*Proceedings of the 7th International Conference on Statistics: Theory and Applications (ICSTA'25)*
*Paris, France - August, 2025*
*Paper No. 156*
*DOI: 10.11159/icsta25.156*

# Performance Comparison of Statistical Emulators for Parameter Estimation in Complex Systems

**Hongjin Ren[1], Hao Gao[1], Mu Niu[1], Vinny Davies[1]*, Benn Macdonald[1]***
[1]School of Mathematics and Statistics, University of Glasgow,
Glasgow G12 8SQ, UK
Hongjin.Ren@glasgow.ac.uk; Hao.Gao@glasgow.ac.uk; Mu.Niu@glasgow.ac.uk;
Vinny.Davies@glasgow.ac.uk; Benn.Macdonald@glasgow.ac.uk, *Indicates joint last authorship

***Abstract*** **-** Mathematical models allow us to simulate complex systems, whose behaviour depends significantly on their underlying parameters. However, direct parameter inference of these systems typically involves repeatedly computing numerical solutions. Consequently, reducing the computational burden associated with parameter estimation is crucial for enhancing the practicality of these models. Statistical emulators present a promising solution to this issue, as they approximate mathematical models and substantially reduce computational demands. Despite the potential benefits of emulators, selecting an appropriate emulation strategy remains challenging, primarily due to issues such as high dimensionality, sparse data and correlated outputs. In this study, we assess the effectiveness of various emulation strategies for parameter inference under different data scenarios. Our evaluation encompasses statistical models based on standard Gaussian Processes, variational Gaussian Processes, deep kernel learning, deep Gaussian Processes, and deep neural networks. We construct several simulated data sets and analyse the parameter estimation accuracy of these models under different conditions, including output independence, different input-output dimensionality ratios and data sparsity. Our results demonstrate that the multi-output Gaussian Process consistently achieves superior parameter estimation accuracy compared to other Gaussian Process variants and deep neural networks, particularly in high-dimensional complex systems with multiple dependent outputs, and maintains greater stability in scenarios with sparse data. These findings give insight into emulation strategies applicable to parameter estimation of high-dimensional complex systems and provide a foundation for the future development of real-time parameter estimation in practical applications.

***Keywords*****:** Gaussian Processes; Statistical Emulation; Neural Networks; Parameter Estimation

## 1. Introduction

In recent years, advances in mathematical modelling have enabled researchers from various fields to develop increasingly sophisticated models for analysing complex systems. Examples of these developments include biomechanical heart models governed by non-linear constitutive laws [1] and geological models describing landscape evolution [2]. In such studies, the system equations combined with their numerical implementations are commonly called simulators [3]. Typically, the outputs of the simulator are solutions to these equations, representing observations of the studied system. Understanding and predicting the behaviour of the system depends heavily upon the parameters within these equations, which generally carry explicit practical interpretations. For example, in biomechanical heart modelling, parameters are frequently associated with the physiological characteristics of myocardial tissue, and their rapid and accurate estimation facilitates the timely detection of cardiovascular diseases such as myocardial infarction [4]. Consequently, real-time estimation of model parameters has become an important research priority.

Parameter estimation of these systems often involves comparing measurable data with simulator outputs, employing an optimisation algorithm to identify parameter values that minimise the discrepancy between the two. In mathematical sciences, this process is referred to as inverse problem solving. However, complex mathematical simulators are typically unsuitable for direct application in real-time decision-making contexts. This limitation is primarily because most differential equations do not have closed-form analytical solutions, necessitating repeated numerical computations for each forward simulation. The computational cost per solution depends on the complexity of the model and can take anywhere from a few minutes to several hours per run. Estimating even a single parameter often demands hundreds or thousands of repeated simulations of the mathematical model, and the resulting high computational cost makes it challenging to implement real-time estimation.

To address these computational challenges, employing low-cost statistical emulators as approximations of computationally expensive simulators has become a widely adopted strategy [5]. These emulators can be pre-trained using previously generated simulation data, enabling their predicted outcomes to replace direct simulator computations. Once trained, emulators can be evaluated rapidly, significantly accelerating the estimation process. Consequently, researchers are able to perform parameter estimation in real-time applications with significantly reduced computational demands. A variety of statistical regression and deep learning approaches have been explored to develop such emulators. Gaussian Processes (GPs), in particular, have been extensively utilised due to their flexibility as non-parametric models [6] In studies involving inverse estimation of material parameters for left ventricular biomechanics, both GP-based models and neural network models have been shown to facilitate accurate parameter estimation [7].

While statistical emulators significantly enhance the efficiency of parameter estimation in real-time applications, their effectiveness heavily depends on selecting an appropriate emulator structure. When analysing increasingly complex systems, standard emulators often fail to adequately represent intricate relationships inherent in high-dimensional input-output mappings, meaning that it is necessity to consider more advanced models capable of addressing these limitations. Although deep learning models can address such complexity, they often have the drawback of being difficult to tune for suitable model structures. Standard GPs often struggle to achieve a balance between training efficiency and model flexibility. To overcome these limitations, several GP-based extensions have been proposed, each offering distinct advantages. For example, Deep Kernel GPs integrate the non-parametric flexibility of GPs with the representational capabilities of deep learning, enhancing the ability of the model to represent complexity without requiring extensive adjustment of kernel structures [8]. Variational GPs combine variational inference with evidence lower bound optimisation, leading to improved predictive accuracy alongside reduced computational complexity [9]. In addition, Deep GPs extend the application range of GPs, enabling it to effectively handle tens of billions of data points and excel at regression problems with large data sets [10].

This study evaluates a variety of emulation strategies for solving inverse problems for complex systems, including different types of multi-output GPs, combinations of single-output GPs, and deep learning models. Previous studies on parameter estimation for biomechanical systems have shown that, among various statistical models, GP-based approaches have demonstrated outstanding performance in addressing inverse problems [7]. However, most of these emulation strategies have paid relatively limited attention to the correlations among output variables and capturing complex interdependencies. In our study, we create simulation data with complex multi-dimensionality by controlling the correlation coefficients between output variables, the ratio of input to output dimensions, and data sparsity. These datasets present more complex interdependencies between dimensions, which will allow us to assess the accuracy of parameter estimation under different strategies and explore their applicability.

## 2. Methodology

### 2.1. Emulator

Emulators, $\mathcal{M}$, often referred to as surrogate models, can be used to accelerate computations by approximating the output of a complex mathematical model, $M$. Typically, emulators based on statistical methods are trained using data generated beforehand by numerically solving the mathematical model. The time taken to produce these solutions is not included in the computational cost of the real-time solutions to the inverse problem, as once trained, repeated solving of the mathematical model can be avoided by using the emulator predictions. Given that solving the original model is computationally intensive, repeatedly solving it as required by parameter inference tasks would quickly exceed practical time constraints, however, the simulator can efficiently be run hundreds or even thousands of times. This capability makes it possible to implement fast parameter estimation in high-dimensional complex problems.

### 2.2. Design of Simulations

When building an emulator, $\mathcal{M}$, it is essential to collect a sufficiently large and systematically organized dataset that includes both input parameters and their corresponding output variables for training. Ideally, the input parameters should be densely sampled across the parameter space to ensure that the emulator accurately captures the system's behaviour throughout the entire domain. In practical applications, data are typically generated by numerically solving the mathematical model, $M$, which simulates these systems. Based on this process, an appropriate dataset $\boldsymbol{D}$ should be designed to train the emulator. The dataset $\boldsymbol{D}$ consists of input parameters $\boldsymbol{\theta}$ and output $\boldsymbol{y}$, $\boldsymbol{D} = \{(\boldsymbol{\theta}_i, \boldsymbol{y}_i)\}$, for $i = 1,2,\dots,N$, where $N$ denotes the size of the dataset.

### 2.2.1 Input Generation

A straightforward approach to obtaining parameter combinations is to sample uniformly across the parameter space. However, uniform sampling may result in multiple samples adopting near-identical parameter values in particular dimensions, which can reduce the informative content of the dataset, especially if the corresponding output is relatively insensitive to those parameters. Sobol sequences and Latin hypercube sampling can effectively alleviate this issue through deterministic rules and hierarchical strategies, respectively. In high-dimensional contexts, Sobol sequences are particularly advantageous as they mitigate correlation in the distribution of sampled points, thereby ensuring stability [11]. This study employs Sobol sequences to generate the input parameters. Each set of generated input parameters $\boldsymbol{\theta}_i$ must satisfy $\boldsymbol{\theta}_i \in [a,b]^d, i = 1,2,\ldots,N$, where $d$ is the dimensionality of the input space and $[a,b]$ is the range of parameters.

### 2.2.2 Output Generation

In most studies, the outputs of the dataset are typically obtained by designing the parameter combination and then solving the simulator. In this work, simulated data can be used to better demonstrate different scenarios, allowing for more comprehensive testing. Specifically, we generated multiple datasets using different multivariate Gaussian distributions which can model the correlation between outputs. The mean of these multivariate Gaussian distributions is set to 0, and the covariance matrix is $\boldsymbol{K}$. After obtaining the input set $\boldsymbol{\Theta}$, the covariance matrix can be generated using a radial basis function (RBF) kernel $k(\boldsymbol{\theta},\boldsymbol{\theta}')$, which is given by the following formula:

$$k(\boldsymbol{\theta},\boldsymbol{\theta}') = \sigma^2 \, exp\left( -\frac{1}{2}\left\| \frac{\boldsymbol{\theta} - \boldsymbol{\theta}'}{\boldsymbol{l}} \right\|^2 \right), \tag{1}$$

where $\theta$ and $\theta'$ represent different inputs, $\boldsymbol{l}$ represents the length scale for each input dimension, and $\sigma^2$ is the variance parameter. $\boldsymbol{l}$ and $\sigma^2$ are randomly selected for each different covariance matrix. We can sample the intermediate variable $f_{inner}$ from this multivariate Gaussian distribution, which is expressed as follows:

$$\boldsymbol{f}_{\boldsymbol{inner}} \sim \boldsymbol{\mathcal{MVN}}(0,\boldsymbol{K}). \tag{2}$$

The sampling process is carried out in $m$ independent distributions and then combined into a single matrix, $\boldsymbol{F}_{\boldsymbol{inner}} \in \mathbb{R}^{N\times m}$, the final output $\boldsymbol{Y} \in \mathbb{R}^{N\times m}$ is obtained through the weight matrix $\boldsymbol{W}$

$$\boldsymbol{Y} = \boldsymbol{F}_{\boldsymbol{inner}}\boldsymbol{W}, \tag{3}$$

where the design strategy of the mapping matrix $\boldsymbol{W} \in \mathbb{R}^{m\times m}$ determines the relationship between output variables $Y$. We set $\boldsymbol{W}$ as a diagonal matrix to give independent outputs, with its diagonal elements randomly sampled from $\mathcal{U}(0.5, 1)$. To simulate correlations among outputs, we introduce off-diagonal elements to the above diagonal matrix, where these off-diagonal elements are sampled from $\mathcal{N}(0, 1)$.

## 2.3. Parameter Estimation

Building on the dataset described in the preceding sections, the emulator is trained to approximate the outputs of the mathematical model. Once trained, the emulator can be employed for parameter estimation by minimising the discrepancy between the test data $y_{test}$ and the emulator prediction $\mathcal{M}(\theta)$. The estimate of the parameter vector $\hat{\theta}$ is obtained by solving the following optimisation problem:

$$\hat{\theta} = \arg\min_{\theta} l\left(\theta | \mathcal{M}, \boldsymbol{y}_{test}\right), \tag{4}$$

where $l(\theta | \mathcal{M}, \boldsymbol{y}_{test})$ indicates the loss function between the observed value and the predicted value, which is measured using the Euclidean distance. The Adam global optimisation algorithm is used to minimise the loss function. Compared to the traditional stochastic gradient descent method, the Adam algorithm can converge more quickly using adaptive estimates of the first and second moments [12]. At the same time, we use multiple starting points to explore the parameter space, thereby reducing the possibility of getting stuck in a local minima.

## 3. Emulation Methods

Emulator strategies are often divided into local and global models. The local models aim to accurately fit the variations around the observation point. This approach significantly reduces computational complexity by fitting a separate GP model using a local training dataset with the nearest neighbours selected for $y_{\text{test}}$. In contrast, global models rely on training the emulator on the full dataset, with approximation techniques often used to reduce the computational cost associated with this approach.

### 3.1. Local Models

#### 3.1.1 Standard GPs

GP regression is a powerful non-parametric Bayesian technique. Priors are placed directly on the function space, providing a flexible framework that naturally allows predictions to incorporate a measure of uncertainty [6]. However, the computational complexity of the standard GP model in a dataset of size $n$ is approximately $O(n^3)$, which increases cubically with the data. This high complexity can be avoided by matching the model to the local dataset, as the complexity of the model will be kept within the size of the local dataset.

The simplest single-output local GP (hereafter L.GP) treats each output variable independently. When applied to a multi-output dataset, although this approach reduces the computational complexity, it ignores the potential correlations between the output variables and requires a separate model to be established for each output variable. Formally, a single-output GP for a stochastic process $f(\theta)$ can be written as:

$$f(\theta) \sim \mathrm{GP}(\mathrm{m}(\theta),\mathrm{k}(\theta,\theta')), \tag{5}$$

where $f(\theta)$ represents the prediction of $y$, $\theta$ and $\theta'$ represent different input vectors, $m(\theta)$ is the mean function, and $k(\theta,\theta')$ is the covariance function. In this work, a constant is used as the mean function and an automatic relevance determination (ARD) RBF is used as the covariance function. This single-output model independently models each output and cannot take advantage of the correlation between multiple outputs.

An extension that also applies to local datasets is the multiple output GP (L.MGP), which can capture the relationship between multiple tasks by incorporating an additional covariance structure. Specifically, the covariance takes the form of:

$$k([\theta,\mathrm{i}],[\theta',\mathrm{j}]) = \mathrm{k}(\theta,\theta') \times \mathrm{k}_{\text{tasks}}(\mathrm{i},\mathrm{j}), \tag{6}$$

where $i$ and $j$ denote different outputs and $k_{\text{tasks}}(i,j)$ denotes the correlation between outputs. By exploiting this covariance matrix, the L.MGP framework can effectively model the correlation between outputs.

#### 3.1.2 Deep Kernel Learning (DKL)

Deep Kernel Learning (DKL) introduces a deep kernel by combining a deep neural network (DNN) with a GP. This approach retains the GP's ability to quantify prediction uncertainty while enhancing its capacity to model complex data patterns through deep feature extraction. Experiments have shown that it outperforms the traditional GP and stand-alone DNN in small samples and high-dimensional outputs [8]. The covariance matrix in this structure is expressed as:

$$k(\theta,\theta'|\boldsymbol{\ell})\rightarrow \mathrm{k}(\mathrm{g}(\theta,\mathrm{w}),\mathrm{g}(\theta',\mathrm{w})|\boldsymbol{\ell},\mathrm{w}), \tag{7}$$

where $\boldsymbol{\ell}$ is the hyperparameter of the base kernel and $g(\theta,w)$ is a nonlinear mapping given by the neural network. This kernel function structure can be combined with single-output and multi-output models, and is represented by L.DKGP and L.DKMGP respectively.

### 3.2. Global Models

#### 3.2.1 Variational Gaussian Processes (VGPs)

Variational GPs (VGPs) extend the concept of sparse approximation by using a variational inference framework to approximate the posterior distribution of the GP[9]. This approach optimises the variational parameters as well as the location of the induced points and the model hyperparameters, providing a more flexible approximation that can balance computational efficiency with modelling accuracy. The multi-output VGP is implemented by the Linear Model of Coregionalization, which assumes that each output dimension is a linear combination of several latent functions:

$$f_{task}(\theta) = \sum_{i=1}^{Q} a_i g_i(\theta), \tag{8}$$

where $a_i$ are learnable parameters, optimised along with the rest of the hyperparameters of the model. $Q$ is the number of potential functions used to compose each output and $g_i$ is a set of variational GP models.

### 3.2.2 Deep Gaussian Processes (DGPs)

Deep GPs (DGPs) are a hierarchical extension of variational GPs, designed to address the limitations of single-layer GPs, which can model only a single non-linear relationship. DGPs retain the advantages of GPs, while offering greater expressiveness, similar to how deep networks surpass generalized linear models [10]. Unlike DKL models with highly parameterised kernels, DGPs require fewer hyperparameters, reducing the risk of overfitting. In this study, we construct a two-layer DGP model with matching input and intermediate dimensions. Extending to higher layers would exponentially increase training time and result in the covariance matrix to become non-positive definite, making the approach computationally impractical.

### 3.2.3 Deep Neural Networks (DNNs)

Deep Neural Networks (DNNs) are highly flexible, parameterised learning models. They can be trained relatively quickly, even on full datasets, and high prediction accuracy is often achievable by tuning the network architecture. In our approach, we identified the network structure that yielded the best predictive performance using a sampled full dataset. A validation set was then derived from this sample, and a forward neural network was constructed consisting of five fully connected layers. ReLU activation functions and Batch Normalisation were applied between each layer.

## 4. Simulation Data

Based on the methods described in Section 2.2, we designed three experiments to evaluate the parameter estimation performance of different emulation methods in different scenarios. Each experiment investigated the impact of output correlation, dimensionality effects and data requirements on the estimation results. All experiments kept the same test set size of 54 and generated 50 independent datasets to ensure robustness and reproducibility of the results.

### 4.1 Analysis of Output Correlation

To investigate how the correlation between output variables affects the accuracy of parameter estimation, we constructed two datasets, one with highly correlated outputs and the other with independent output variables. In this experiment, we set the input dimension to $d = 3$, the output dimension to $p = 6$, and the training set size to $n = 2024$. This setup limits the training time and parameter estimation process to a reasonable time to demonstrate the advantages and disadvantages of different emulation methods.

### 4.2. Output Dimensional Transformation

Considering that a higher output dimension will increase the complexity of emulator construction, while a lower output dimension may not be sufficient to solve the inverse problem, we will fix the input dimension and gradually increase the output dimension to generate different datasets. In this experiment, the input dimension is fixed at $d = 5$ dimensions, the

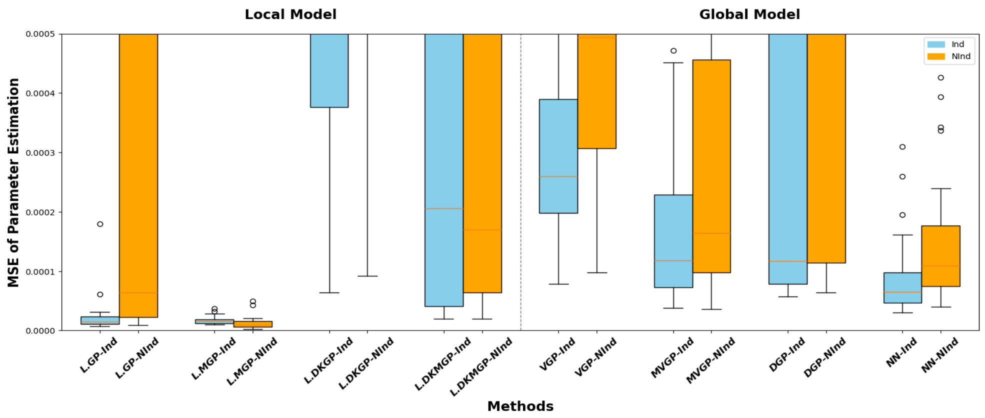


Fig. 1: MSE of parameter estimation using different emulator methods across datasets where outputs are either independent or correlated. Different colors indicate whether the outputs are independent: Ind means the outputs are independent, whereas NInd indicates that the outputs are correlated. Method names with the prefix L. represent local methods, while those without a prefix denote global methods. The vertical dashed line separates local method results from global method results.

output dimension $p$ is gradually increased from 4 to 8. Since the increase in the number of input dimensions requires a suitable increase in the size of the training set to better cover the parameter space, we set the training set to $n = 4048$. This setting aims to explore the feasibility of parameter estimation under different input and output dimensions and to analyse the best possible ratio between them.

### 4.3. Variation in Data Requirements

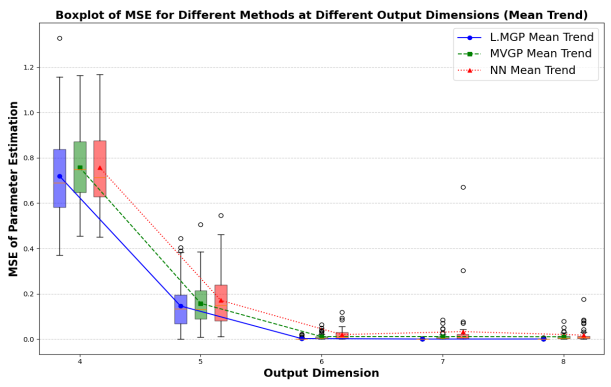
The balance between training data coverage and model complexity is also a factor to consider. We fix the input and output dimensions $(d = 5, p = 6)$ and gradually reduce the size of the training set $n$ from 4048 to 1024 to evaluate the impact of data volume on parameter estimation performance. Through this experiment, we can quantify the parameter space coverage requirements of different emulation strategies and provide practical guidance for experimental design in high-dimensional environments.

## 5. Results

We generated three distinct dataset scenarios using the data generation methods described in Section 4. The approaches outlined in Section 3 were then applied, and the corresponding parameters were estimated according to the procedures detailed in Section 2.3. To evaluate the robustness of these methods, we repeated the entire process 50 times using newly generated datasets. Finally, we computed the mean squared error (MSE) between the estimated parameter values and the parameter values used to generate the data, for each repetition.

First, we use eight methods, four local and four global models, to assess the impact of output relevance on parameter estimation tasks. The local models include single-output (L.GP) and multi-output (L.MGP) versions of the standard GP and single-output (L.DKGP) and multi-output (L.DKMGP) models that incorporate Deep Kernel Learning. Meanwhile, the global models comprise single-output (VGP) and multi-output (MVGP) Variational GP, Deep Gaussian Process (DGP), and a baseline Deep Neural Network (NN). The results are presented in Fig. 1, which shows that the performance of all global models is inferior to that of L.MGP. L.GP performs well on datasets with mutually independent outputs, but results in poorer parameter estimation when the outputs are not independent. Moreover, the DKL-based methods do not exhibit notably superior capabilities.

Next, from the output-relevant dataset illustrated in Fig. 1, we select the three best-performing models for further analysis, namely L.MGP (representing local models), MVGP (representing global models), and a DNN (as the baseline). We then fix the input dimensionality (at $d = 5$) and gradually increase the output dimensionality. As shown in Fig. 2, enlarging the output dimension alleviates the challenges of inverse problems, resulting in more accurate parameter inference. Notably,



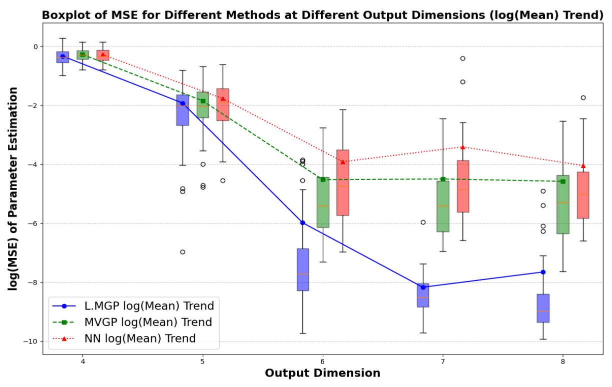
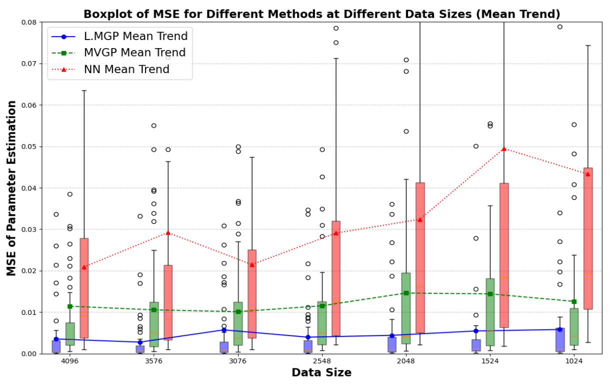


Fig. 3: MSE of parameter estimation using different methods across datasets with varying output dimensions, showing both the original (left) and log-transformed (right) results.



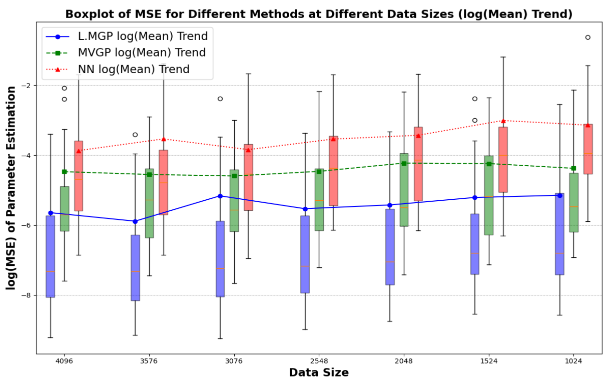


Fig. 2: MSE of parameter estimation using different methods across datasets of varying sizes, with both the original (left) and log-transformed (right) results

once the output dimension surpasses a certain threshold, further increases do not lead to additional performance gains for any of the three models. Table 1 summarises the results of carrying out t-tests on the MSE values for these methods under varying output dimensions. The results indicate that when the output dimensionality is less than or equal to the input dimensionality, there is no significant average performance difference among the three models. Only when the output dimensionality exceeds the input dimensionality do GP-based emulation strategies outperform the neural network model.

Finally, as described in Section 4.3, we varied the training dataset size while keeping the input and output dimensionalities fixed. The results in Fig. 3 indicate that reducing the training dataset size does not cause significant performance degradation across models. The MSE distribution and mean fluctuations of GP-based models are smaller compared to those of the DNN model.

## 6. Conclusion

This study evaluates and compares statistical emulation strategies for estimating multiple parameters in complex systems, focusing on their performance under different conditions such as output correlation, input-output dimensionality

Table 1: P-values from the t-test conducted on the MSE of different methods, evaluated across datasets with varying output dimensionalities. Statistically significant differences in average MSE, at the 5 % significance threshold, are indicated in bold.

| Models \ Output Dimension | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|
| L.MGP vs NN | 0.342 | 0.280 | **<0.001** | **0.0271** | **<0.001** |
| MVGP vs NN | 0.976 | 0.566 | **0.0300** | 0.139 | 0.137 |
| L.MGP vs MVGP | 0.308 | 0.575 | **<0.001** | **<0.001** | **<0.001** |

ratio, and the amount of training data. Our analysis includes standard Gaussian processes, variational Gaussian processes, deep Gaussian processes, deep kernel learning, and deep neural network models.

Our results show that multi-output Gaussian process models, in particular local multi-output Gaussian process (L.MGP), consistently achieve superior accuracy in parameter estimation when applied to datasets with correlated outputs. Single-output Gaussian process models excel with independent outputs, but are limited when dealing with correlated data due to their inability to capture cross-dimensional dependencies. It is worth noting that deep kernel learning did not significantly improve the performance of the emulator compared to the standard Gaussian process method in the scenarios we tested.

Further investigation of the dimensional effect shows that the accuracy of parameter estimation generally improves as the output dimension is increased relative to the input dimension. However, beyond a certain threshold, further increases in the output dimension no longer significantly improve performance.

In addition, we assess how variations in training dataset size affect emulator performance. The results demonstrate that Gaussian-process-based approaches remain robust, maintaining stable estimation accuracy even with reduced dataset sizes. In contrast, neural network models exhibit larger performance fluctuations, indicating greater sensitivity to data size.

Overall, the L.MGP model, the MVGP model, and the neural network model each offer promise for parameter inference in complex systems. In some instances, GP-based methods match or surpass the accuracy of neural networks, while requiring fewer structural adjustments. When local and global GP models achieve similar accuracy, the global approach is preferred, since local models must be retrained for each test point, thereby increasing computational overhead. In future work, we plan to incorporate Bayesian inference to more rigorously quantify and interpret parameter uncertainty, providing a more robust framework for parameter estimation in complex systems.

## References

[1] G. A. Holzapfel and R. W. Ogden, ‘Constitutive modelling of passive myocardium: a structurally based framework for material characterization’, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1902, pp. 3445–3475, Sep. 2009, doi: 10.1098/rsta.2009.0091.

[2] R. Barnes, ‘Accelerating a fluvial incision and landscape evolution model with parallelism’, *Geomorphology*, vol. 330, pp. 28–39, Apr. 2019, doi: 10.1016/j.geomorph.2019.01.002.

[3] A. O’Hagan, ‘Bayesian analysis of computer code outputs: A tutorial’, *Reliability Engineering & System Safety*, vol. 91, no. 10, pp. 1290–1300, Oct. 2006, doi: 10.1016/j.ress.2005.11.025.

[4] U. Noè, A. Lazarus, H. Gao, V. Davies, B. Macdonald, K. Mangion, C. Berry, X. Luo, and D. Husmeier, ‘Gaussian process emulation to accelerate parameter estimation in a mechanical model of the left ventricle: a critical step towards clinical end-user relevance’, *Journal of The Royal Society Interface*, vol. 16, no. 156, p. 20190114, Jul. 2019, doi: 10.1098/rsif.2019.0114.

[5] R. B. Gramacy, *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. New York: Chapman and Hall/CRC, 2020. doi: 10.1201/9780367815493.

[6] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2005. doi: 10.7551/mitpress/3206.001.0001.

[7] D. Dalton, A. Lazarus, and D. Husmeier, ‘Comparative evaluation of different emulators for cardiac mechanics’, G. Ladde and S. Noelle, Eds., Ottawa, Canada: Avestia Publishing, 2020, p. 126. Accessed: Oct. 13, 2023. [Online]. Available: https://eprints.gla.ac.uk/225701/

[8] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, ‘Deep Kernel Learning’, Nov. 06, 2015, *arXiv*: arXiv:1511.02222. doi: 10.48550/arXiv.1511.02222.

[9] M. Titsias, ‘Variational Learning of Inducing Variables in Sparse Gaussian Processes’, in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, PMLR, Apr. 2009, pp. 567–574. Accessed: Jan. 26, 2024. [Online]. Available: https://proceedings.mlr.press/v5/titsias09a.html

[10] A. C. Damianou and N. D. Lawrence, ‘Deep Gaussian Processes’, Mar. 22, 2013, *arXiv*: arXiv:1211.0358. doi: 10.48550/arXiv.1211.0358.

[11] S. S. Garud, I. A. Karimi, and M. Kraft, ‘Design of computer experiments: A review’, *Computers & Chemical Engineering*, vol. 106, pp. 71–95, Nov. 2017, doi: 10.1016/j.compchemeng.2017.05.010.

[12] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’, Jan. 29, 2017, *arXiv*: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.