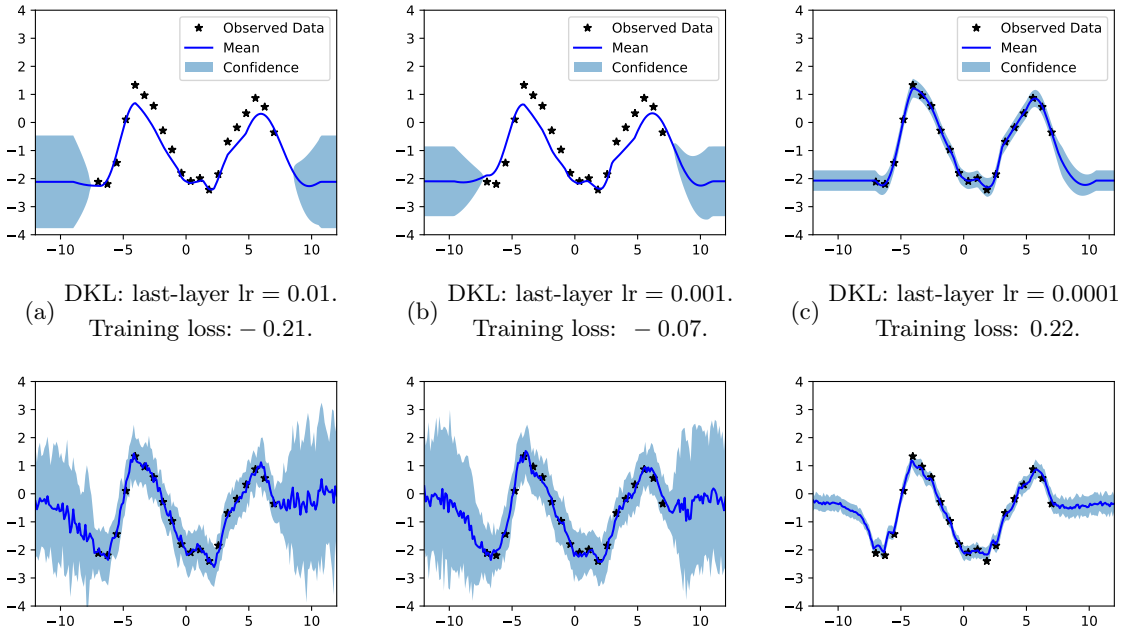
Mathematically, ECE is given by:

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\right|, \tag{51}$$

where M is the number of bins into which the confidence values are partitioned, B_m is the set of indices of samples whose predicted confidence falls into the m-th bin, n is the total number of samples.

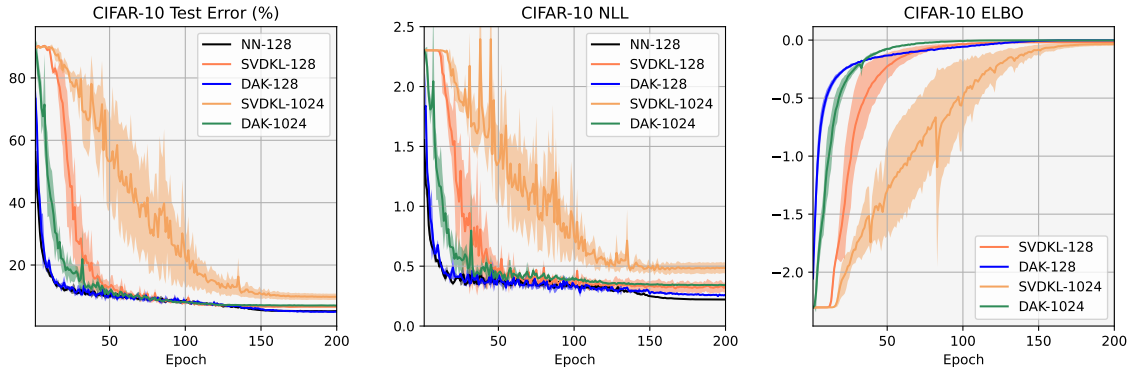
Computing Infrastructure The experiments for classification were run on a Linux machine with NVIDIA RTX4080 GPU, and 32GB of RAM.

G.3 Additional Tables and Figures



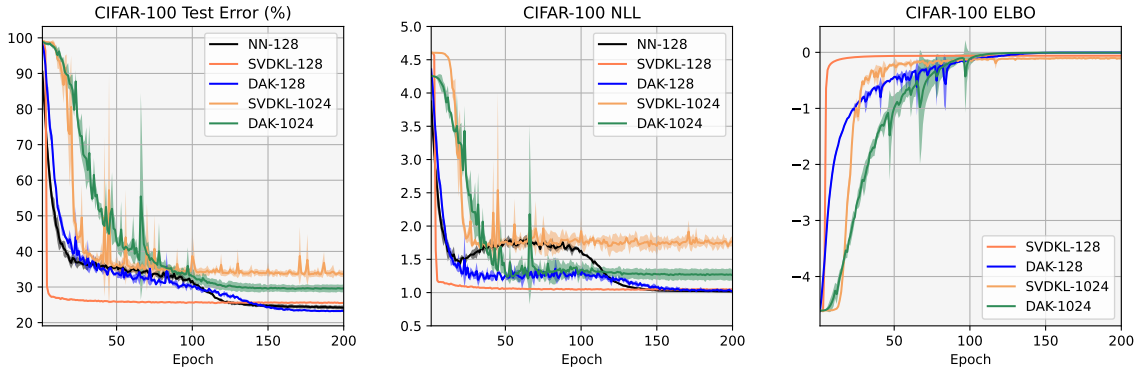
(a) DKL: last-layer lr = 0.01. Training loss: − 0.21.
(b) DKL: last-layer lr = 0.001. Training loss: − 0.07.
(c) DKL: last-layer lr = 0.0001. Training loss: 0.22.
(d) DAK: last-layer lr = 0.1. Training loss: 0.10.
(e) DAK: last-layer lr = 0.01. Training loss: 0.10.
(f) DAK: last-layer lr = 0.001. Training loss: 0.22.

Figure 3: Results on 1D regression with different last-layer learning rates. The learning rate of NN feature extractor is set as 0.01. (a)–(f) shows the regression fits and corresponding training losses. DAK fits for the same learning rate strategy with NN feature extractor (lr=0.01), while DKL requires a separate tuning for last-layer learning rate of GPs. Additionally, a better training loss does not necessarily prevent overfitting for DKL.



(a) Test Error (%). (b) Test NLL. (c) ELBO.

Figure 4: Test errors, test NLLs, ELBOs of NN, SVDKL, and DAK curves with batch size of 128/1024 for CIFAR-10 averaged on 3 runs. DAK outperforms SVDKL on both test error and NLL along the training epochs. Additionally, SVDKL degrades more and struggles to fit when the batch size becomes larger.



(a) Test Error (%). (b) Test NLL. (c) ELBO.

Figure 5: Test errors, test NLLs, ELBOs of NN, SVDKL, and DAK curves with batch size of 128/1024 for CIFAR-100 averaged on 3 runs. DAK trained NN and last-layer additive GPs jointly, while SVDKL used the pre-trained NN and fine-tuned the last-layer GP since SVDKL struggles to fit using full-training. DAK outperforms SVDKL on both test error and NLL along the training epochs. SVDKL struggled to fit in high-dimensional multitask cases, indicating the necessity of pre-training in SVDKL. However, DAK fitted well with high dimensionality and large batch sizes.