A Proof of Lemma 2

As stated in Section 4.1 and the previous section, Theorem 1 relies on Lemma 2 which states that all entries in $\boldsymbol{z}^{(1)}$ (the representation extracted by the neural network) are independent with each other. To prove this lemma, we can explicitly write the k^{th} entry in $\boldsymbol{z}^{(1)}$ as follows:

$$z_k^{(1)}(\cdot) = b_k^{(1)} + \sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(\cdot)\right), \tag{18}$$

where $W_{kr}^{(1)}$ and $b_k^{(1)}$ are i.i.d. Gaussian weight and bias parameters and $z'^{(1)}_r$ is the r^{th} entry in the previous layer. We see that $z_k^{(1)}$ is a linear combination of i.i.d. Gaussian terms, and we can show that $z_k^{(1)}(x_i)$ and $z_l^{(1)}(x_j)$ are independent for two different inputs x_i and x_j and $k \neq l$ by computing the covariance as shown below.

$$\begin{aligned}
&\operatorname{cov}\left(z_k^{(1)}(x_i), z_l^{(1)}(x_j)\right) \\
&= \mathbb{E}\left[\left(b_k^{(1)} + \textstyle\sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right)\left(b_l^{(1)} + \textstyle\sum_{s=1}^{N_{z^{(1)}}} W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right)\right] \\
&= \mathbb{E}\left[b_k^{(1)} b_l^{(1)}\right] + \mathbb{E}\left[b_k^{(1)} \sum_{s=1}^{N_{z^{(1)}}} W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right] + \mathbb{E}\left[b_l^{(1)} \sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right] \\
&+ \mathbb{E}\left[\left(\sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right)\left(\sum_{s=1}^{N_{z^{(1)}}} W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right)\right]
\end{aligned}$$

Since the weights and biases are all i.i.d. Gaussian with zero mean *a priori* and they are independent of the outputs from the previous layer, we have:

$$\begin{aligned}
&\mathbb{E}\left[b_k^{(1)} b_l^{(1)}\right] = \mathbb{E}\left[b_k^{(1)}\right]\mathbb{E}\left[b_l^{(1)}\right] = 0 \\
&\mathbb{E}\left[b_k^{(1)} \textstyle\sum_{s=1}^{N_{z^{(1)}}} W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right] = \mathbb{E}\left[b_k^{(1)}\right] \textstyle\sum_{s=1}^{N_{z^{(1)}}} \mathbb{E}\left[W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right] = 0 \\
&\mathbb{E}\left[b_l^{(1)} \textstyle\sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right] = \mathbb{E}\left[b_l^{(1)}\right] \textstyle\sum_{r=1}^{N_{z^{(1)}}} \mathbb{E}\left[W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right] = 0 \\
&\mathbb{E}\left[\left(\textstyle\sum_{r=1}^{N_{z^{(1)}}} W_{kr}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right)\right)\left(\textstyle\sum_{s=1}^{N_{z^{(1)}}} W_{ls}^{(1)} \phi\left(z'^{(1)}_s(x_j)\right)\right)\right] \\
&= \sum_{r=1}^{N_{z^{(1)}}} \sum_{s=1}^{N_{z^{(1)}}} \mathbb{E}\left[W_{kr}^{(1)} W_{ls}^{(1)} \phi\left(z'^{(1)}_r(x_i)\right) \phi\left(z'^{(1)}_s(x_j)\right)\right] \\
&= \sum_{r=1}^{N_{z^{(1)}}} \sum_{s=1}^{N_{z^{(1)}}} \mathbb{E}\left[W_{kr}^{(1)}\right] \mathbb{E}\left[W_{ls}^{(1)}\right] \mathbb{E}\left[\phi\left(z'^{(1)}_r(x_i)\right) \phi\left(z'^{(1)}_s(x_j)\right)\right] \\
&= 0.
\end{aligned}$$

Therefore, we have $\operatorname{cov}\left(z_k^{(1)}(x_i), z_l^{(1)}(x_j)\right) = 0$, meaning that $z_{ik}^{(1)}$ and $z_{jl}^{(1)}$ are uncorrelated with each other *a priori*, which completes the proof of Lemma 2. However, we do not claim the validity of this lemma *a posteriori*.

B A More Detailed Proof for Theorem 1

In this section, we provide a more detailed proof for Theorem 1. Again, suppose we have data $\boldsymbol{X} = [\boldsymbol{x}_i]_{i=1}^N$ with information from 2 different sources $\boldsymbol{x} = \left\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right\}$ where $\boldsymbol{x}^{(1)} \in \mathbb{R}^{D_1}$ is high-dimensional and $\boldsymbol{x}^{(2)} \in \mathbb{R}^{D_2}$ is low-dimensional with some known relationship with the targets $\boldsymbol{Y} = [y_i]_{i=1}^N$. With ICK formulation, we have $\boldsymbol{z}^{(1)} = f_{\text{NN}}\left(\boldsymbol{x}^{(1)}\right) \in \mathbb{R}^p$ where f_{NN} is a NN-implied function with parameters $\boldsymbol{\theta}^{(1)}$ and $\boldsymbol{z}^{(2)} = g\left(\boldsymbol{x}^{(2)}\right) \in \mathbb{R}^p$

where g is a kernel-to-latent-space mapping specified by a chosen kernel function $K^{(2)}$ with parameters $\boldsymbol{\theta}^{(2)}$. In Section 3.3, it is stated that the latent representation $\boldsymbol{z}^{(1)}$ from f_{NN} will converge in distribution to a multi-output GP in the infinite width limit. In case when the NN has finite width, $\boldsymbol{z}^{(1)}$ will *approximately* follows a GP with empirical NNGP kernel function $\hat{K}^{\text{NNGP}}$

$$\boldsymbol{z}^{(1)} \sim \mathcal{GP}_{approx}\left(0, \hat{K}^{\text{NNGP}}\right). \tag{19}$$

Let $\boldsymbol{Z}^{(1)} = \left[\boldsymbol{z}_i^{(1)}\right]_{i=1}^{N}$. Based on Lemma 2, $\boldsymbol{Z}^{(1)}$ can be approximately treated as a sample from a multivariate Gaussian as shown below

$$\boldsymbol{Z}^{(1)} \sim \mathcal{N}\left(\boldsymbol{0}, \hat{\boldsymbol{K}}^{\text{NNGP}}\right), \tag{20}$$

where $\boldsymbol{Z}^{(1)} \in \mathbb{R}^{N\times p}$ and $\hat{\boldsymbol{K}}^{\text{NNGP}} \in \mathbb{R}^{Np\times Np}$ is the corresponding kernel matrix

$$\hat{\boldsymbol{K}}^{\text{NNGP}} = \begin{bmatrix} \hat{\boldsymbol{K}}_1^{\text{NNGP}} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \hat{\boldsymbol{K}}_2^{\text{NNGP}} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \hat{\boldsymbol{K}}_p^{\text{NNGP}} \end{bmatrix}, \tag{21}$$

and $\hat{\boldsymbol{K}}_k^{\text{NNGP}} = \left[\hat{K}_k^{\text{NNGP}}\left(x_i^{(1)}, x_j^{(1)}\right)\right]_{i,j=1,\ldots,N} \in \mathbb{R}^{N\times N}$ for all $k = 1, 2, ..., p$. Also, since all entries in $\boldsymbol{z}^{(1)}$ are from the same NN architecture with parameters drawn from the same distribution, as we increase the NN width, all $\hat{K}_1^{\text{NNGP}}, ..., \hat{K}_p^{\text{NNGP}}$ will converge to a deterministic limit. Therefore, in the finite-width case, it is reasonable to say that $\hat{K}_1^{\text{NNGP}} \approx \cdots \approx \hat{K}_p^{\text{NNGP}} \approx \hat{K}^{\text{NNGP}}$. Based on the marginalization property of GPs (Williams & Rasmussen, 2006), any $\boldsymbol{z}_i^{(1)} \in \mathbb{R}^p$ in $\boldsymbol{Z}^{(1)}$ should *approximately* satisfy

$$\boldsymbol{z}_i^{(1)} \sim \mathcal{N}\left(\boldsymbol{0}, \hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(1)}\right)\boldsymbol{I}_p\right), \tag{22}$$

where $\boldsymbol{I}_p$ is a $p \times p$ identity matrix. In other words, each entry in $\boldsymbol{z}_i^{(1)}$ has a univariate Gaussian distribution

$$z_{ik}^{(1)} \sim \mathcal{N}\left(0, \hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(1)}\right)\right), k = 1, 2, ..., p. \tag{23}$$

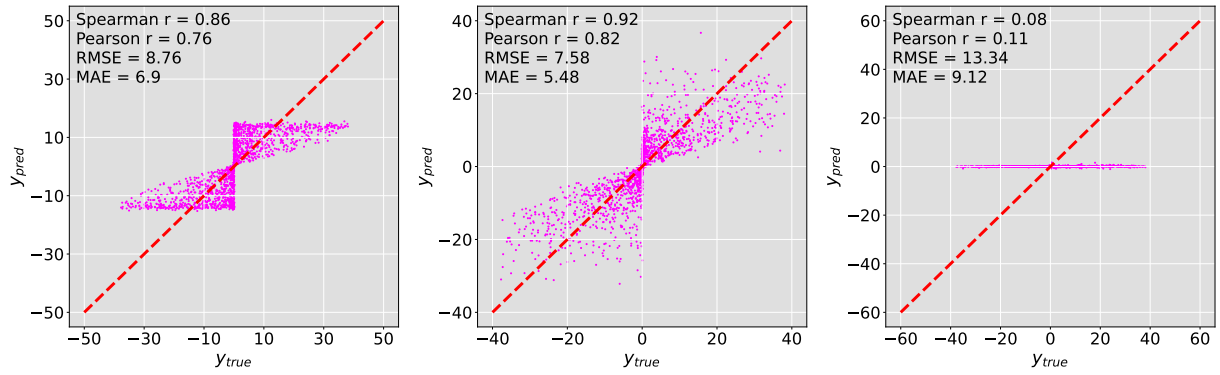
Let $\alpha_{ik} = z_{ik}^{(2)} = g\left(\boldsymbol{x}_i^{(2)}\right)_k$, since the final output is $\hat{y} = f_{\text{ICK}}\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) = {\boldsymbol{z}^{(1)}}^T \boldsymbol{z}^{(2)}$, for the i^{th} data point, $\hat{y}_i$ can be viewed as a weighted sum of p independent Gaussian random variables under Lemma 2

$$\begin{aligned} \hat{y}_i &\sim \sum_{k=1}^{p} \alpha_{ik} \mathcal{N}\left(0, \hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(1)}\right)\right) \\ &= \sum_{k=1}^{p} \mathcal{N}\left(0, \alpha_{ik}^2 \hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(1)}\right)\right) \qquad (24) \\ &= \mathcal{N}\left(0, \hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(1)}\right)\sum_{k=1}^{p} \alpha_{ik}^2\right). \qquad (25) \end{aligned}$$

Therefore, the joint distribution of the final predictions of the whole training set $\hat{\boldsymbol{Y}} = [\hat{y}_i]_{i=1}^N$ will be a multivariate Gaussian as given below:

$$\hat{\boldsymbol{Y}} \sim \mathcal{N}\left(\boldsymbol{0}, \sum_{k=1}^{p} \boldsymbol{\alpha}_k \boldsymbol{\alpha}_k^T * \hat{\boldsymbol{K}}^{\text{NNGP}}\right) = \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\alpha}\boldsymbol{\alpha}^T \odot \hat{\boldsymbol{K}}^{\text{NNGP}}\right), \tag{26}$$

where $\boldsymbol{\alpha}_k = [\alpha_{ik}]_{i=1}^N \in \mathbb{R}^N$, $\boldsymbol{\alpha} = [\alpha_{ik}]_{\substack{i=1,\ldots,N \\ k=1,\ldots,p}} \in \mathbb{R}^{N\times p}$, $\hat{\boldsymbol{K}}^{\text{NNGP}} = \left[\hat{K}^{\text{NNGP}}\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_j^{(1)}\right)\right]_{i,j=1,\ldots,N}$, and "$\odot$" represents elementwise multiplication. Therefore, if g includes the kernel-to-latent-space mapping



(a) (b) (c)

Figure C1: Scatter plots of the true values of y against the predicted values of y using our ICKr framework with information from (a) one source $\hat{y} = f_{\text{NN}}\left(x^{(1)}\right)$, (b) 2 sources $\hat{y} = f_{\text{ICK}r}\left(x^{(1)}, x^{(2)}\right)$, and (c) 3 sources $\hat{y} = f_{\text{ICK}r}\left(x^{(1)}, x^{(2)}, x^{(3)}\right)$.

$K_2\left(\boldsymbol{x}_i^{(2)}, \boldsymbol{x}_j^{(2)}\right) \approx {\boldsymbol{z}_i^{(2)}}^T \boldsymbol{z}_j^{(2)} = g\left(\boldsymbol{x}_i^{(2)}\right)^T g\left(\boldsymbol{x}_j^{(2)}\right), i, j = 1, ..., N$ as shown in Equation 5, then $\boldsymbol{K}^{(2)} = \boldsymbol{\alpha}\boldsymbol{\alpha}^T$ and we derive that

$$\hat{\boldsymbol{Y}} \sim \mathcal{N}\left(\boldsymbol{0}, \hat{\boldsymbol{K}}^{\text{NNGP}} \odot \boldsymbol{K}_2\right). \tag{27}$$

Since this derivation reasonably holds for any finite set of $\boldsymbol{X}$ and $\boldsymbol{Y}$, we say the ICK framework is *approximately* equivalent to a GP with zero mean and a multiplicative kernel between the NNGP kernel matrix $\hat{\boldsymbol{K}}^{\text{NNGP}}$ and the user-specified kernel matrix $\boldsymbol{K}_2$.

C Experimental Results of Random Fourier Features

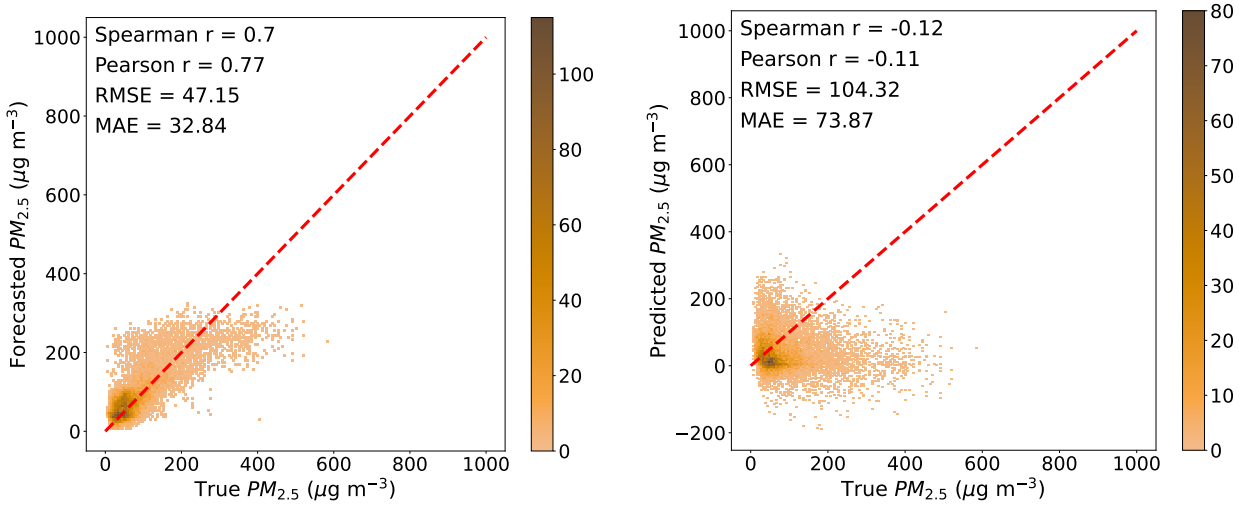
C.1 Synthetic Data

We use the same toy data set where each data point $\boldsymbol{x} = \left\{x^{(1)}, x^{(2)}, x^{(3)}\right\}$ contains 3 sources of information as described in Section 5.1.2. Also, we use the same types of kernels as those in ICKy. The only difference here is that we use RFF instead of Nyström method to transform the kernel matrix into the latent space in ICKr framework.

The results are displayed in Figure C1. It can be observed that when we add in only the side information $x^{(2)}$ along with the *exponential sine squared* kernel, both the correlation and the predictive performance are improved (though not as good as the results from ICKy as shown in Figure 9). However, after we further include $x^{(3)}$ with the *RBF* kernel, we realize that the parameters of ICKr become very hard to optimize and it fails to make valid predictions and starts to guess randomly around zero.

C.2 Remote Sensing Data

We also try ICKr on the *forecasting* task using the remote sensing data (see Section 5.2) and compare the results with those from ICKy. Each data point $\boldsymbol{x} = \{x, t\}$ contains a satellite image x as the high-dimensional information and its corresponding timestamp t as the low-dimensional information. The satellite images are processed with a two-layer CNN and the timestamps are processed with an *exponential-sine-squared* kernel with a period of $T = 365$ (days). As can be observed from Figure C2, ICKr yields much higher error compared to ICKy.



(a) (b)

Figure C2: Density plots of the true $PM_{2.5}$ concentrations against the forecasted $PM_{2.5}$ concentrations for $t \geq 500$ using (a) ICKy and (b) ICKr.

D Number of Inducing Points

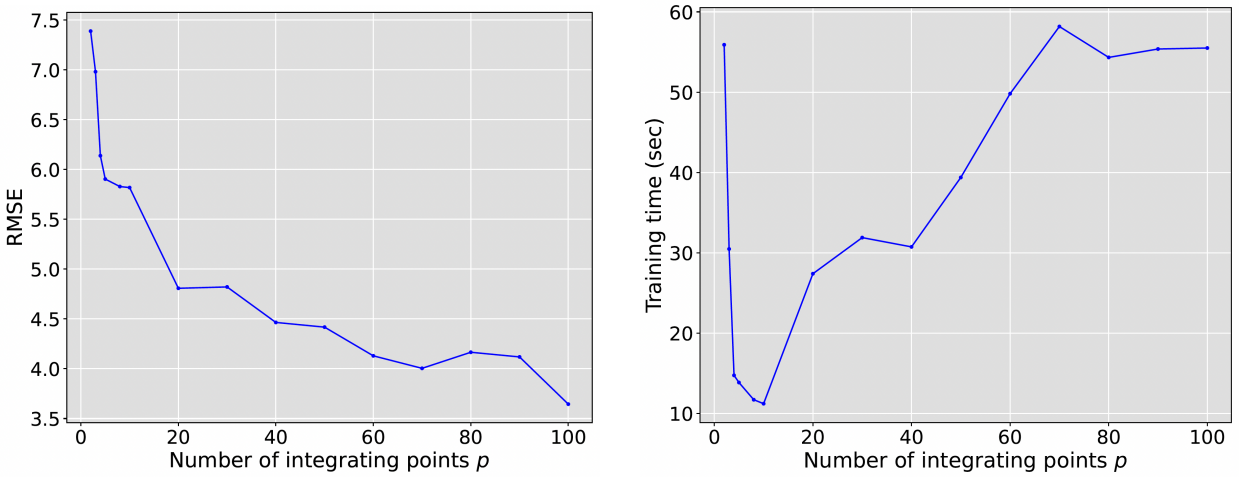
As discussed in Section 4.2.1, as we increase the number of inducing points p, we expect the approximation error between the true kernel matrix $\boldsymbol{K}$ and the approximated kernel matrix $\hat{\boldsymbol{K}}$ to decrease. Here, we empirically show how the value of p impacts our predictions. In Figure D1a, we plot the prediction error of $\hat{y} = f_{\text{ICK}y}\left(x^{(1)}, x^{(2)}, x^{(3)}\right)$ against the number of inducing points using the synthetic data generated in Appendix F. As can be observed, the prediction error drops sharply as we raise p from a small value (e.g. $p = 2$). When p is relatively large, increasing p yields smaller improvement on the predictions. Additionally, in Figure D1b, we plot the total training time against p. The total training time is dependent on how long a single iteration takes and the total number of epochs required. We note that once $p > 80$ the training time is relatively flat, which is due to the fact that the total computation in the Cholesky is less than the computation in the neural network. Interestingly, it appears that when p is very small, ICKy takes longer to converge due to the need for many more epochs. As we increase p, the training time goes down and then goes up again due to the computational complexity, i.e. $\mathcal{O}(p^3)$, of the Cholesky decomposition. Based on these observations, we are not concerned about the computational complexity for reasonable values of p.

E Applying Sample-then-optimize Procedure to ICK

As elaborated in Section 4.3.1, a deep ensemble with proper initialization scheme will have a GP posterior interpretation in the infinite width limit when trained by a squared-error loss (He et al., 2020), which is an example of the "sample-then-optimize" procedure of Matthews et al. (2017) Algorithm 3 is an alternative to replace f_{NN} with such a deep ensemble $F = \{f_{n_e}\}_{n_e=1}^{N_e}$ where the dimension of the final readout layer of each f_{n_e} is p. In this case, each baselearner f_{n_e} can be viewed as an i.i.d. sample from a multi-output GP in the infinite width limit. In the finite-width case, this relationship becomes approximate:

$$f_{n_e} \xrightarrow{d} \mathcal{GP}_{approx}\left(0, K_F\right), \tag{28}$$

As stated by Lee et al. (2019), if all the parameters in f_{n_e} are randomly drawn from a Gaussian distribution and are all fixed except the last layer, then after training F on a squared-error loss, we will have $K_F \to K^{\text{NNGP}}$ where K^{NNGP} is the NNGP kernel. He et al. (2020) also proposed to add a random and untrainable function $\delta(\cdot)$ to the output of f_{n_e} so we have $K_F \to K^{\text{NTK}}$ where K^{NTK} is the limiting NTK.



(a) (b)

Figure D1: Plots of (a) prediction error and (b) training time of $\hat{y} = f_{\text{ICK}y}\left(x^{(1)}, x^{(2)}, x^{(3)}\right)$ against the number of inducing points p

With ICK formulation, by following the proof given in Appendix B, we can derive that, for each baselearner in the ICK ensemble, the joint distribution of the final predictions of the whole training set $\hat{\boldsymbol{Y}}_{n_e}$ will again be a multivariate Gaussian as shown below if Lemma 2 holds *a posteriori*:

$$\hat{\boldsymbol{Y}}_{n_e} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{K}_F \odot \boldsymbol{K}^{(2)}\right), \tag{29}$$

where $(\boldsymbol{K}_F)_{ij} = K_F\left(\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_j^{(1)}\right)$. Since Equation 29 holds for any finite input data set, we can conclude that each baselearner f_s in the ICK ensemble $F_{\text{ICK}y} = \{f_s\}_{s=1}^{N_e}$ can be *approximately* viewed as an i.i.d. sample from a single-output GP

$$f_s \sim \mathcal{GP}_{approx}\left(0, K_F K^{(2)}\right) = \mathcal{GP}_{approx}\left(0, K^{\text{comp}}\right). \tag{30}$$

Building an ensemble $F_{\text{ICK}y} = \{f_s\}_{s=1}^{N_e}$ is thus equivalent to performing a Monte Carlo approximation to a GP predictive posterior distribution whose mean and covariance matrix for a test data set $\boldsymbol{X}^* = [\boldsymbol{x}_i^*]_{i=1}^{N^*}$ are

$$f_s(\boldsymbol{X}^*|\boldsymbol{X}) \sim \mathcal{N}\left(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*\right), \tag{31}$$

$$\boldsymbol{\mu}^* = \boldsymbol{K}_{X^*X}^{\text{comp}} \left(\boldsymbol{K}_{XX}^{\text{comp}}\right)^{-1} \boldsymbol{Y}, \tag{32}$$

$$\boldsymbol{\Sigma}^* = \boldsymbol{K}_{X^*X^*}^{\text{comp}} - \boldsymbol{K}_{X^*X}^{\text{comp}} \left(\boldsymbol{K}_{XX}^{\text{comp}}\right)^{-1} \boldsymbol{K}_{XX^*}^{\text{comp}}, \tag{33}$$

where $\left(\boldsymbol{K}_{X^*X}^{\text{comp}}\right)_{ij} = K^{\text{comp}}\left(\boldsymbol{x}_i^*, \boldsymbol{x}_j\right)$, $\left(\boldsymbol{K}_{XX}^{\text{comp}}\right)_{ij} = K^{\text{comp}}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)$, $\left(\boldsymbol{K}_{X^*X^*}^{\text{comp}}\right)_{ij} = K^{\text{comp}}\left(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*\right)$, and $\boldsymbol{K}_{XX^*}^{\text{comp}} = \left(\boldsymbol{K}_{X^*X}^{\text{comp}}\right)^T$. In other words, we can approximate the GP predictive posterior using the predictive mean and variance generated by Algorithm 3 as follows:

$$\hat{\boldsymbol{\mu}}^* = \frac{1}{N_e} \sum_{n=1}^{N_e} f_s(\boldsymbol{X}^*|\boldsymbol{X}) \approx \mathbb{E}\left[f_s(\boldsymbol{X}^*|\boldsymbol{X})\right] = \boldsymbol{\mu}^*, \tag{34}$$

$$\begin{aligned} \hat{\boldsymbol{\sigma}}^{*2} &= \frac{1}{N_e} \sum_{n=1}^{N_e} \left[f_s(\boldsymbol{X}^*|\boldsymbol{X}) - \hat{\boldsymbol{\mu}}^*\right]^2 \approx \mathbb{V}\left[f_s(\boldsymbol{X}^*|\boldsymbol{X})\right] \\ &= \text{diag}\left(\boldsymbol{\Sigma}^*\right). \end{aligned} \tag{35}$$

Table 5: Model architecture and training details for remote sensing data experiment in Section 5.2

	Backbone architecture details	Output FC layers dimension	Optimizer
CNN-RF	# Conv blocks = 2, # Channels = 16, Kernel size = 3, Stride = 1	1000 + d_{RT}, 512, 512, 1	Adam $\beta_1 = 0.9$ $\beta_2 = 0.999$
ViT-RF	# Transformer blocks = 2, # Attention heads = 8, Dropout ratio = 0.1	1000 + d_{RT}, 512, 512, 1	Adam $\beta_1 = 0.9$ $\beta_2 = 0.999$
DeepViT-RF	# Transformer blocks = 2, # Attention heads = 8, Dropout ratio = 0.1	1000 + d_{RT}, 512, 512, 1	Adam $\beta_1 = 0.9$ $\beta_2 = 0.999$
MAE-ViT-RF	# Transformer blocks = 2, # Attention heads = 8, Dropout ratio = 0.1, Masking ratio = 0.75	1000 + d_{RT}, 512, 512, 1	Adam $\beta_1 = 0.9$ $\beta_2 = 0.999$
CNN-ICKy	# Conv blocks = 2, # Channels = 16, Kernel size = 3, Stride = 1	1000, 512, p	SGD momentum = 0.9
ViT-ICKy	# Transformer blocks = 2, # Attention heads = 8, Dropout ratio = 0.1	1000, 512, p	SGD momentum = 0.9
DeepViT-ICKy	# Transformer blocks = 2, # Attention heads = 8, Dropout ratio = 0.1	1000, 512, p	SGD momentum = 0.9

F Experimental Details

F.1 Synthetic Data

We use the GPytorch package to generate the synthetic data. The data set is first randomly shuffled and then divided into train and test set with a 50:50 ratio. Before feeding $x^{(1)}$ into MLP, we first map $x^{(1)}$ into higher dimension using an unsupervised algorithm called Totally Random Trees Embedding. All the MLP structures in this experiment (including those in MLP-RF and ICKy) contain one single fully connected (FC) layer of width 1000, which serves as a simple benchmark since a one-hidden-layer MLP can only capture linear relationship between the input and output. For model training, we optimize a Mean Squared Error (MSE) objective using Adam optimizer with a weight decay of 0.1.

F.2 Remote Sensing Data

We collect remote sensing data from 51 air quality monitoring (AQM) stations located in the National Capital Territory (NCT) of Delhi and its satellite cities over the period from January 1, 2018 to June 30, 2020 (see Appendix G for notes on data availability). The timestamps are converted into numerical values t (where the day 2018-01-01 corresponds to $t = 0$) before feeding them into the models. We split the train, validation, and test data set based on t. Specifically, we use all the data points with $t < 365$ for training, $365 \leq t < 500$ for validation, and $t \geq 500$ for testing.

The model architecture and training details are listed in Table 5. Here p denotes the length of latent representations $\boldsymbol{z}$ as discussed in Section 4 and d_{RT} denotes the transformed dimension of timestamp t using the Random Trees Embedding as mentioned in Section F.1. Note that we use stochastic gradient descent (SGD) optimizer with a momentum of 0.9 for ICKy as we realize that SGD helps ICKy find a local minimum on the objective more efficiently. We use MSE objective for ICKy and all benchmark models in this experiment.

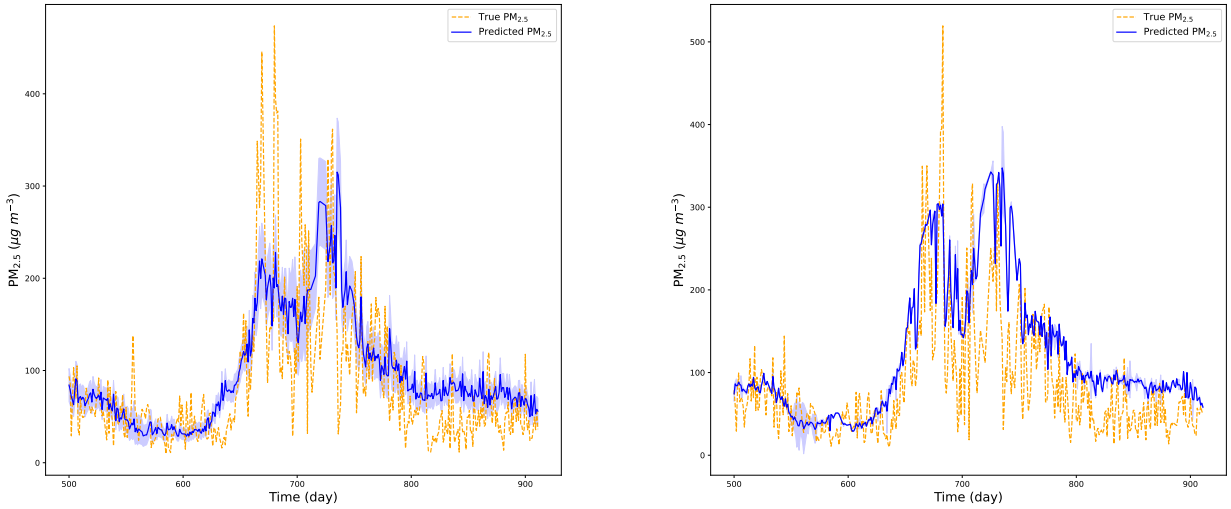


Figure H1: Time series visualization of predictive mean and uncertainty of $PM_{2.5}$ in Section 5.2 (remote sensing experiment) for (a) CNN-ICKy (1.92M parameters) and (b) DeepViT-ICKy (21.80M parameters).

F.3 Other Regression Datasets

For the worker productivity, we separate out the temporal information (i.e. date and time) and use it as the low-dimensional information. The rest of the features are then concatenated together to serve as the high-dimensional information. The MLPs (including the MLP part in ICKy) in this experiment share the same structure as the one used in Al Imran et al. (2019), which consist of 3 hidden layers of width 128, 32, and 32, respectively. For plain MLP, cyclic MLP, and ICKy, we use the mean absolute error (MAE) objective to put less weight on the outliers and thus enhance the model performance. All these objectives are optimized by an Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The power consumption data is preprocessed in a similar way to the worker productivity data, where we separate out the data and time features, transform them into a one-dimensional time index, and use it as the low-dimensional information. The rest of the features are concatenated together to serve as the high-dimensional information. The performance of the GP benchmarks are directly obtained from Wang et al. (2019).

G Accessibility and Restrictions of the Data

All experiments are conducted on a computer cluster equipped with a GeForce RTX 2080 Ti GPU. The synthetic data in Section 5.1 are generated using the GPyTorch package. The remote sensing data in Section 5.2 is downloaded using PlanetScope API whose content is protected by copyright and/or other intellectual property laws. To access the data on PlanetScope, the purchase of an end-user license is required. When this manuscript is accepted, we will provide the codes we used to acquire the data. The UCI machine learning repository data we use in Section 5.3 has an open access license, meaning that the data is freely available online.

H Time Series Visualization for Remote Sensing Experiment

To explore the problem of posterior uncertainty calibration for ViT variants of ICKy (i.e. extremely large MSLL as shown in Table 2 in Section 5.2), we visualize the results as time series by first grouping the predictions by the timestamp t, taking the *minimum* of the predictive variance, and plotting them along with the corresponding true values and the predictive mean as shown in Figure H1. The shaded region represents

the confidence interval $[\mu-2\sigma, \mu+2\sigma]$ where μ and σ are the predictive mean and standard deviation of $PM_{2.5}$, respectively. We realize that CNN-ICKy ensemble tends to yield much higher variance than DeepViT-ICKy ensemble when the predictive mean deviates from the true values. A plausible explanation is that the DeepViT structure contains much more parameters than the CNN structure (21.80M vs 1.92M) used in our remote sensing experiment, which makes DeepViT-ICKy overparameterized (The DeepViT architecture is set to be consistent with the CNN architecture as shown in Table 5). To alleviate this problem, we try reducing the number of transformer blocks in ViT and DeepViT and we do observe a significant drop in MSLL.