Advances in Engineering Software 195 (2024) 103686

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

GP+: A Python library for kernel-based learning via Gaussian processes

Amin Yousefpour, Zahra Zanjani Foumani, Mehdi Shishehbor, Carlos Mora, Ramin Bostanabad [*,1]

Department of Mechanical and Aerospace Engineering, University of California, Irvine, United States of America

ARTICLE INFO

Keywords:
Gaussian processes
Python library
Uncertainty quantification
Kernel methods
Manifold learning
Bayesian optimization

ABSTRACT

In this paper we introduce GP+, an open-source library for kernel-based learning via Gaussian processes (GPs) which are powerful statistical models that are completely characterized by their parametric covariance and mean functions. GP+ is built on PyTorch and provides a user-friendly and object-oriented tool for probabilistic learning and inference. As we demonstrate with a host of examples, GP+ has a few unique advantages over other GP modeling libraries. We achieve these advantages primarily by integrating nonlinear manifold learning techniques with GPs' covariance and mean functions. As part of introducing GP+, in this paper we also make methodological contributions that (1) enable probabilistic data fusion and inverse parameter estimation, and (2) equip GPs with parsimonious parametric mean functions which span mixed feature spaces that have both categorical and quantitative variables. We demonstrate the impact of these contributions in the context of Bayesian optimization, multi-fidelity modeling, sensitivity analysis, and calibration of computer models.

1. Introduction

Gaussian processes (GPs) are indispensable building blocks of many powerful probabilistic frameworks such as Bayesian optimization (BO) [1–11], function and operator learning [12–17], data-driven calibration of expensive simulations [18–23], and multi-fidelity (MF) modeling [24–28]. The recent software and hardware developments, combined with the new means of data collection and the society's drive to tackle ever-challenging goals, have sparked significant innovations in the broad field of machine learning (ML). GPs have also substantially benefited from these advancements and recent works have enabled them to leverage GPUs [29,30] and to accommodate high dimensions [31–35], large datasets [36–44], or disjoint feature spaces that have both qualitative and quantitative variables [16,45–48]. In line with these advancements, in this paper we introduce GP+ which is a modular and user-friendly library that aims to empower researchers and practitioners in harnessing the full potential of GPs for a wide range of applications such as emulation (i.e., probabilistic metamodeling), single- and multi-fidelity Bayesian optimization (SFBO and MFBO), kernel-based generalized MF modeling, inverse parameter estimation, anomaly detection, and sensitivity analyses.

As reviewed in Section 3, there are quite a few existing libraries for GP modeling and in fact we leverage one of them (i.e., GPyTorch [30]) in developing ours. While these libraries have been successfully used in many applications across sciences and engineering, we believe a few distinct features set GP+ aside. First, we design parametric covariance and prior mean functions that automatically enable GPs to fuse and emulate multi-source datasets, detect anomalies, find calibration parameters of computer models (i.e., inverse parameter estimation), and handle categorical variables. These functions leverage kernel-based nonlinear manifold learning, provide interpretable solutions (see Section 5 for multiple examples), and their parameters are all jointly learnt via the maximum a posteriori (MAP) method. Second, we provide a unified platform to use GPs for many tasks such as SFBO and MFBO, probabilistic regression, and sensitivity analysis. As shown in Section 5, all of these functionalities are achieved via a few lines of codes. As an example, we develop a hyperparameter estimation routine based on the method of continuation [49] which, at the expense of slightly higher computational costs, provides more numerical stability and accuracy. This method is accessible by merely changing the default optimization settings in GP+. Lastly, GP+ is accompanied with a rich set of datasets (from engineering applications) and benchmark analytic examples that can be used by researchers in evaluating the performance of emulation techniques beyond GPs.

The rest of this paper is organized as follows. In Section 2 we provide a brief background on GPs and then review the relevant literature in Section 3. We introduce the primary components of GP+ (i.e., its covariance and prior mean functions) in Section 4 where we also introduce the three new methodological innovations of this paper.

* Corresponding author.
E-mail address: Raminb@uci.edu (R. Bostanabad).
[1] GitHub Repository: https://github.com/Bostanabad-Research-Group/GP-Plus.

https://doi.org/10.1016/j.advengsoft.2024.103686
Received 12 January 2024; Received in revised form 17 April 2024; Accepted 21 May 2024

The first contribution is focused on the kernel and endows GPs with probabilistic embeddings that benefit both MF modeling (in terms of quantifying model-form errors) and mixed data emulation (in terms of learning the relations among the categorical variables and their levels). The second contribution is on designing parametric mean functions such that they naturally handle multi-source or mixed data that have categorical features. The third contribution is on inverse parameter estimation where the goal is to probabilistically calibrate computer models using limited high-fidelity (HF) data such as observations or experiments. We provide some details and examples on some of the most important functionalities of GP+ in Section 5 where we also conduct comparative studies against existing methods. Concluding remarks and future research directions are provided in Section 6.

1.1. Nomenclature

Unless otherwise stated, throughout the paper we denote scalars, vectors, and matrices with regular, bold lower-case, and bold upper-case letters, respectively (e.g., $x, \boldsymbol{x}$, and $\boldsymbol{X}$). Vectors are by default column vectors and subscript or superscript numbers enclosed in parenthesis indicate sample numbers. For instance, $x^{(i)}$ or $\boldsymbol{x}^{(i)}$ denote the ith sample in a training dataset while x_i indicates the ith component of the column vector $\boldsymbol{x} = \left[x_1, \dots, x_{dx}\right]^T$. For clarity, we sometimes indicate the size of vectors and matrices via subscripts, e.g., $\boldsymbol{x}_n$ and $\boldsymbol{X}_{nq}$. Specifying the size is useful in cases where we do not follow our notational convention that distinguishes vectors and matrices (e.g., $\mathbf{Y}_q$ is a vector while $\mathbf{Y}$ is a matrix).

Lastly, we distinguish between a function and samples taken from that function by specifying the functional dependence. As an example, $y(x)$ and $y(\boldsymbol{x})$ are functions while y and $\boldsymbol{y}$ are a scalar and a vector of values, respectively. We also assume functions accommodate batch computations. That is, a single-response function returns a column vector of n values if n inputs are simultaneously fed into it, i.e., $\boldsymbol{y} = y(\boldsymbol{X})$.

2. Background on Gaussian processes

To explain the working principles of GPs, we consider $\mathbf{Y}_q$ and $\mathbf{Y}_n$ which are two jointly normal random vectors of sizes q and n, respectively.[2] We write this joint distribution as:

$$p\left(\begin{bmatrix}\mathbf{Y}_q\\ \mathbf{Y}_n\end{bmatrix}\right) = \mathcal{N}_{q+n}\left(\begin{bmatrix}\boldsymbol{\mu}_q\\ \boldsymbol{\mu}_n\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{qq} & \boldsymbol{\Sigma}_{qn}\\ \boldsymbol{\Sigma}_{nq} & \boldsymbol{\Sigma}_{nn}\end{bmatrix}\right) \tag{1}$$

where the subscripts indicate the array sizes, $\boldsymbol{\mu}_q = \mathbb{E}[\mathbf{Y}_q]$, $\boldsymbol{\Sigma}_{qq} = cov(\mathbf{Y}_q) = \mathbb{E}[(\mathbf{Y}_q - \boldsymbol{\mu}_q)(\mathbf{Y}_q - \boldsymbol{\mu}_q)^T]$, and $\boldsymbol{\Sigma}_{qn} = \boldsymbol{\Sigma}_{nq}^T = \mathbb{E}[(\mathbf{Y}_q - \boldsymbol{\mu}_q)(\mathbf{Y}_n - \boldsymbol{\mu}_n)^T]$. If $\boldsymbol{y}_n$ is observed for $\mathbf{Y}_n$, we can update our knowledge on $\mathbf{Y}_q$ based on its conditional distribution:

$$p\left(\mathbf{Y}_q \mid \mathbf{Y}_n = \boldsymbol{y}_n\right) = \mathcal{N}_q\left(\boldsymbol{\mu}_q^*, \boldsymbol{\Sigma}_{qq}^*\right) \tag{2}$$

where $\boldsymbol{\mu}_q^* = \boldsymbol{\mu}_q + \boldsymbol{\Sigma}_{qn}\boldsymbol{\Sigma}_{nn}^{-1}(\boldsymbol{y}_n - \boldsymbol{\mu}_n)$ is the conditional mean vector and $\boldsymbol{\Sigma}_{qq}^* = \boldsymbol{\Sigma}_{qq} - \boldsymbol{\Sigma}_{qn}\boldsymbol{\Sigma}_{nn}^{-1}\boldsymbol{\Sigma}_{nq}$ is the conditional covariance matrix. Similarly, in interpolation with GPs one first assumes that the given data $\boldsymbol{y}_n$ and the to-be-predicted values $\boldsymbol{y}_q$ are jointly Gaussian and then infers the latter via Eq. (2). However, the mean vectors and covariance matrices in Eq. (1) are unknown and thus Eq. (2) can be used for prediction only after we (1) endow the underlying GP with a parametric mean function and a parametric covariance function (or kernel), and (2) estimate the parameters of these two sets of functions.[3]

[2] For this part of the description, we distinguish between the random vector $\mathbf{Y}_q$ and the specific realization $\boldsymbol{y}_q$ that it takes.

[3] In a fully Bayesian setting, instead of estimating the parameters, their posterior distributions are obtained and predictions on $\boldsymbol{y}_q$ require marginalization with respect to these distributions. Due to the significantly higher computational costs of fully Bayesian techniques and their marginal accuracy improvements in the case of GPs, we recommend and use MAP.

More formally, assume the training dataset $\left\{\boldsymbol{x}^{(i)}, y^{(i)}\right\}_{i=1}^n$ is given where $\boldsymbol{x} = [x_1, \dots, x_{dx}]^T \in \mathbb{X} \subset \mathbb{R}^{dx}$ and $y^{(i)} = y(\boldsymbol{x}^{(i)}) \in \mathbb{R}$ denote the inputs and response,[4] respectively. Given $\boldsymbol{y} = [y^{(1)}, \dots, y^{(n)}]^T$ and $\boldsymbol{X}$ whose ith row is $(\boldsymbol{x}^{(i)})^T$, our goal is to predict $y(\boldsymbol{x}^*)$ at the arbitrary point $\boldsymbol{x}^* \in \mathbb{X}$. Following the above description, we assume $\boldsymbol{y} = [y^{(1)}, \dots, y^{(n)}]^T$ is a realization of a GP with the following parametric mean and covariance functions:

$$\mathbb{E}[y(\boldsymbol{x})] = m(\boldsymbol{x}; \boldsymbol{\beta}), \tag{3a}$$

$$\operatorname{cov}\left(y(\boldsymbol{x}), y(\boldsymbol{x}')\right) = c(\boldsymbol{x}, \boldsymbol{x}'; \sigma^2, \boldsymbol{\theta}) = \sigma^2 r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}) \tag{3b}$$

where $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are the parameters of the mean and covariance functions, respectively. The mean function in Eq. (3a) can take on many forms such as polynomials or even a feedforward neural network (FFNN). In many applications of GP modeling, a constant value is used as the mean function (i.e., $m(\boldsymbol{x}; \boldsymbol{\beta}) = \beta$) in which case the performance of the GP depends entirely on its kernel. In Eq. (3b), σ^2 is the process variance (or inverse precision) and $r(\cdot, \cdot)$ is the correlation function whose parameters are collectively denoted via $\boldsymbol{\theta}$. Common choices for $r(\cdot, \cdot)$ are the Gaussian, power exponential, and Matérn correlation functions defined as:

$$r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}) = \exp\left\{-\sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2\right\} \tag{4a}$$

$$r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}, p) = \exp\left\{-\sum_{i=1}^{dx} 10^{\omega_i}|x_i - x_i'|^p\right\} \tag{4b}$$

$$r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}) = \frac{2^{1-\nu}}{\Gamma(\nu)} K_\nu\left(\sqrt{2\nu} \times \sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2\right)^{1+\nu} \tag{4c}$$

where $\omega_i \in \mathbb{R}$,[5] $p \in [1, 2]$, $\nu \in \{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}\}$, K_ν is the modified Bessel function of the second kind, and Γ is the gamma function. The inductive bias that the kernels in Eq. (4) encode into the learning process is that close-by input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ have similar (i.e., correlated) output values. The degree of this correlation is quantified by the interpretable length-scale (aka roughness) parameters where the magnitude of 10^{ω_i} is directly related to the response fluctuations along x_i.

Having defined these kernels we can now write the likelihood function of the observation vector $\boldsymbol{y}$ as:

$$p(\boldsymbol{y}; \boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) = (2\pi)^{-\frac{n}{2}}|\boldsymbol{C}|^{-\frac{1}{2}} \times \exp\left\{\frac{-1}{2}(\boldsymbol{y} - \boldsymbol{m})^T\boldsymbol{C}^{-1}(\boldsymbol{y} - \boldsymbol{m})\right\} \tag{5}$$

where $|\cdot|$ denotes the determinant operator, $\boldsymbol{C}_{nn} := c(\boldsymbol{X}, \boldsymbol{X}; \sigma^2, \boldsymbol{\theta})$ is the covariance matrix whose (i, j)th element is $C_{ij} = c(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}; \sigma^2, \boldsymbol{\theta}) = \sigma^2 r(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}; \boldsymbol{\theta})$, and $\boldsymbol{m}$ is an $n \times 1$ vector whose ith element is $m_i = m(\boldsymbol{x}^{(i)}; \boldsymbol{\beta})$. The point estimates for $\boldsymbol{\beta}, \sigma^2$, and $\boldsymbol{\theta}$ can now be found by maximizing the likelihood function in Eq. (5). Alternatively, Bayes' rule can be used to leverage prior knowledge in estimating these parameters. Specifically, the joint posterior distribution of the parameters is:

$$p(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta})p(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta})}{p(\boldsymbol{y})}, \tag{6}$$

where $p(\boldsymbol{y})$ is the evidence. Since $p(\boldsymbol{y})$ is a normalizing constant, we can find the MAP estimates of $\boldsymbol{\beta}, \sigma^2$, and $\boldsymbol{\theta}$ by maximizing the right-hand-side of Eq. (6). That is:

$$[\widehat{\boldsymbol{\beta}}, \widehat{\sigma^2}, \widehat{\boldsymbol{\theta}}] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}}{\operatorname{argmax}} |2\pi\boldsymbol{C}|^{-\frac{1}{2}} \times \exp\left\{\frac{-1}{2}(\boldsymbol{y} - \boldsymbol{m})^T\boldsymbol{C}^{-1}(\boldsymbol{y} - \boldsymbol{m})\right\} \times p(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) \tag{7}$$

[4] We focus on regression problems whose output dimensionality is one but note that GPs can handle multi-response or multi-task problems as well [50–52].

[5] To ensure numerical stability, ω_i is typically bounded to a subset of $\mathbb{R}$, e.g., $[-10, 4]$.

or equivalently:

$$[\hat{\boldsymbol{\beta}}, \hat{\sigma}^2, \hat{\boldsymbol{\theta}}] = \underset{\boldsymbol{\beta},\sigma^2,\boldsymbol{\theta}}{\operatorname{argmin}} \; L_{MAP}$$
$$= \underset{\boldsymbol{\beta},\sigma^2,\boldsymbol{\theta}}{\operatorname{argmin}} \; \frac{1}{2}\log(|\boldsymbol{C}|) + \frac{1}{2}(\boldsymbol{y}-\boldsymbol{m})^T\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{m}) - \log\left(p(\boldsymbol{\beta},\sigma^2,\boldsymbol{\theta})\right) \tag{8}$$

where $\log(\cdot)$ denotes the natural logarithm. We can now efficiently estimate all the model parameters by minimizing Eq. (8) via a gradient-based optimization algorithm[6] and then adopt Eq. (2) to obtain the mean and variance of the response distribution at the arbitrary point $\boldsymbol{x}^*$:

$$\mathbb{E}[y(\boldsymbol{x}^*)] = \mu(\boldsymbol{x}^*) = m(\boldsymbol{x}^*;\hat{\boldsymbol{\beta}}) + c(\boldsymbol{x}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{m}) \tag{9a}$$

$$\operatorname{cov}(y(\boldsymbol{x}^*), y(\boldsymbol{x}^*)) = \tau^2(\boldsymbol{x}^*) = c(\boldsymbol{x}^*,\boldsymbol{x}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) - c(\boldsymbol{x}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}^{-1}c(\boldsymbol{X},\boldsymbol{x}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) \tag{9b}$$

where $c(\boldsymbol{x}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)$ is a $1\times n$ row vector with entries $c_i = c(\boldsymbol{x}^*,\boldsymbol{x}^{(i)};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)$ and its transpose is $c(\boldsymbol{X},\boldsymbol{x}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2)$. Eqs. (9a) and (9b) can be straightforwardly extended to predict the response distribution for a batch of samples denoted by $\boldsymbol{X}^*$:

$$\mathbb{E}[y(\boldsymbol{X}^*)] = m(\boldsymbol{X}^*;\hat{\boldsymbol{\beta}}) + c(\boldsymbol{X}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{m}) \tag{10a}$$

$$\operatorname{cov}\left(y\left(\boldsymbol{X}^*\right), y(\boldsymbol{X}^*)\right) = c(\boldsymbol{X}^*,\boldsymbol{X}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) - c(\boldsymbol{X}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}^{-1}c(\boldsymbol{X},\boldsymbol{X}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) \tag{10b}$$

The above formulations build interpolating GPs. To handle datasets with noisy observations, the nugget or jitter parameter, denoted by δ [12,53,54], is used where $\boldsymbol{C}$ is replaced by $\boldsymbol{C}_\delta = \boldsymbol{C} + \delta\boldsymbol{I}_{nn}$ where $\boldsymbol{I}_{n\times n}$ is the $n\times n$ identity matrix (with this adjustment, the stationary noise variance estimated by the GP is $\hat{\delta}$).[7] In addition to modeling stationary noise, the nugget parameter is also used to mitigate the numerical issues associated with $\boldsymbol{C}$. That is, even with noise-free $\boldsymbol{y}$, $\boldsymbol{C}_\delta$ is used while minimizing Eq. (8) to ensure the correlation matrix is always invertible. When the nugget parameter is used for fitting a GP to noisy observations, Eq. (1) takes on the following form:

$$p\left(\begin{bmatrix}\mathbf{Y}_q\\ \mathbf{Y}_n\end{bmatrix}\right) = \mathcal{N}_{q+n}\left(\begin{bmatrix}\boldsymbol{\mu}_q\\ \boldsymbol{\mu}_n\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{qq}+\delta\boldsymbol{I}_{qq} & \boldsymbol{\Sigma}_{qn}\\ \boldsymbol{\Sigma}_{nq} & \boldsymbol{\Sigma}_{nn}+\delta\boldsymbol{I}_{nn}\end{bmatrix}\right) \tag{11}$$

which means that Eq. (10) should be updated to:

$$\mathbb{E}[y(\boldsymbol{X}^*)] = m(\boldsymbol{X}^*;\hat{\boldsymbol{\beta}}) + c(\boldsymbol{X}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}_\delta^{-1}(\boldsymbol{y}-\boldsymbol{m}) \tag{12a}$$

$$\operatorname{cov}\left(y\left(\boldsymbol{X}^*\right), y(\boldsymbol{X}^*)\right) = c(\boldsymbol{X}^*,\boldsymbol{X}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) - c(\boldsymbol{X}^*,\boldsymbol{X};\hat{\boldsymbol{\theta}},\hat{\sigma}^2)\boldsymbol{C}_\delta^{-1} \times c(\boldsymbol{X},\boldsymbol{X}^*;\hat{\boldsymbol{\theta}},\hat{\sigma}^2) + \hat{\delta}\boldsymbol{I}. \tag{12b}$$

We highlight that Eq. (12b) does not consider the additional uncertainties incurred by estimating the parameters of the mean and covariance functions (note that Eq. (2) assumed the mean vector and covariance matrices are known). These additional uncertainties can be quantified by building and using the GP within a Bayesian framework where sampling methods (e.g., Markov Chain Monte Carlo or MCMC) are required for marginalization as closed-form expressions are only available for specific cases (see [58] for an example). Since such sampling methods are typically expensive and the provided benefits are marginal, MAP is frequently employed in GP modeling [59].

[6] Since the profile of the objective function in Eq. (8) has many local minima, it is important to start the gradient-based optimization via multiple initial guesses. We control this setting in GP+ via the `num_restarts` parameter whose default value is 32.

[7] Some recent works [55–57] apply the nugget directly to $\boldsymbol{R}$ but herein we adhere to [53] and add δ to $\boldsymbol{C}$.

3. Related works

Many open-source GP libraries have been recently developed and in this section we review some of the most well-known ones. One of the earliest open-source GP packages is TreedGP [60] which is developed in R and primarily aims to address the stationarity and scalability issues of GPs. In particular, TreedGP recursively partitions the input space via parallel and axes-aligned boundaries [61] and then endows each partition with a GP whose covariance function is stationary. TreedGP uses Bayesian averaging to combine these GPs which is particularly important for obtaining smooth predictions on the partition boundaries. The major limitations of TreedGP are its non-differentiability on the boundaries, high computational costs (as the Bayesian analyses rely on MCMC [62]), reliance on trees which can only partition the input space with axis-aligned boundaries [63], and inability to efficiently handle categorical features in small-data applications.

GPfit [57] and GPM [56] are also R packages and they are primarily designed to improve the hyper-parameter optimization process at the expense of increased computational costs. GPfit has a multi-step pre-processing stage that aims to improve the quality of the initial points that are used via L-BFGS[8] in minimizing Eq. (8). Unlike GPfit, GPM develops a multi-step continuation-based strategy to increase both the robustness and accuracy of the optimization process. In particular, GPM indirectly controls δ via the auxiliary parameter ϵ that puts a lower bound on the smallest eigenvalue of the correlation matrix.[9] GPM first uses a large value for ϵ (e.g., 10^{-2}) and minimizes Eq. (8) while requiring the smallest eigenvalue of $\boldsymbol{R}$ to always be larger than the imposed ϵ. In addition to guaranteeing numerical robustness, this requirement dramatically smooths the profile of the objective function and hence most (if not all) optimizations quickly converge to the same solution. Then, GPM relaxes the constraint on $\boldsymbol{R}$ (e.g., $\epsilon = 10^{-3}$) and repeats the optimization while using the solution(s) of the previous step as the initial guess(es) in the current step, see Fig. 1. These steps are continued until the minimum value of ϵ is reached and then the parameters of the final GP are chosen by identifying the step (or ϵ) at which the leave-one-out cross-validation (LOO-CV) error of the model is minimized. GPy [64] is a popular object-oriented library that is implemented based on numeric Python (NumPy) by the Sheffield machine learning group. GPy provides a number of basic and advanced functionalities for GP regression that include multi-output learning and non-Gaussian likelihood functions which are accompanied with Laplace approximation [65] and expectation propagation since exact inference with non-Gaussian likelihoods is not tractable. However, GPy does not fully leverage modern hardware capabilities (e.g., GPU acceleration) and integration with deep neural networks (NNs) which are increasingly crucial in contemporary GP applications. It also lacks some of the most recent advancements that enable GPs to accommodate high dimensions or categorical features.

One of the first open-source GP libraries that supports GPU acceleration and leverages automatic differentiation is GPflow [29] which is based on Tensorflow and has an object-oriented Python front-end. GPflow supports regression and classification problems, uses variationally sparse methods for scalability to large data, and provides both Bayesian and point-estimate-based inference classes for Gaussian and non-Gaussian likelihoods. While GPflow has significant capabilities, it lacks some of the key recent advancements in GPs such as natural integration with SFBO or MFBO frameworks, fusing multi-source data, calibrating unknown parameters, or directly supporting categorical variables.

Perhaps the most widely used open-source package for GP modeling is GPytorch [30] which accommodates a wealth of functionalities such

[8] Limited-memory Broyden–Fletcher–Goldfarb–Shanno.

[9] The rationale behind this choice is that the smallest eigenvalue of $\boldsymbol{R}$ can sometimes be negative due to numerical issues.

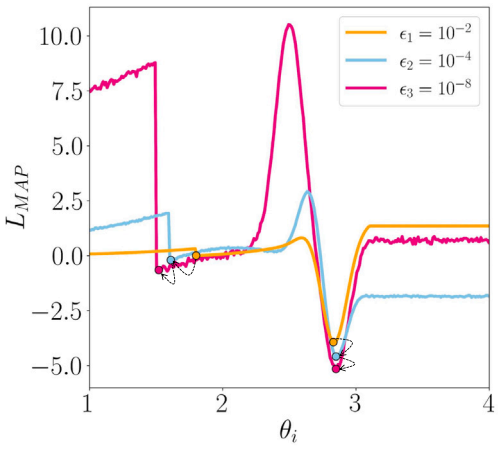


Fig. 1. Schematic illustration of continuation-based optimization: The profile of L_{MAP} in Eq. (8) is smoothed with a larger ϵ (or, equivalently, a larger nugget).

as deep kernel [66] and multi-task/multi-output learning [50–52], dimensionality reduction via latent variable GPs [67,68], and variational and approximate modeling for handling non-Gaussian likelihoods (e.g., in classification) or large datasets. A particular feature of GPytorch is its use of preconditioned conjugate gradients in accelerating the expensive computations (i.e., matrix inversion and determinant calculation) associated with exact GPs in Eqs. (8) and (10). This feature is coined blackbox matrix-matrix multiplication (BBMM) and is uniquely suited for GPU-based computations. GPytorch relies on Pyro [69] for probabilistic programming, forms the backbone of BoTorch which is an open-source Python library for BO, and has specialized kernels for MF modeling or handling categorical variables. Given the versatility and robustness of GPytorch, we use some of its core functionalities to build GP+ while addressing the limitations of GPytorch in handling categorical features, directly solving inverse problems, learning from MF datasets, or interfacing with SFBO/MFBO engines.

Other notable open-source GP libraries that we mention in passing include those of Ambikasaran et al. [70] which hierarchically factors the covariance matrix[10] into a product of block low-rank updates of the identity matrix to accelerate matrix inversion and determinant calculation (their method loses accuracy for $dx > 3$), Vanhatalo et al. [71] which is a Matlab library that integrates various elementary computational tools (e.g., sparse approximation) for GP-based regression, and GPML [12] which is also a Matlab library and has been widely used in a wide range of applications.

4. Kernel-based learning

The *vanilla* GP formulations reviewed in Section 2 break down in high dimensions or with large samples [72,73], do not directly accommodate MF modeling or MFBO [74,75], and cannot handle categorical features [76]. Since the scalability issue of GPs is rigorously studied in many recent works, in GP+ we focus on holistically addressing the latter two limitations based on the ideas that were first introduced in [76]. In particular, we generalize the concept of kernel-based learning for GPs by introducing new bases and kernels with customized parametric functions that directly enable probabilistic learning from multi-source data and handling qualitative features. Compared to existing works that also develop new kernels for GPs (see for example [77–79] for handling categorical inputs, building multi-response emulators, and MF modeling), our functions are quite versatile and produce nonlinearly learned embeddings that, while being low-dimensional and highly interpretable, enable GPs to model more complex relations.

To explain our kernel-based approach, we consider an emulation scenario where the input space includes two qualitative features $t_1 = \{Math, Chemistry\}$ and $t_2 = \{Japan, France, Canada\}$ which have $l_1 = 2$ and $l_2 = 3$ levels, respectively. Vanilla GPs cannot directly work with $\boldsymbol{t} = [t_1, t_2]^T$ since typical kernels such as those in Eq. (4) require each feature to be associated with a distance metric while categorical variables naturally lack such measures. As schematically illustrated in Fig. 2, we address this limitation by first endowing the categorical variables $\boldsymbol{t} = [t_1, \ldots, t_{dt}]^T$ with the quantitative prior representations $\boldsymbol{\pi}_t = f_\pi(\boldsymbol{t})$ where $f_\pi(\cdot)$ is a deterministic user-specified function. These priors are typically high dimensional (i.e., $d\pi > dt$) and can be designed in many ways (we describe some of these below, see supplementary comments on our GitHub page for more options). To reduce the dimensionality of these representations while learning the effects of $\boldsymbol{t}$ on the response, we then pass $\boldsymbol{\pi}_t$ through the parametric embedding function $f_h(\boldsymbol{\pi}_t; \boldsymbol{\theta}_h)$ to obtain $\boldsymbol{h}$ which is a dh dimensional *latent* representation of $\boldsymbol{t}$ where $d\pi \gg dh$. Since $\boldsymbol{h} = f_h(f_\pi(\boldsymbol{t}); \boldsymbol{\theta}_h)$ are quantitative, they can be easily used to develop new kernels. For instance, we can extend the Gaussian and Matérn correlation functions as:

$$r\left(\boldsymbol{u}, \boldsymbol{u}'; \boldsymbol{\omega}, \boldsymbol{\theta}_h\right) = \exp\left\{ -\sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2 - \sum_{i=1}^{dh}(h_i - h_i')^2 \right\} \tag{13a}$$

$$r\left(\boldsymbol{u}, \boldsymbol{u}'; \boldsymbol{\omega}, \boldsymbol{\theta}_h\right) = \frac{2^{1-\nu}}{\Gamma(\nu)} K_\nu \left(\sqrt{2\nu} \times \sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2 + \sum_{i=1}^{dh}(h_i - h_i')^2 \right)^{1+\nu} \tag{13b}$$

where $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \end{bmatrix}$. We note that (1) no scale parameters are associated with $\boldsymbol{h}$ in Eqs. (13a) and (13b) since, as opposed to $\boldsymbol{x}$, $\boldsymbol{h}$ are learnt, and (2) $\boldsymbol{\theta}_h$ are estimated jointly with the other parameters of the GP via MAP where the covariance matrix in Eq. (8) is now built via one of the correlation functions in Eq. (13).

As we explain in the proceeding subsections, the above kernel reformulations not only allow GPs to operate in feature spaces with categorical variables, but they also enable GPs to directly fuse MF datasets (from an arbitrary number of sources) or inversely estimate calibration parameters. Given this general applicability of our approach, we have equipped GP+ with various mechanisms to design the priors and parameterize the embeddings. We believe these options increase the interpretability of the model (in particular, the learnt embeddings) as well as computational efficiency.

In Fig. 2 we schematically demonstrate a few options for designing $\boldsymbol{\pi}_t$ and the embedding functions for an emulation example where the feature space has quantitative variables $\boldsymbol{x}$ and the two categorical variables $\boldsymbol{t} = [t_1, t_2]$ mentioned above. As shown in the top row of the embedding block, $f_\pi(\cdot)$ can simply be a deterministic bijective[11] function that (1) groups the one-hot-encoded representations of $\boldsymbol{t}$ into a single matrix (this option is the default in GP+), (2) builds a random matrix whose unique rows correspond to the unique combinations of $\boldsymbol{t}$, or (3) constructs multiple matrices where each one corresponds to the one-hot-encoding of one of the categorical variables. The second row in the embedding block of Fig. 2 illustrates two options for $f_h(\boldsymbol{\pi}_t; \boldsymbol{\theta}_h)$ that consist of parametric matrices (denoted by $\boldsymbol{A}$ in Fig. 2) and FFNNs. The construction of $f_h(\boldsymbol{\pi}_t; \boldsymbol{\theta}_h)$ is affected by $f_\pi(\cdot)$. For instance, the row size of $\boldsymbol{A}$ depends on $\boldsymbol{\pi}_t$[12] while its column size is chosen by the user and determines the dimensionality of the to-be-learnt embedding.

[10] As long as it is built with specific covariance functions such as the Gaussian or Matérn.

[11] Surjective and injective functions may also be used especially if some prior knowledge encourages such choices. We focus on bijective functions in this paper and leave other choices for future studies.

[12] Hence, the random prior encoding can work with a smaller $\boldsymbol{A}$ compared to the grouped one-hot-encoding.

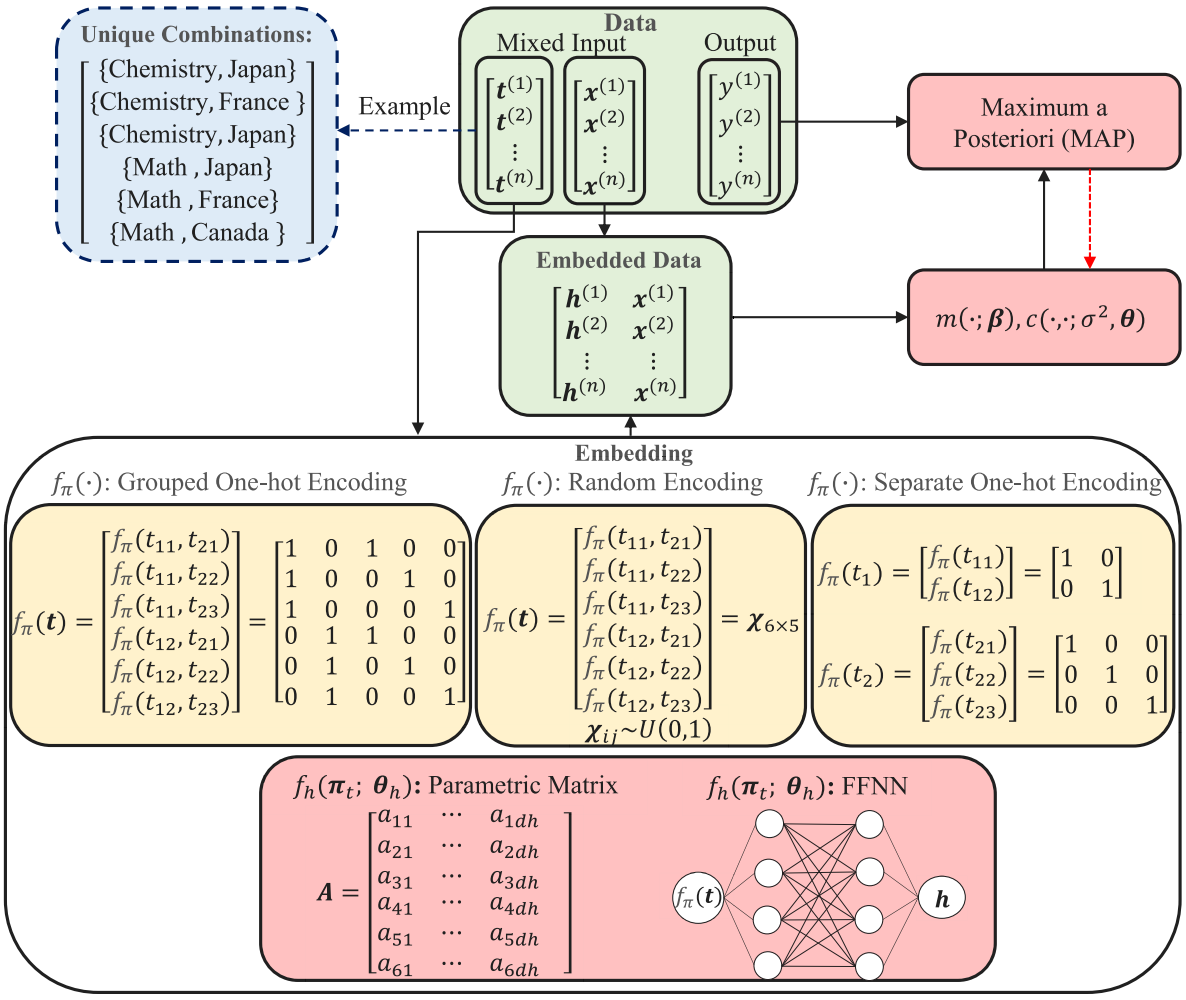


Fig. 2. Emulation via GP+ in mixed input spaces: We first endow the categorical variables $\boldsymbol{t}$ with some quantitative prior representations which are then mapped to a low-dimensional embedding with a parametric function. The embedded variables $\boldsymbol{h}$ are then concatenated with $\boldsymbol{x}$ and fed into the mean and covariance functions. All the model parameters are jointly learnt via MAP.

The number of $\boldsymbol{A}$ matrices also depends on the priors since it should match with the number of matrices that $f_\pi(\cdot)$ generates. We note that these dependencies are automatically enforced in GP+ and the available options for $f_\pi(\cdot)$ or $f_h(\boldsymbol{\pi}_t;\boldsymbol{\theta}_h)$ in GP+ can be easily accessed by changing its default settings.

Once the parametrized embedding is constructed with any of the procedures described above (or other settings available in GP+), they are concatenated with the numerical features and used in our reformulated mean and covariance functions to build the likelihood function. Then, all the model parameters are estimated via MAP. In the following subsections, we elaborate on how these embeddings as well as reformulated mean and covariance functions benefit MF modeling, inverse parameter estimation, and MFBO.

4.1. Multi-fidelity modeling via deterministic embedding

The premise of MF modeling is to leverage low-fidelity (LF) data to reduce the reliance on expensive high-fidelity (HF) samples in many-query applications such as design optimization [80,81], uncertainty propagation and variance reduction [82], BO, calibration of computer models [83], and sensitivity analysis [84]. With the exception of a few recent works such as [11,85,86], most existing MF techniques fuse only two data sources while imposing a specific functional relation between them. For instance, the method of Kennedy and O'Hagan (KOH) [22] and its various extensions [18–21,87–91] fuse the HF and LF data based on the following generic relation:

$$y_h(\boldsymbol{x}) = \rho \times y_l(\boldsymbol{x},\boldsymbol{\zeta}) + y_b(\boldsymbol{x},\boldsymbol{\zeta}) + \varepsilon \tag{14}$$

where $y_h(\boldsymbol{x})$ and $y_l(\boldsymbol{x},\boldsymbol{\zeta})$ denote the HF and LF sources, respectively, $y_b(\boldsymbol{x},\boldsymbol{\zeta})$ is the bias function that aims to quantify the systematic bias of the LF data source, $\boldsymbol{\zeta}$ are the calibration parameters whose values must be inversely estimated during the fusion process (see Section 4.4), and ε denotes normal noise whose variance may be known or not. Eq. (14) is based on some strong assumptions that do not always hold in practice (e.g., existence of only one LF data source whose bias is additive). To dispense with such inflexible assumptions, the MF modeling capabilities of GP+ are based on converting the fusion process into a nonlinear latent variable learning problem [26].

Suppose we have ds data sources of varying accuracy levels and aim to emulate all sources while dealing with (1) scarce data (especially from accurate sources), (2) unknown and source-dependent noise variances, and (3) nontrivial biases of LF sources with respect to the HF source, i.e., we do not rely on any knowledge on the relative accuracy of the LF sources and their bias form (e.g., additive, multiplicative, etc.), see Fig. 3. To this end GP+ first augments the input space with the additional *categorical* variable $s = \{\text{'1'},\dots,\text{'ds'}\}$ whose jth element corresponds to data source j for $j = 1,\dots,ds$. Upon this augmentation,

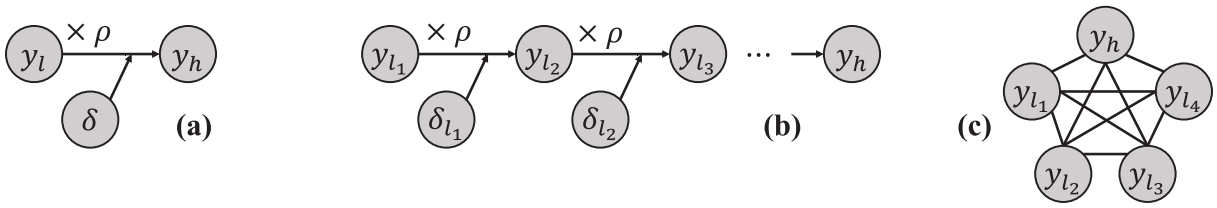


Fig. 3. Graphical representation of multi-fidelity modeling techniques: The method of KOH [22] **(a)** and its extension to hierarchical techniques **(b)** impose specific relations between the data sources. However, GP+ **(c)** does not impose any prior relation among the data sources and its structure resembles an undirected graph.

the ds datasets are concatenated as:

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}_1 & {}'\mathbf{1}'_{n_1\times 1} \\ \boldsymbol{U}_2 & {}'\mathbf{2}'_{n_2\times 1} \\ \vdots & \vdots \\ \boldsymbol{U}_{ds} & {}'\mathbf{ds}'_{n_{ds}\times 1} \end{bmatrix} \text{ and } \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_{ds} \end{bmatrix} \tag{15}$$

where the subscripts $1, 2, \ldots, ds$ correspond to the data sources, n_j is the number of samples obtained from source j, $\boldsymbol{U}_j$ and $\boldsymbol{y}_j$ are, respectively, the $n_j \times (dx + dt)$ feature matrix and the $n_j \times 1$ vector of responses obtained from $s(j)$, and $'\boldsymbol{j}'$ is a categorical vector of size $n_j \times 1$ whose elements are all set to $'j'$. Once the unified $\{\boldsymbol{U}, \boldsymbol{y}\}$ dataset is built, GP+ fits an emulator to it following a process similar to Fig. 2. Given the importance of identifying the relative discrepancies among data sources, GP+ slightly changes the correlation functions in Eq. (13) to learn two embeddings where the first one encodes the categorical variables in the input space (denoted by $\boldsymbol{t}$ in Fig. 2) while the second one encodes the data source identifier (s). Following this modeling assumption, the correlation functions in Eq. (13) are updated as:

$$r(\boldsymbol{u}, \boldsymbol{u}'; \boldsymbol{\omega}, \boldsymbol{\theta}_h, \boldsymbol{\theta}_z) = \exp\left\{ -\sum_{i=1}^{dx} 10^{\omega_i}(x_i - x'_i)^2 - \sum_{i=1}^{dh}(h_i - h'_i)^2 - \sum_{i=1}^{dz}(z_i - z'_i)^2 \right\} \tag{16a}$$

$$r(\boldsymbol{u}, \boldsymbol{u}'; \boldsymbol{\omega}, \boldsymbol{\theta}_h, \boldsymbol{\theta}_z) = \frac{2^{1-\nu}}{\Gamma(\nu)} K_\nu \times \left(\sqrt{2\nu} \times \sum_{i=1}^{dx} 10^{\omega_i}(x_i - x'_i)^2 + \sum_{i=1}^{dh}(h_i - h'_i)^2 + \sum_{i=1}^{dz}(z_i - z'_i)^2 \right)^{1+\nu} \tag{16b}$$

where $\boldsymbol{u} = [\boldsymbol{x}, \boldsymbol{t}, s]^T$ and $\boldsymbol{z} = f_z(\boldsymbol{\pi}_s; \boldsymbol{\theta}_z)$ is the latent representation of data source s and is obtained similar to $\boldsymbol{h}$. Looking at Eq. (16) we observe that the correlation between the estimated outputs of sources s and s' at the same inputs is:

$$0 \le r\left(\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \\ s \end{bmatrix}, \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{t} \\ s' \end{bmatrix}\right) = \exp\left\{ 0 - 0 - \sum_{i=1}^{d_z}(z_i - z'_i)^2 \right\} \le 1 \tag{17}$$

which illustrates that highly correlated data sources must have similar latent representations, (i.e., they must be encoded with close-by points in the z−space), see Section 5 for multiple examples.

We highlight that these learned latent distances provide an average measure of correlation among the sources and cannot identify local discrepancies since the encodings in the z−space are not functions of $\boldsymbol{x}$ or $\boldsymbol{t}$. Thus, if some LF sources are only locally correlated with the HF source, they will be encoded relatively far from the HF source in the learned embedding. This implies that those sources provide valuable insights in certain areas of the domain and keeping or dropping them depends on the specific applications. For instance, if the application is emulation, techniques such as cross-validation or train-test splits can assist in determining which sources to keep or drop. However if the goal is multi-fidelity BO (which typically starts with very small initial data, especially from the HF source), we recommend keeping all the sources during the optimization process (see [92] for more details).

MF modeling in GP+ differs significantly from most existing methods in that its structure does not prioritize learning any source (e.g., the HF source) over the others, i.e., GP+ aims to integrate all the data sets together to improve its accuracy in emulating all the sources. For example, multilevel best linear unbiased estimators (MBLUE) and approximate control variate (ACV) are two variance reduction-based techniques that leverage MF data to more accurately learn the HF source [93,94]. Since these methods prioritize surrogating the HF source and do not build surrogates for the LF sources, they cannot be used in applications such as MFBO where one has to emulate all sources. In this paper, we do not explore the possibility of prioritizing emulation of a particular source (e.g., the HF source) but note that this direction can be pursued in a number of ways such as constraining the embeddings, penalizing the objective function in Eq. (8), or designing specific priors.

4.1.1. Source-dependent noise modeling

Noise inevitably arises in most applications and incorrectly modeling it reduces the performance of any emulator. As mentioned in Section 2, GPs model noise via the nugget or jitter parameter, δ, which changes the covariance matrix from $\boldsymbol{C}$ to $\boldsymbol{C}_\delta = \boldsymbol{C} + \delta \boldsymbol{I}_{nn}$. Although this approach works quite well in SF problems, it does not yield the same benefits in MF emulation due to the dissimilar nature of the data sources and their corresponding noises. Consider a bi-fidelity scenario where the HF data comes from an experimental setup and is subject to measurement noise, while the LF data is generated by a deterministic computer code that has a systematic bias due to missing physics. In this case, using only one nugget parameter for MF emulation is obviously not an optimum choice.

To address this issue effectively, we follow [92] and use a nugget vector $\boldsymbol{\delta} = [\delta_1, \delta_2, \ldots, \delta_{ds}]$ to modify the covariance matrix:

$$\boldsymbol{C}_\delta = \boldsymbol{C} + \boldsymbol{N}_\delta \tag{18}$$

where $\boldsymbol{N}_\delta$ denotes an $n \times n$ diagonal matrix whose (i,i)th element is the nugget element corresponding to the data source of the ith sample. For instance, suppose the ith sample $\boldsymbol{u}^{(i)}$ is generated by source ds. Then, (i,i)th element of $\boldsymbol{N}_\delta$ is δ_{ds}. With this modification, the estimated stationary noise variance for the ith data source is $\widehat{\delta}_i$. We highlight that GP+ uses Eq. (18) by default when learning from multi-source data and updates the training and inference formula accordingly. For instance, all model parameters in this case are obtained as:

$$[\widehat{\boldsymbol{\beta}}, \widehat{\sigma}^2, \widehat{\boldsymbol{\theta}}, \widehat{\boldsymbol{\delta}}] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}, \boldsymbol{\delta}}{\operatorname{argmin}} \ \frac{1}{2}\log(|\boldsymbol{C}_\delta|) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{m})^T \boldsymbol{C}_\delta^{-1}(\boldsymbol{y} - \boldsymbol{m}) - \log\left(p(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}, \boldsymbol{\delta})\right) \tag{19}$$

We highlight that the above formulations model stationary noise for each data source. We make this choice due to the fact that modeling an input-dependent noise increases the number of hyperparameters by at least $ds \times dx$ which can result in overfitting [95,96]. Therefore, to balance the risk of overfitting with the uncertainty quantification capacity of our emulator, we assume the noise variance is not a function of the input variables and only depends on the data source.

4.2. Multi-fidelity modeling via probabilistic embedding

The MF modeling approach described in Section 4.1 is deterministic in that the learnt embedding encodes a data source with a single point

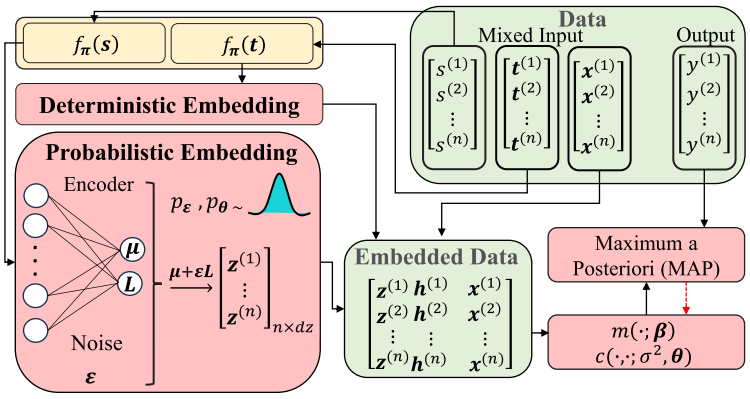


Fig. 4. Probabilistic multi-fidelity modeling via GP+: Categorical inputs t are mapped to latent points in the h−space while the source indicator variable s is mapped to a conditional distribution in z−space. Both mappings are achieved via deterministic and differentiable functions. Due to the probabilistic nature of z, multiple forward passes are required to obtain the final outputs of the model.

in the z−space. To more accurately quantify the epistemic uncertainties and model form errors, in this paper we develop a computationally efficient technique to learn probabilistic embeddings. While we can naturally obtain such embeddings within a Bayesian setting, we opt for a variational approach as it is computationally much more efficient.

To obtain the probabilistic latent representation of the categorical source indicator variable s, we reformulate $f_z(\boldsymbol{\pi}_s;\boldsymbol{\theta}_z)$ to obtain the conditional distribution $q(\boldsymbol{z} \mid s)$. To this end we use the reparameterization trick [97] and design $f_z(\boldsymbol{\pi}_s;\boldsymbol{\theta}_z)$ accordingly. Specifically, we model $q(\boldsymbol{z} \mid s)$ via a multi-variate normal distribution[13] that is fully characterized via the mean vector $\boldsymbol{\mu}_z$ and covariance matrix $\boldsymbol{\Sigma}_z = \boldsymbol{L}_z\boldsymbol{L}_z^T$ where $\boldsymbol{L}_z$ denotes the lower Cholesky decomposition of $\boldsymbol{\Sigma}_z$. $\boldsymbol{z}$ is then obtained by:

$$\boldsymbol{z} = \boldsymbol{\mu}_z + \boldsymbol{L}_z\boldsymbol{\epsilon} \tag{20}$$

where $\boldsymbol{\epsilon}$ is a dz dimensional vector whose elements are independent standard normal variables. $\boldsymbol{\mu}_z$ and $\boldsymbol{L}_z$ in Eq. (20) are obtained via the differentiable and deterministic function $f_z(\boldsymbol{\pi}_s;\boldsymbol{\theta}_z)$ which we choose to be a fully connected FFNN, see Fig. 4.

We note that GP+ by default only builds a probabilistic encoding for s and not $\boldsymbol{t}$ to avoid overfitting: since the number of categorical variables and their levels is typically much larger than the number of levels of s (which indicates the number of sources), probabilistically encoding $\boldsymbol{t}$ requires an FFNN with a large number of parameters and hence may result into overfitting especially if the training data is small. An alternative approach (which can be achieved in GP+ by changing its default parameters) is to encode a subset of $\boldsymbol{t}$ in a probabilistic latent space. We leave pursuing this direction to our future studies as it is application specific.

With a probabilistic fidelity embedding, we must sample from $q(\boldsymbol{z} \mid s)$ multiple times during both training and testing since even for fixed $s, \boldsymbol{t}$, and $\boldsymbol{x}$ the predicted covariance from the GP model varies due to $\boldsymbol{\epsilon}_z$ (the effect on the mean function depends on its formulation, see Section 4.3). Hence, for any fixed values of $s, \boldsymbol{t}, \boldsymbol{x}$, and model parameters (i.e., $\boldsymbol{\beta}, \boldsymbol{\theta}_h, \sigma^2$, and $\boldsymbol{\theta}_z$), we generate M samples from $q(\boldsymbol{z} \mid s)$[14] to build an ensemble or mixture of M GPs. Since $\boldsymbol{\epsilon}_z$ are independent and identically distributed (IID), each member of the GP ensemble is equally probable and we can extend Eq. (10) via the laws of total expectation and covariance [98,99]. To this end, we must obtain expressions for $\boldsymbol{m}$ and $\boldsymbol{C}$ in Eq. (8) during training. We distinguish among the GPs in the mixture model via the random variable I (whose probability mass function or PMF is $p(I = k) = \frac{1}{M}$ for $k = 1, \dots, M$) and we calculate the ensemble mean as:

$$\bar{m}(\boldsymbol{u}) = \mathbb{E}[m_k(\boldsymbol{u})] = \frac{1}{M}\sum_{k=1}^{M} m_k(\boldsymbol{u}) \tag{21}$$

where $m_k(\boldsymbol{u})$ is the expected value of the kth GP in the ensemble and correspondingly we denote $\bar{\boldsymbol{m}} = \frac{1}{M}\sum_{k=1}^{M}\boldsymbol{m}_k$ as the ensemble mean over the training data.

To obtain the ensemble expression for $\boldsymbol{C}$, we start by writing the covariance between the two random variables $y(\boldsymbol{u})$ and $y(\boldsymbol{u}')$ as:

$$\operatorname{cov}\left(y(\boldsymbol{u}), y(\boldsymbol{u}')\right) = \mathbb{E}\left[\operatorname{cov}\left(y(\boldsymbol{u}), y(\boldsymbol{u}')\right) | I\right] + \operatorname{cov}\left(\mathbb{E}[y(\boldsymbol{u})|I], \mathbb{E}[y(\boldsymbol{u}')|I]\right) \tag{22}$$

Given the PMF of I and Eq. (3b), we can calculate the first term on the right hand side of Eq. (22) as:

$$\mathbb{E}\left[\operatorname{cov}\left(y(\boldsymbol{u}), y(\boldsymbol{u}')\right)\middle| I\right] = \frac{1}{M}\sum_{k=1}^{M} c_k(\boldsymbol{u}, \boldsymbol{u}'; \sigma^2, \theta) = \bar{c}(\boldsymbol{u}, \boldsymbol{u}'; \sigma^2, \theta) \tag{23}$$

where the subscript k only affects the z−components in the kernel. For instance, $c_k(\boldsymbol{u}, \boldsymbol{u}'; \sigma^2, \boldsymbol{\omega}, \boldsymbol{\theta}_h, \boldsymbol{\theta}_z) = \sigma^2 \exp\left\{-\sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2 - \|\boldsymbol{h} - \boldsymbol{h}'\|_2^2 - \|\boldsymbol{z}_k - \boldsymbol{z}_k'\|_2^2\right\}$ for a Gaussian kernel.

We now turn to the second term on the right hand side of Eq. (22) and represent it as:

$$\operatorname{cov}\left(\mathbb{E}[y(\boldsymbol{u})|I], \mathbb{E}[y(\boldsymbol{u}')|I]\right) = \frac{1}{M}\sum_{k=1}^{M}\left(m_k(\boldsymbol{u}) - \bar{m}(\boldsymbol{u})\right)\left(m_k(\boldsymbol{u}') - \bar{m}(\boldsymbol{u})\right). \tag{24}$$

Inserting Eqs. (23) and (24) into Eq. (22) we obtain:

$$\operatorname{cov}\left(y(\boldsymbol{u}), y(\boldsymbol{u}')\right) = \bar{c}(\boldsymbol{u}, \boldsymbol{u}'; \sigma^2, \theta) + \frac{1}{M}\sum_{k=1}^{M}\left(m_k(\boldsymbol{u}) - \bar{m}(\boldsymbol{u})\right)\left(m_k(\boldsymbol{u}') - \bar{m}(\boldsymbol{u})\right) \tag{25}$$

which allows us to calculate the ensemble $\bar{\boldsymbol{C}}$ for Eq. (8) as:

$$\bar{\boldsymbol{C}} = \frac{1}{M}\sum_{k=1}^{M}\boldsymbol{C}_k + (\boldsymbol{m}_k - \bar{\boldsymbol{m}})(\boldsymbol{m}_k - \bar{\boldsymbol{m}})^T \tag{26}$$

where $\boldsymbol{C}_k$ denotes the covariance matrix of the kth ensemble member whose (i, j)th element is given by $c_k(\boldsymbol{u}^{(i)}, \boldsymbol{u}^{(j)}; \sigma^2, \boldsymbol{\theta})$ for $i, j = 1, \dots, n$. We use $\bar{\boldsymbol{C}}$ and $\bar{\boldsymbol{m}}$ while solving the optimization problem in Eq. (8).

For prediction, we take Q samples from the probabilistic fidelity embedding to determine the ensemble mean and variance:

$$\mathbb{E}[y(\boldsymbol{u}^*)] = \bar{\mu}(\boldsymbol{u}^*) = \frac{1}{Q}\sum_{k=1}^{Q}\mu_k(\boldsymbol{u}^*) \tag{27a}$$

$$\operatorname{cov}(y(\boldsymbol{u}^*), y(\boldsymbol{u}^*)) = \bar{\tau}^2(\boldsymbol{u}^*) = \frac{1}{Q}\sum_{k=1}^{Q}\left(\tau_k^2(\boldsymbol{u}^*) + \mu_k^2(\boldsymbol{u}^*)\right) - \bar{\mu}^2(\boldsymbol{u}^*) \tag{27b}$$

[13] Other distributions can also be used but we have had great success with simple ones such as the bivariate normal distribution when $dz = 2$.

[14] These samples are generated by drawing M random ϵ_z vectors of size dz from a standard multi-variate normal distribution.

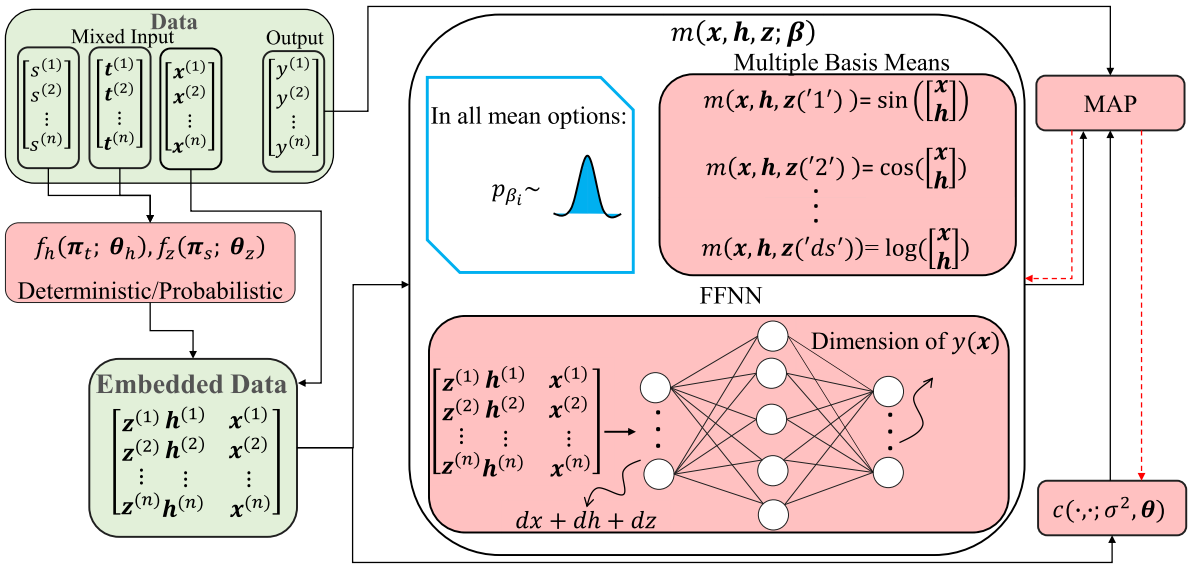


Fig. 5. Multi-fidelity modeling via mixed basis functions: Two generic options are defined in GP+ for building the mixed bases: (1) predetermined bases where multiple bases like polynomial, $sin(\cdot)$ and $cos(\cdot)$ can be defined for each data source, (2) FFNNs with user-defined architectures. All the parameters of the mean functions have a normal prior and are jointly learned through MAP.

where $\mu_k(\boldsymbol{u}^*)$ and $\tau_k^2(\boldsymbol{u}^*)$ are the mean and variance of the kth GP and obtained via Eq. (9). We note that the ensemble variance formula is a specific instance of the ensemble covariance given in Eq. (25). Additionally, we typically use $Q > M$ to reduce the training costs.

4.3. Gaussian processes with mixed basis functions

The parametric mean and covariance functions in Eq. (3) can be formulated in many ways. In this regard, most advancements have focused on designing the kernel (e.g., the ones we develop in Eqs. (13) and (25) or deep kernels [100,101]) since it significantly affects the performance of the resulting GP model. However, the mean function in Eq. (3a) plays an important role in many applications that involve, e.g., extrapolation [102], fusing multi-source data, or identifying model form errors.

Existing techniques typically leverage polynomials (in combination with other analytic functions such as $\sin(\cdot), \log(\cdot), \ldots$) or NNs in designing $m(\boldsymbol{x};\boldsymbol{\beta})$. In GP+, we extend these methods to seamlessly include the categorical variables in the mean function. Specifically, our idea is to feed the learnt representations of $\boldsymbol{t}$ and s into the mean function instead of the original categorical variables, i.e., we reparameterize $m(\boldsymbol{x},\boldsymbol{t},s;\boldsymbol{\beta})$ to $m(\boldsymbol{x},\boldsymbol{h},\boldsymbol{z};\boldsymbol{\beta})$, see Fig. 5. A major difference between our reparameterization and other alternatives (such as an NN whose inputs are one-hot encoded representation of $\boldsymbol{t}$ and s) is that our mean and covariance functions are directly coupled since the latent variables used in $m(\boldsymbol{x},\boldsymbol{h},\boldsymbol{z};\boldsymbol{\beta})$ are parameterized in the kernel, i.e., $\boldsymbol{h}=f_h(\boldsymbol{\pi}_t;\boldsymbol{\theta}_h)$ and $\boldsymbol{z}=f_z(\boldsymbol{\pi}_s;\boldsymbol{\theta}_z)$. Based on this idea, in GP+ we provide the following two options for modeling the mean function: (1) Having a global function that is shared among *all* combinations of $\boldsymbol{t}$ and s, and (2) Having mixed basis functions where a unique mean function is learnt for specific combinations of the categorical variables (e.g., in MF modeling, we can learn a unique mean function for each of the s data sources).

Fig. 5 illustrates two generic options that we define in GP+ for building mixed bases. The first option builds the mean function based on pre-determined bases that can include polynomials, $sin(\cdot), log(\cdot)$ or any other analytic functions. The second option is based on a fully connected FFNN whose architecture (e.g., number of hidden layers and their sizes) should be designed by the user. The size of the input layer of the NN depends on the dimensionality of $\boldsymbol{x}, \boldsymbol{h}$, and $\boldsymbol{z}$ while its output layer size depends on the response dimensionality (hence the output size is 1 for a single-response dataset).

Mixed bases are useful in applications where the input space has categorical features. The MF modeling approach described in Sections 4.1 and 4.2 is one such application as it requires adding the categorical variable s to the original input space. As shown in Section 5, using mixed bases improves MF modeling by allowing the fused GP model to emulate data source i with a unique mean function that better captures the global and local features of source i. To visualize this benefit, we consider the simple `Sinusoidal` example described in Appendix A.2 where 4 and 20 noisy data points from the HF and LF sources, respectively, are provided and the goal is to emulate both the HF and LF sources while inversely learning the model form error of the LF source. To investigate the effects of mixed bases, we build two GPs where the first one learns a single constant mean function for the fused data while the second one considers different mean functions for the two sources, namely, a zero mean for the HF source and a second-degree polynomial for the LF source.

The results are illustrated in Fig. 6 and indicate that the second GP emulates both sources better than the first GP in both interpolation and extrapolation. As it can be seen in Table 4 the true model form error is $0.3x^2-0.7x+1$ while the discovered one with the second GP is $0.2981x^2-0.7059x+0.9939$. We attribute the small differences between these two functions primarily to the fact that the training data is very small and noisy. As also shown in Fig. 6 we observe that the inclusion of the mixed bases affects the learnt encoding for s whose two levels are mapped to distant latent points in the first GP but close-by points in the second GP. This behavior indicates that the entire model form error in the second GP can be obtained by comparing the mean functions associated with the two sources.

In the above example, the true model form error is a 2nd degree polynomial and so we choose polynomial bases (of degree zero and two for the HF and LF sources, respectively) as the mean functions for the second GP. In practice, however, identification of the true model form error in realistic applications is much more challenging due to its unknown form, high dimensionality of the problem, lack of data, or noise. In these scenarios, we recommend using mean functions such as FFNNs that can adapt to the data and better model the global and local trends.

4.4. Inverse parameter learning for computer models

Most computer models are built to be applicable to a broad range of applications. Using these models in a specific context typically

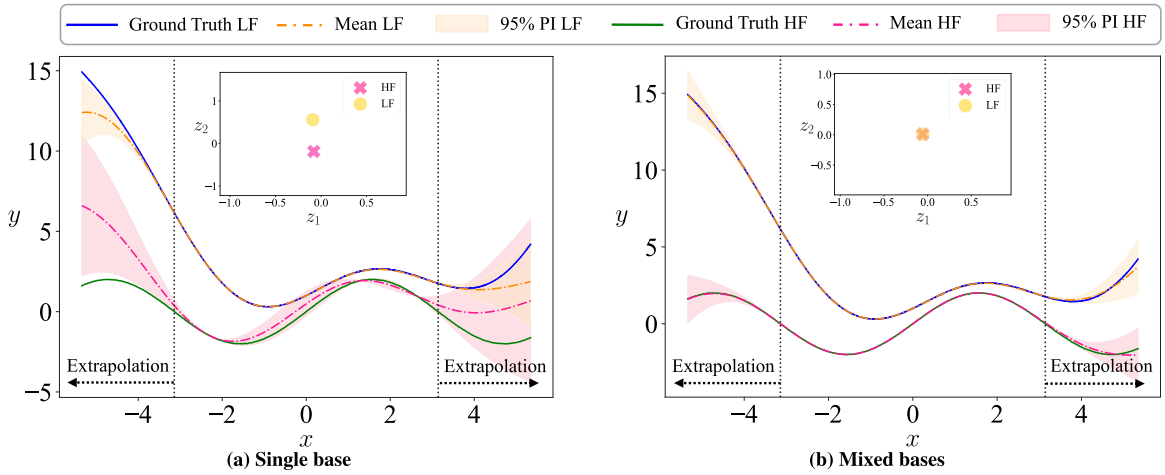


Fig. 6. Effect of mixed bases in MF emulation: The comparison is conducted on the 1-dimensional bi-fidelity problem where the LF source has a polynomial bias relative to the HF source, see `Sinusoidal` example in Table 4. Mixed bases **(b)** enables the emulator to better capture the local and global biases which, in turn, increases its interpolation and extrapolation powers. The learnt fidelity manifolds are shown as insets in each figure.

relies on adjusting their context-specific parameters using our domain knowledge or some measurements/observations. For instance, finite element (FE) models can be used to simulate the behavior of many materials under a wide range of loading conditions [103]. However, for a specific application such as modeling the thermoforming process of woven carbon fiber reinforced plastics [28,104,105], a specific FE model is needed. The material parameters such as directional Young's moduli and yarn-yarn friction coefficients in this FE model should be *calibrated* such that it can reproduce force–displacement curves obtained via experiments such as tension or three-point bending tests. Since such experimental data does not include the material parameters themselves, one must solve an inverse problem where the FE model's calibration parameters are estimated such that the model fits the experimental data. During this process, it is implicitly presumed that the response (e.g., force–displacement curve) is sufficiently sensitive to the calibration parameters (e.g., Young's moduli and yarn-yarn friction coefficients) as otherwise they cannot be accurately estimated.

Calibration of a computer model is tightly connected to that model's bias with respect to an HF data source (e.g., experiments). To explain this connection, we note that most computer models suffer from systematic errors that arise from, e.g., their missing physics, the simplifying assumptions made during their development, or numerical errors. To mitigate the effect of these errors, computer models sometimes include a few additional calibration or tuning parameters that may not even correspond to any physical properties of the system. One example is the artificial viscosity parameter that is used to stabilize explicit solution methods that are needed when modeling dynamic processes such as fracture via the FE method. A related example is the calibration of physics-based reduced-order models (ROMs) [27,106–110] that simplify expensive computer models (such as direct numerical simulations or DNS) to gain computational speedups. Such simplifications introduce some bias into the ROMs whose effects are typically mitigated by calibrating material parameters such that a ROM can reproduce small HF data obtained from DNS. That is, even if the material parameters are known, one may have to adjust them for ROMs.

Calibration of computer models is closely related to MF modeling since it requires fusing multiple datasets that typically have different levels of fidelity (e.g., fusing simulations with experiments or observations). Hence, we extend the capabilities introduced in Sections 4.1 and 4.2 to accommodate inverse parameter learning for computer models. For this extension, we consider two application scenarios:

- Simultaneous calibration of multiple (>1) computer models: We presume that the calibration parameters of these models correspond to some unobserved characteristics of a system. We make this assumption since it is not optimal to jointly calibrate the *tuning* parameters of different models that are added to them for reasons besides characterizing unobserved features of a system (note also that the number of these tuning parameters generally varies across different models that simulate the same system).
- Calibration of a single computer model: We do not distinguish between the calibration parameters regardless of whether they are merely tuning knobs or they correspond to some unobservable features.

We highlight that in both scenarios we can use multiple HF data sets in `GP+` as long as they correspond to the same physical system (e.g., obtaining force–displacement curves via different universal testing machines that have different levels of fidelity), see Fig. 7.

Following the notation of previous sections, we denote the quantitative inputs by $\boldsymbol{x}$ and the latent representations of qualitative inputs and the categorical source indicator variable by $\boldsymbol{h}$ and $\boldsymbol{z}$, respectively. These inputs are shared across all the data sources but, as described above, the LF sources have additional quantitative inputs that correspond to the calibration parameters and are denoted by $\boldsymbol{\zeta} = [\zeta_1, \zeta_2, \dots, \zeta_{d_\zeta}]^T$. The "best" calibration parameters ($\boldsymbol{\zeta}^*$) should be estimated using the HF data to accurately characterize the physical system. We denote these estimates by $\widehat{\boldsymbol{\zeta}}$ and modify the correlation function to obtain them. For instance, the Gaussian and Matérn correlation functions are reformulated as follows:

$$r(\boldsymbol{u},\boldsymbol{u}';\boldsymbol{\omega},\boldsymbol{\theta}_h,\boldsymbol{\theta}_z) = \exp\left\{ -\sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2 \; -\sum_{i=1}^{dh}(h_i - h_i')^2 - \sum_{i=1}^{dz}(z_i - z_i')^2 - \sum_{i=1}^{d\zeta} 10^{\omega_{i+dx}}(\zeta_i - \zeta_i')^2 \right\} \tag{28a}$$

$$r(\boldsymbol{u},\boldsymbol{u}';\boldsymbol{\omega},\boldsymbol{\theta}_h,\boldsymbol{\theta}_z) = \frac{2^{1-\nu}}{\Gamma(\nu)} K_\nu \left(\sqrt{2\nu} \times \sum_{i=1}^{dx} 10^{\omega_i}(x_i - x_i')^2 + \sum_{i=1}^{dh}(h_i - h_i')^2 + \sum_{i=1}^{dz}(z_i - z_i')^2 + \sum_{i=1}^{d\zeta} 10^{\omega_{i+dx}}(\zeta_i - \zeta_i')^2 \right)^{1+\nu} \tag{28b}$$

where $\boldsymbol{u} = [\boldsymbol{x}, \boldsymbol{t}, s, \boldsymbol{\zeta}]^T$ and $\boldsymbol{\omega}$, $\boldsymbol{\theta}_h$ and $\boldsymbol{\theta}_z$ are defined as before. While training the model, the correlation between LF samples can be readily calculated via Eq. (28). However, if at least one of the samples is an HF one, in the last term of Eqs. (28a) and (28b) we use $\widehat{\zeta}_i$ which are estimated jointly with all the other parameters of the model via MAP.

Similar to Section 4.2, we can inversely learn the calibration parameters within a probabilistic setting to more accurately quantify the

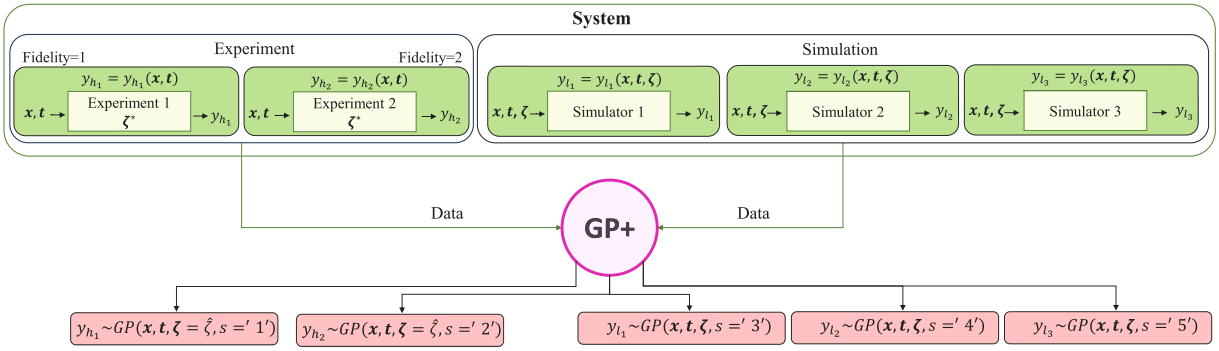


Fig. 7. Inverse parameter calibration via data fusion: GP+ builds a fused model that jointly emulates all the sources while inversely estimating the calibration parameters of the HF sources. It is assumed that (1) all the data sources share the same inputs, (2) the HF data sets correspond to the same underlying system, and (3) the calibration parameters represent some properties of the HF sources.

uncertainties compared to the deterministic counterpart based on the MAP. Following Section 4.2, we learn the calibration parameters within a variational framework by formulating their joint posterior with a multivariate normal distribution that is fully characterized by its mean vector and covariance matrix. We learn the parameters of this joint distribution via the reparametrization trick:

$$\zeta_i = \mu_{\zeta i} + \tau_{\zeta i}\epsilon_{\zeta i}, \tag{29}$$

where $\epsilon_{\zeta i} \sim \mathcal{N}(0,1)$ is an auxiliary noise variable while $\mu_{\zeta i}$ and $\tau_{\zeta i}$ parameterize the posterior distribution of ζi. During both the training and prediction phases, we draw samples for calibration parameters. As a result, the values of $\boldsymbol{\zeta}$ fluctuate in each optimization iteration which consequently changes the covariance and mean functions of the emulator. To efficiently consider these variations, we follow our method used for probabilistic manifold modeling (see Section 4.2) and employ ensembling to calculate both the mean vector and covariance matrix.

We highlight that in both deterministic and probabilistic calibration cases, the estimated calibration parameters and the learnt bias are tightly connected in that the former depends on what bias form has been chosen. Most existing methods [18,19,22,89,90,111–115] first assume a specific functional form (e.g., a GP or a polynomial) for the bias and the relation between the LF and HF sources (see Eq. (14) and Fig. 3 for one example). Then, given data from both LF and HF sources, they estimate the calibration parameters and the parameters of the bias function. For these approaches, the estimated calibration parameters are strongly dependent on the *assumed* form of the bias function and how it relates the HF and LF sources. If these assumptions are incorrect, the calibration results will be misleading. In GP+ we significantly relax these assumptions. Specifically, we (1) do not assume the bias term is additive, and (2) simultaneously calibrate multiple sources (rather than just calibrating one source at a time). Therefore, we do not eliminate the so-called identifiability issue but provide the means that analysts can use to address it depending on the application. For instance, an effective way to reduce identifiability issues is using multiple-response data during calibration [18,19,114,116]. Similarly, the multi-source calibration mechanism in GP+ provides the calibration process with more information and hence has the potential to reduce non-identifiability. Additionally, in GP+ we can use mixed basis functions which can help analysts in choosing appropriate mean functions for each source and study the effects of this choice on the estimated calibration parameters, accuracy on unseen data, and learnt bias functions (note that the difference between two mean functions essentially gives the global bias between the corresponding sources, see Fig. 6).

5. Functionalities of GP+ and comparative studies

In this section, we demonstrate the core functionalities of GP+ and compare them against some of the widely used methods or open-source GP modeling packages. We start with emulation and MF modeling in Sections 5.1 and 5.2, respectively, where we also study the potential benefits of using a probabilistic embedding instead of a deterministic one in GP+ in Section 5.2.1. Then, in Section 5.3 we conduct a few carefully designed studies to evaluate the capabilities of GP+ in inverse parameter estimation. Finally, in Section 5.4 we assess the performance of GP+ in BO which is a many-query outer-loop application where GPs are dominantly used for emulation.

Throughout this section, we use normalized root mean squared error (NRMSE) and normalized interval score (NIS) for assessing the accuracy of, respectively, the mean values and prediction intervals provided by a GP:

$$NRMSE = \frac{1}{std(\boldsymbol{y})}\sqrt{\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}(y^{(i)} - \mu^{(i)})^2} \tag{30}$$

$$\begin{aligned} NIS = \frac{1}{std(\boldsymbol{y})}\Big(\frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}}(\mathcal{U}^{(i)} - \mathcal{L}^{(i)}) + \frac{2}{\upsilon}(\mathcal{L}^{(i)} - y^{(i)})\mathbb{1}\{y^{(i)} < \mathcal{L}^{(i)}\} \\ + \frac{2}{\upsilon}(y^{(i)} - \mathcal{U}^{(i)})\mathbb{1}\{y^{(i)} > \mathcal{U}^{(i)}\}\Big) \end{aligned} \tag{31}$$

where $y^{(i)} = y(\boldsymbol{u}^{(i)})$ and $\mu^{(i)} = \mu(\boldsymbol{u}^{(i)})$ denote the output and predicted mean of test sample $\boldsymbol{u}^{(i)}$, respectively, and $std(\boldsymbol{y}) = std(y(\boldsymbol{U}))$ shows the standard deviation of the test samples obtained at $\boldsymbol{U}$. $\mathcal{U}^{(i)}$ and $\mathcal{L}^{(i)}$ are upper and lower endpoints of the prediction interval for the ith test sample. These endpoints are the predictive quantiles at levels $\upsilon/2$ and $1-\upsilon/2$, respectively. We use 95% prediction interval ($\upsilon = 0.05$) and hence these endpoints are defined as $\mathcal{U}^{(i)} = \mu^{(i)} + 1.96\tau^{(i)}$ and $\mathcal{L}^{(i)} = \mu^{(i)} - 1.96\tau^{(i)}$. $\mathbb{1}\{\cdot\}$ is an indicator function which is 1 if its condition holds and zero otherwise. For both metrics in Eqs. (30) and (31) lower values indicate more accuracy.

Unless otherwise stated, we use the default settings of GP+ which are detailed in Tables 9 and 10. For instance, the prior distributions of the parameters in all the examples are $\omega_i \sim N(-3,3)$, $\beta \sim N(0,1)$, $A_{ij} \sim N(0,1)$, $\sigma^2 \sim LN(0,1)$,[15] and $\delta_i \sim LHS(0,0.01)$[16] [117].

5.1. Emulation

We use three analytic and two engineering problems to compare the performance of GP+ against the GP emulation capabilities of GPytorch,[17] MATLAB,[18] BoTorch, and SMT2 [30,118–120]. These problems

[15] Log-Normal.

[16] Log-Half-Horseshoe with zero lower bound and scale parameter 0.01.

[17] GPyTorch uses the Adam optimizer by default and to improve its performance we use a learning rate scheduler during the training process. To avoid convergence to local optima, we repeat the optimization process 64 times, each with a different initialization for the model's parameters.

[18] We use automatic relevance determination (ARD) squared exponential kernel for GP emulation in MATLAB.

```python
from gpplus.models.GP_Plus import GP_Plus
from gpplus.test_functions.physical import borehole_mixed_variables
from gpplus.preprocessing import train_test_split_normalizeX
from gpplus.utils import set_seed

set_seed(4)                                                          # Set random seed for reproducibility
qual_dict = {0: 5, 5: 5}                                             # Categorical input indices
U, y = borehole_mixed_variables(n=10000, qual_dict=qual_dict, random_state=4)   # Generate data
Utrain, Utest, ytrain, ytest \
    = train_test_split_normalizeX(U, y, test_size=0.99, qual_dict=qual_dict)    # Split and normalize data

model = GP_Plus(Utrain, ytrain, qual_dict=qual_dict)                 # Initialize GP_Plus model
_ = model.fit()                                                      # Train model
model.evaluation(Utest, ytest)                                       # Evaluate model performance
```

Fig. 8. Emulation via GP+: We emulate the `Borehole-Mixed` function by importing the necessary modules, identifying the indices of the categorical variables, generating data, initializing the GP+ model, and finally training and evaluating the model.

are briefly described below (see details in Appendix A) and they cover a range of input dimensionality and characteristics (e.g., `HOIP` only has categorical inputs). To provide a comprehensive analysis, we illustrate the emulation comparisons in this section and discuss the computational costs of these baselines in Appendix I.

As shown in Fig. 8, emulation via GP+ is achieved via a few lines of code regardless of whether the problem has categorical variables or not. As opposed to GP+, MATLAB handles categorical variables by first one-hot encoding them and then treating the resultant variables as numerical. BoTorch leverages mixed single-task GP (MST-GP) which defines two distinct kernels for numerical and categorical features. Specifically, MST-GP uses the Matèrn kernel for the numerical features while for the categorical features it calculates the exponential of their normalized binary distance which is 0 when the two categorical variables are the same. The final kernel of MST-GP is the combination of the categorical and numerical parts (see Appendix D for details). Since MST-GP is specifically developed to handle mixed input spaces that have both categorical and numerical features, we use GPytorch in problems with only quantitative features (note that Gpytorch cannot handle categorical inputs). SMT2 uses a Gaussian Kernel for problems involving solely numerical inputs which is multiplied by a categorical kernel in case the input space has qualitative features. The options for categorical kernel provided in SMT2 are homoscedastic hypersphere (SMT2_{HH}) [121], exponential homoscedastic hypersphere (SMT2_{EHH}) [122], and Gower distance-based correlation kernels ($\text{SMT2}_{\text{Gower}}$) [123].

`Wing` and `Borehole` are two single-response analytic examples whose input space only has quantitative features. The dimensionality of the input space for `Wing` and `Borehole` is 10 and 8, respectively. To compare the performance of the three methods in mixed input spaces, we convert the first and sixth features of `Borehole` to categorical variables with 5 distinct levels. This analytic example is referred to as `Borehole-Mixed` (see Appendix A.2 for further details). We generate 10 000 samples from the HF source in Table 4 and use 1% of the data for training and the rest for testing. `HOIP` and `NTA` are both 3-dimensional problems and only have categorical inputs (see Appendix A.1 for more details). We use 150 and 100 HF samples from `HOIP` and `NTA`, respectively, for emulation and the rest of the HF data for testing. For all problems, we repeat the emulation process 10 times and report the average values for each metric to ensure the metrics are robust to random initialization.

Emulation results are summarized in Table 1 which demonstrate that GP+ more accurately predicts the responses in all examples and provides more reliable prediction intervals. In the case of `Wing` or `Borehole` which only have quantitative features, we attribute the superiority of GP+ primarily to the parameter optimization process. Specifically, we use MAP (as opposed to maximum likelihood estimation or MAE), search for the length-scale parameters (i.e., $\boldsymbol{\omega}$ in the correlation function) in the log scale, and leverage L-BFGS-B.[19] for optimization. These choices smooth the profile of the objective function and accelerate the convergence. The comparable performance of SMT2 and GP+ in terms of NRMSE is attributed to SMT2's usage of the so-called profiling technique [124] for parameter estimation.

GP+ outperforms all baselines in `Borehole-Mixed`, `HOIP`, and `NTA` which have categorical variables. In these problems, GP+ explicitly learns the relations between different categorical variables and their levels which not only improve the emulation performance, but also provide visually interpretable embeddings (see Figs. 16(a) to 16(c) in Section 5.4 for an example). We note GP+ estimate more parameters than other methods in Table 1 since it directly learns the correlations among categorical variables. This approach results in a more expensive and challenging optimization process which can converge to suboptimal solutions if the training data is very small and the categorical variables have many levels. To mitigate potential overfitting issues in such cases, we recommend using tighter priors in $f_h(\boldsymbol{\pi}_t, \boldsymbol{\theta}_h)$. Correspondingly, in `NTA` and `HOIP` with 240 and 480 distinct categorical combinations, respectively, we use $\mathcal{N}(0, 0.1)$ and $\mathcal{N}(0, 0.01)$ priors for $f_h(\boldsymbol{\pi}_t, \boldsymbol{\theta}_h)$.

[19] Limited-memory Broyden–Fletcher–Goldfarb–Shanno that considers simple bounds on the variables.

Table 1
Comparison of emulation accuracy: We test the performance of GP+ in emulation against GPyTorch, Matlab, and MST-GP on five examples. The reported NRMSE and NIS are for unseen data and averaged across 10 repetitions.

Model	`Wing`		`Borehole`		`Borehole-Mixed`		`HOIP`		`NTA`	
	NRMSE	NIS	NRMSE	NIS	NRMSE	NIS	NRMSE	NIS	NRMSE	NIS
GP+	0.0010	**0.0049**	**0.0008**	**0.0045**	**0.0023**	**0.0139**	**0.490**	**2.9563**	**0.2950**	**2.2128**
MATLAB	0.0045	0.0447	0.0033	0.0430	0.0042	0.0464	0.5404	4.198	0.4043	2.6360
MST-GP	–	–	–	–	0.0062	0.0338	0.515	3.940	1.38	2.499
GPytorch	0.0081	0.0715	0.0078	0.0641	–	–	–	–	–	–
SMT2	**0.0009**	0.0052	**0.0008**	0.0054	–	–	–	–	–	–
$\text{SMT2}_{\text{Gower}}$	–	–	–	–	0.0034	0.0331	0.526	3.5468	0.3939	2.5104
SMT2_{HH}	–	–	–	–	0.0159	0.2899	0.8912	10.8922	1.4979	9.6483
SMT2_{EHH}	–	–	–	–	0.0126	0.2203	3.2678	21.9938	1.8296	8.9458

Table 2
Multi-fidelity emulation: We test the performance of GP+ in various settings against V-GP, STMF-GP, and FFNN across three examples and report NRMSE and NIS on unseen HF data.

Model	Option	Sinusoidal		Wing		DNS-ROM	
		NRMSE	NIS	NRMSE	NIS	NRMSE	NIS
GP+	Single constant as $m(\boldsymbol{u};\boldsymbol{\beta})$	0.2501	1.2070	0.0743	0.4294	0.1572	0.9051
	Multiple constants as $m(\boldsymbol{u};\boldsymbol{\beta})$	0.2201	0.9274	**0.0729**	0.4419	0.1560	0.8935
	Small FFNN as $m(\boldsymbol{u};\boldsymbol{\beta})$	**0.1999**	0.8261	0.0751	**0.3884**	0.1535	0.8561
	Medium FFNN as $m(\boldsymbol{u};\boldsymbol{\beta})$	0.2062	**0.5475**	0.0751	0.4072	**0.1528**	**0.8477**
V-GP	–	0.4156	1.9842	0.1794	0.9152	0.2101	1.0856
FFNN	Small	0.8076	–	0.6295	–	0.2693	–
	Medium	0.6238	–	0.4320	–	0.2297	–
	Large	0.5244	–	0.3543	–	0.2221	–
STMF-GP	$STMF-GP_1$	0.4835	6.7312	0.1219	1.0125	0.1618	1.0625
	$STMF-GP_2$	0.5362	8.5698	0.2001	1.1661	0.1707	0.9651

5.2. Multi-fidelity modeling

In this section, we assess the performance of GP+ in MF emulation by comparing it against widely used emulators. Our baselines include vanilla GPs trained only on the HF data (V-GP), FFNNs, and single-task multi-fidelity GPs (STMF-GPs) introduced by BoTorch (detailed in Appendix E). Furthermore, we examine different versions of GP+ with distinct basis functions explained in Section 4.3. The mean function in these versions are formulated as a single constant, multiple constants (the number of constants is $ds-1$, as we consider zero mean for HF source), and finally an FFNN.

Since the performance of FFNNs is sensitive to their architecture, we design small, medium, and large networks and for each network size test many different scenarios and report the results of the most accurate ones (see Appendix F for details). In the case of STMF-GP, as detailed in Appendix E the fidelity indices are numerical and must reflect the relative accuracy of the data sources. STMF-GP lacks a built-in metric for determining these indices and relies on the user to provide these values. To address this issue, we first leverage the learnt embedding (i.e., the z–space) of GP+ to find the order of these indices and then assign two different sets of values to them to assess this method's sensitivity to the assigned values. These values are outlined in Table 11 and we denote the corresponding models by $STMF-GP_1$ and $STMF-GP_2$.

We use two analytic (Sinusoidal and Wing) and one engineering (DNS-ROM) examples for the comparison (see Table 4 and [125] for details on these examples). Sinusoidal is a 1-dimensional, bi-fidelity example for which we generate a dataset consisting of 400 HF and 2000 LF samples. Wing has 4 fidelity sources (1 HF and 3 LFs) and we produce 1500 samples from the HF source and 4000 samples from each of the LF sources. In both analytic examples, 1% of data is used for training and the rest of the HF data for testing. DNS-ROM is a 5-dimensional problem on fracture modeling of metallic alloys where the data are generated via four different simulators with $70, 110, 170, 250$ samples. In this example, we use 20% of the samples for training and the rest of the HF data for testing.

The results for each approach on each problem are summarized in Table 2 and demonstrate that GP+ significantly outperforms the other baselines in all problems. More specifically, while V-GP is limited to the small HF data, GP+ effectively leverages the information provided by the LF data to learn the HF source. The poor performance of STMF-GP is due to the fact that from a methodological standpoint it models the inter-relations between the data sources incorrectly. In addition to providing low accuracy, the predictions of STMF-GP are sensitive to the values assigned to its fidelity indices. This is evident in Table 2 where the prediction errors for two different yet close sets of random indices ($STMF-GP_1$ and $STMF-GP_2$, see also Table 11) are very different. Regarding FFNNs, we attribute their poor performance in all problems to their architecture and, in particular, their simple mechanism for handling fidelity levels. These FFNNs simply one-hot encode the fidelity indices and ignore the intricate correlations among the corresponding data sources. Compared to other methods, the reported NRMSEs for FFNNs are more sensitive to the model architecture and notably change as the network size varies. This sensitivity is partly due to the small size of the MF data and can perhaps be improved by iteratively refining the architecture or the optimization parameters (e.g., learning rate schedule or regularization weights). However, we avoid such detailed refinements since none of the other methods are fine-tuned.

Comparing the results of different versions of GP+ reveals that in all cases using mixed bases improves MF modeling by better capturing the global and local features of each source (compare the first row to other mean functions). This choice benefits NIS slightly more than NRMSE since the former metric relies on both the expected value and estimated variance, i.e., $\tau^2(\boldsymbol{u}^{*(i)})$. For instance, GP+ with multiple constants as $m(\boldsymbol{u};\boldsymbol{\beta})$ and medium FFNN $m(\boldsymbol{u};\boldsymbol{\beta})$ achieve similar NRMSEs (0.2201 vs. 0.2062), but their NIS significantly differs (0.9274 vs. 0.5475) in Sinusoidal.

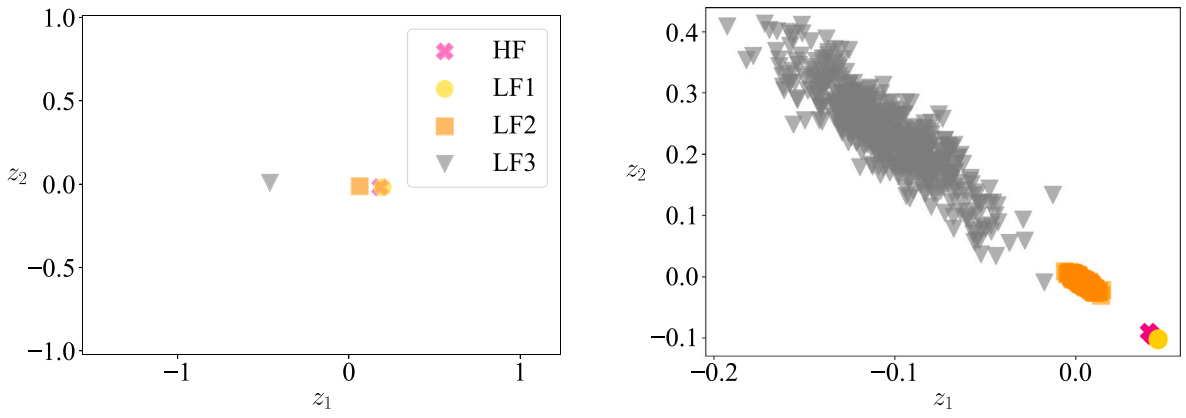
Wing and DNS-ROM are relatively complex problems with small amounts of data and different types of noise (e.g., inDNS-ROM the noise variance depends on the source while in Wing it does not). GP+ is very well suited to tackle these types of problems because the number of its hyperparameters scales much better than FFNNs, is not limited to the small HF data, and better estimates noise as explained in Section 4.1.1. Accordingly, we observe lower prediction errors for GP+ in these examples.

5.2.1. Deterministic and probabilistic embedding

As explained in Sections 4.1 and 4.2, one of the distinctive features of GP+ is its ability to learn both probabilistic and deterministic embeddings for MF modeling. We revisit the Wing example with smaller datasets to evaluate GP+'s efficacy in data-scarce scenarios. Specifically, we generate 1000 samples from the HF source and 2000 samples from each LF source and use 1% for training and the rest for testing. Throughout this section, $f_z(\boldsymbol{\pi}_s;\boldsymbol{\theta}_z))$ is an FFNN with a single five-neuron hidden layer and we use the multiple constants option of GP+ to model the mean function of the GPs.[20] Similar to the previous sections, we repeat both deterministic and probabilistic simulations 10 times.

Fig. 9 illustrates the fidelity embeddings learned through probabilistic and deterministic MF modeling. We only show the learnt embeddings in one of the repetitions since the relative distances across the 10 repetitions are quite similar (albeit the exact locations are different). As explained in Section 4, these embeddings indicate how similar or correlated different data sources are with respect to each other. Specifically, the latent distance between the points encoding the LF sources from the point that encodes the HF source are consistent

[20] With this option, a constant is learnt for each of the LF sources since the data is generated by four sources in Wing and 0 is used for the HF source.



(a) Deterministic Embedding **(b) Probabilistic Embedding**

Fig. 9. Probabilistic vs. deterministic embedding for MF modeling: While both embeddings estimate the same degree of similarity among sources, the probabilistic one characterizes more uncertainties.

with the NRMSE values reported in Table 4 where, e.g., the furthest encoded source (LF3) has the largest NRMSE (5.57). In addition, we observe in Fig. 9 that while the general trends are the same across the two embeddings (e.g., LF3 and LF1 are the furthest and closest to HF), probabilistic embedding more accurately quantifies model form uncertainties especially in the case of highly biased LF sources.

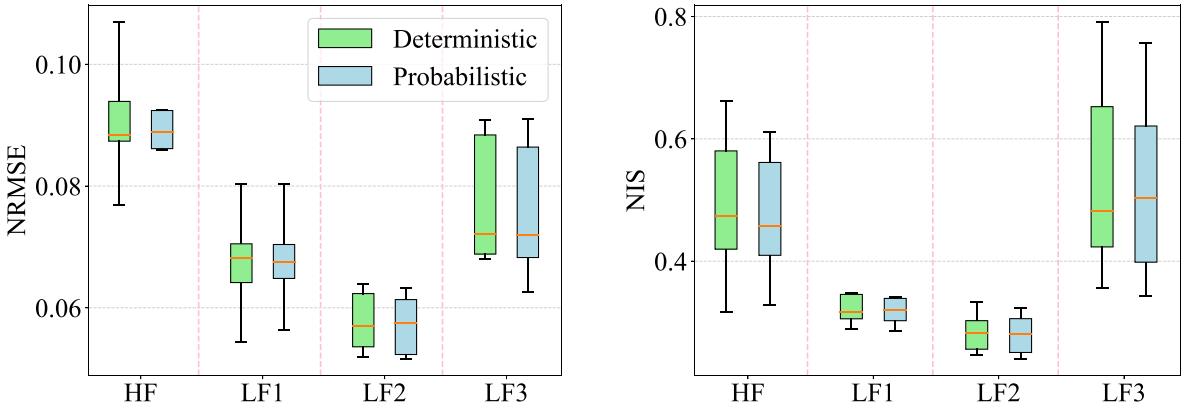
Fig. 10 compares the prediction accuracy on unseen data with probabilistic and deterministic embeddings. As it can be observed, the probabilistic approach is slightly more robust in HF emulation but both approaches (1) provide the same degree of accuracy and robustness for LF sources, and (2) are less accurate in emulating HF and LF3 sources. The reason LF is learnt less accurately than other LF sources is its low correlation while the errors in emulating the HF source primarily stem from the lack of HF data.

5.3. Inverse parameter estimation

In this section, we compare the performance of `GP+` in inverse parameter estimation against the method of KOH and the open-source software package `UQLab` [126]. As schematically demonstrated in Fig. 3**(a)**, KOH uses an additive bias and can calibrate a single LF source at a time. In our studies, we consider two versions of this method where its parameters (e.g., the kernel parameters of LF and HF sources) are estimated either jointly or via a modular scheme [28,91,113] which first estimates the parameters of the LF source and then optimizes the rest of the parameters. The calibration module of `UQLab` relies on Bayesian inference and leverages MCMC for parameter estimation. Similar to previous sections, we repeat each of our studies 10 times and report the average values.

As detailed in Appendix A.2, `beam deflection` is a 5-dimensional bi-fidelity example where the objective is to infer a beam's Young's modulus (ζ) whose ground truth value is 30 GPa. This example is directly taken from the documentation of `UQLab` where there is only one HF data point and the difference between the LF and HF sources is a zero-mean noise. To explore the effects of prior distributions on the results, we assign three different priors to ζ while comparing `GP+` with `UQLab`. While 200 LF samples are used in `GP+`, `UQLab` leverages the analytic form of the LF model in MCMC.

The results presented in Table 3 indicate that both methods estimate similar posterior means for the calibration parameter if the prior is relatively precise. However, the two methods behave quite differently as the prior mean is shifted away from the ground truth value. Specifically, `UQLab` provides posterior means that are quite close to the prior means while `GP+` is significantly less sensitive to imprecise priors. In the case of `GP+`, we note that the deterministic approach does not provide uncertainties and the reported standard deviations for the probabilistic approach are very small. These tight posteriors are expected since the model has access to sufficient LF data and the bias between the sources is a Gaussian noise.



(a) NRMSE **(b) NIS**

Fig. 10. Prediction performance with probabilistic vs. deterministic embedding: We report NRMSE and NIS values calculated across 10 repetitions. Both approaches are very accurate but the probabilistic one is more robust for HF emulation especially since HF data is very small. The similar NRMS and NIS values for LF1 and LF2 are expected because of their high correlation.

Table 3
Inverse estimation of Young's Modulus using GP+ and UQLab: The posterior mean and standard deviations provided by UQLab is very sensitive to the assigned prior while GP+ does not suffer from this issue. The reported uncertainties by GP+ are small since the bias between LF and HF sources is simple and the model has access to sufficient LF data.

Prior on ζ	Method	Estimated mean (GPa)	Estimated std (GPa)
$\mathcal{N}(30,5)$	UQLab	**30.0101**	4.6832
	Probabilistic GP+	29.2363	15.867 10^{-6}
	Deterministic GP+	29.2304	–
$\mathcal{N}(25,5)$	UQLab	26.1540	4.3772
	Probabilistic GP+	29.0311	29.320 10^{-6}
	Deterministic GP+	**29.0449**	–
$\mathcal{N}(20,5)$	UQLab	22.2348	4.3355
	Probabilistic GP+	**29.1462**	10.3699 10^{-5}
	Deterministic GP+	28.9418	–

We now use the high-dimensional Borehole problem presented in Table 5 with 1 HF and 2 LF sources that both have nonlinear model form errors (note that LF1 is more biased compared to LF2, see the NRMSEs in Table 5). We generate 20 and 100 samples from, respectively, the HF and each of the LF sources. We only corrupt the HF training samples with noise and use 1800 noise-free HF samples for evaluation of models' performance. There are two calibration parameters in this example and their ground truth values are $\zeta_1 = 250$ and $\zeta_2 = 1500$. We compare the calibration results obtained by GP+ with those from KOH whose parameters are estimated either jointly or by a modular approach. Since KOH can only fuse two sources at a time, we consider different combinations of LF and HF sources in our experiments.

As shown in Fig. 11, both configurations of GP+ convincingly outperform KOH's approach in terms of both NRMSE and NIS. We attribute this superior performance to three key factors: (1) GP+'s ability to utilize all data simultaneously while KOH's approach can only work with two sources at a time (expectedly, modular KOH with LF1 has the least accuracy), (2) GP+'s capability to capture nonlinear correlations whereas KOH is confined to learning additive model form errors, and (3) GP+ leverages a more stable and regularized training procedure. We note that probabilistic GP+ outperforms its deterministic counterpart as it learns a posterior distribution for ζ rather than just a single point estimate. While probabilistic calibration improves the performance on average, it shows more variability across the 10 repetitions primarily due to the fact that it has more parameters than the deterministic one.

The superiority of GP+ in emulation is coupled with a more accurate estimation of the calibration parameters as illustrated in Fig. 12. While the estimated values are quite accurate across all methods, probabilistic GP+ provides slightly better results, especially compared to modular KOH that fuses the data in a sequence of steps rather than jointly. We also observe that estimations for ζ_1 in Fig. 12(a) are generally more accurate than those of ζ_2 in Fig. 12(b). This trend is primarily due to the fact that the underlying functions (i.e., HF and both LF sources)

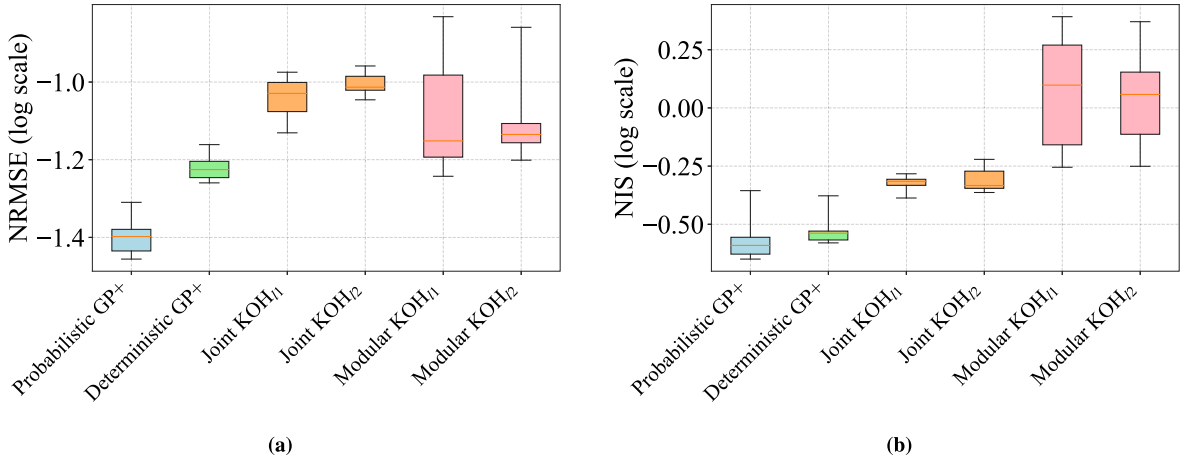


Fig. 11. High-fidelity emulation performance in Borehole: GP+ consistently outperforms KOH's approach. The superior performance of both variations of GP+ are primarily attributed to using all the data jointly and dispensing with the assumption that model form errors are additive.

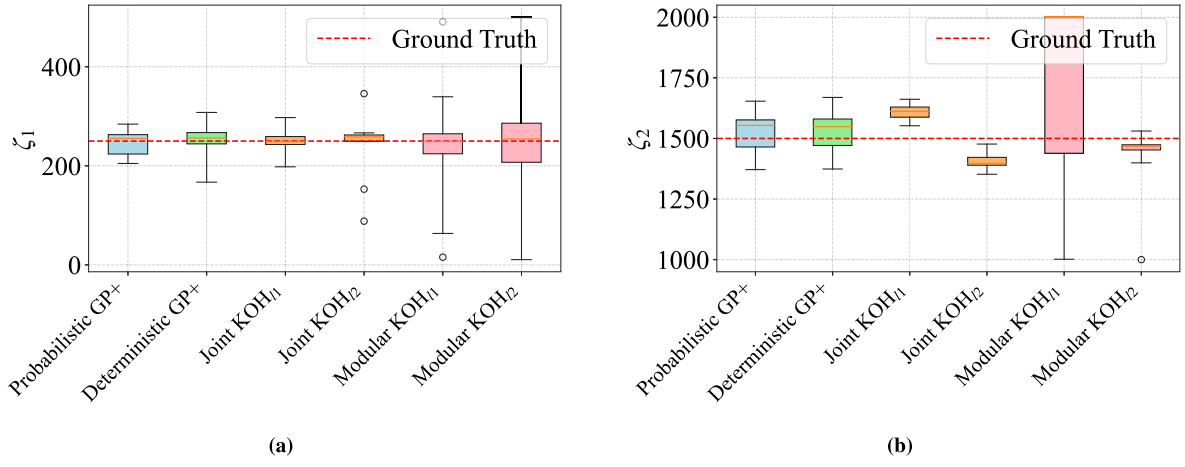


Fig. 12. Calibration performance in Borehole: Ground truth values are the numbers used to generate the HF samples. While all methods estimate ζ quite well, probabilistic GP+ performs the best.

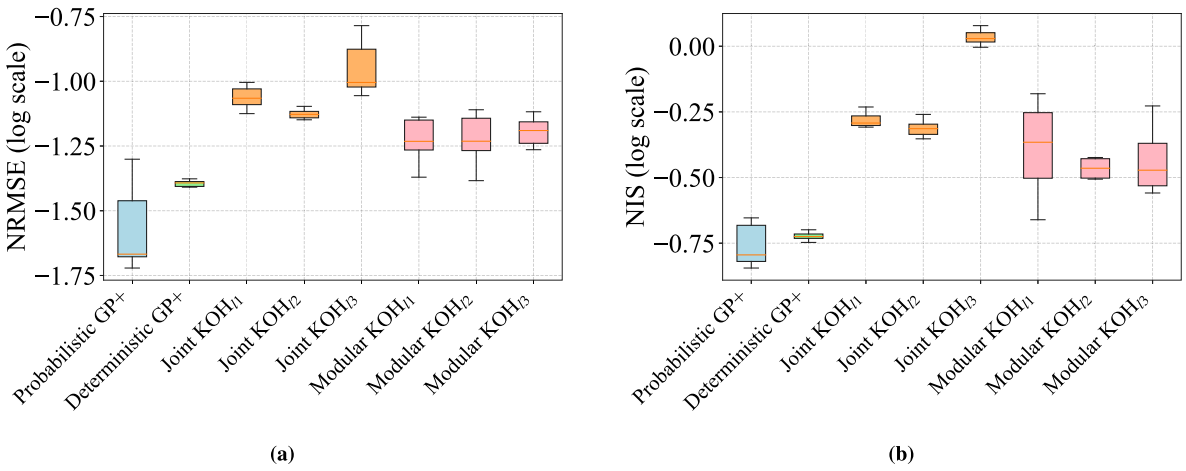


Fig. 13. High-fidelity emulation performance in `Wing`: GP+ strategies outperform KOH's approach which can only fuse two sources at a time and uses an additive formulation for the model form errors.

are more sensitive to ζ_1 than ζ_2 (see Table 6 for sensitivity analysis of `Borehole`).

We now revisit a variation of the `Wing` problem which has four calibration parameters whose ground truth values are $\boldsymbol{\zeta}^{*T} = [40, 0.85, 0.17, 3]$ (these values correspond to the numbers used in the HF source). As detailed in Table 5, there are four data sources in `Wing` where LF1 and LF3 are the most and least accurate LF sources, respectively. We take a small number of samples from each of the four sources and corrupt all the data with noise. Similar to the previous study, we compare probabilistic and deterministic calibration capabilities of GP+ against two versions of KOH. Throughout, we repeat the simulations 10 times and use 2500 noise-free HF samples for testing the emulation performance.

Fig. 13 compares the performance of GP+ in predicting unseen HF data against both the modular and joint variations of KOH's method. As it can be observed, the probabilistic GP+ achieves the lowest NRMSE as well as NIS and is closely followed by its deterministic counterpart. As

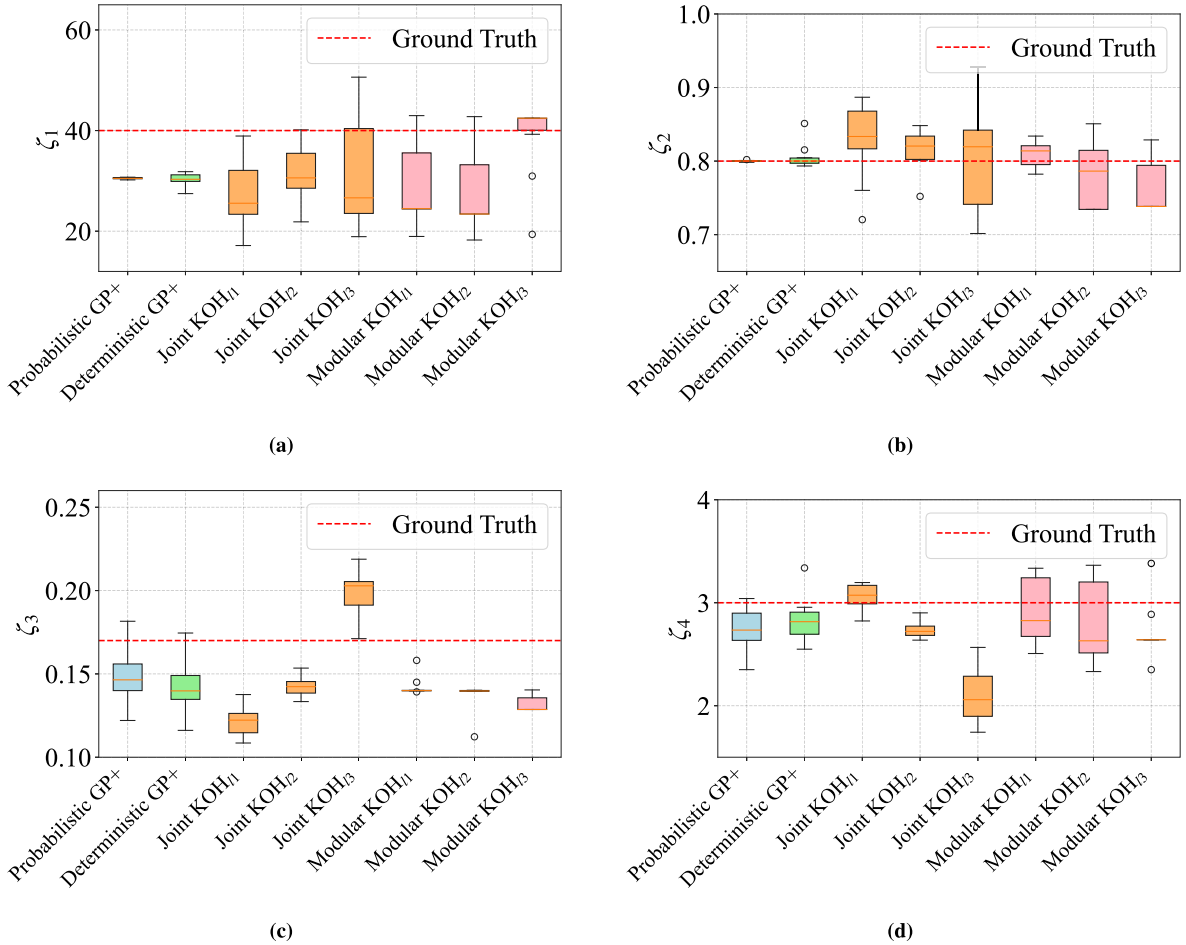


Fig. 14. Calibration performance in `Wing`: Ground truth values are the numbers used to generate the HF samples. While all methods estimate $\boldsymbol{\zeta}$ quite well, probabilistic GP+ performs the best, especially in the case of ζ_1 and ζ_2 which have low sensitivity index as enumerated in Table 8.

demonstrated in Fig. 14 we observe a similar trend in estimating the calibration parameters where both versions of GP+ not only provide estimates that are closer to the ground truth values, but also show more robustness to variations in the training data. Comparing the estimates across the four calibration parameters, we notice that all the models are less accurate in the case of ζ_1. We attribute this behavior to identifiability issues and the fact that the underlying functions are almost insensitive to ζ_1 (see the sensitivity analysis in Table 8).

5.4. Bayesian optimization

BO is a global optimization method that is increasingly used in optimizing black-box and expensive-to-evaluate objective functions. The two main ingredients of BO are an acquisition function (AF) [75,127–131] and an emulator which iteratively interact while searching for the global optimum. In BO, GPs are dominantly used for emulation [132–137] since they are easy and fast to train, can effectively learn from small data, and naturally provide prediction uncertainties which are needed in the AF.

Given the widespread use of GPs in BO, we equip GP+ with a few unique functionalities that streamline optimization of black-box and expensive-to-evaluate objective functions. As schematically demonstrated in Fig. 15, GP+ enables MFBO with just a few lines of codes; primarily with the BO function which has a few features that distinguish it from other BO packages such as BoTorch. First, it leverages the emulator described in Sections 4.1 to 4.3 which provides more accuracy than competing GP modeling packages. Second, it has the option to tailor the emulation process to BO by integrating MAP with scoring rules. As detailed in Appendix G, this integration improves the accuracy of GPs' prediction intervals and, in turn, improves the exploration aspect of BO in the context of MF problems [92]. Lastly, the BO function in GP+ has a specialized AF that quantifies the information value of HF and LF data such that they are used primarily for exploitation and exploration, respectively [11]. At iteration k during MFBO, this AF quantifies the value of a sample as:

$$\gamma_{\text{MFBO}}(\boldsymbol{u};j)=\begin{cases}\gamma_{LF}(\boldsymbol{u};j)/O_j=\dfrac{\tau_j\phi(\frac{y_j^*-\mu_j}{\tau_j})}{O_j} & j=[1,\dots,ds]\ \ \&\ \ j\neq l\\ \gamma_{HF}(\boldsymbol{u};l)/O_l=\dfrac{\mu_l-y_l^*}{O_l} & j=l\end{cases}$$

where $\gamma_{LF}(\cdot)$ and $\gamma_{HF}(\cdot)$ are the AFs of the low- and high-fidelity data sources, O_j is the cost of querying source j, y_j^* is the best function value obtained so far from source j (it is assumed that $j=l$ denotes the HF source which we aim to optimize), and $\phi(\cdot)$ denotes the probability density function (PDF) of the standard normal variable. $\tau_j=\tau_j(\boldsymbol{u})$ and $\mu_j=\mu_j(\boldsymbol{u})$ are the standard deviation and mean, respectively, of point $\boldsymbol{u}$ from source j. To proceed to iteration $k+1$, the AF in Eq. (32) and the emulator are used to solve an auxiliary optimization problem that determines the next point to sample and its corresponding data source:

$$[\boldsymbol{u}^{(k+1)},j^{(k+1)}]=\underset{\boldsymbol{u},j}{\operatorname{argmax}}\ \gamma_{\text{MFBO}}(\boldsymbol{u};j) \tag{32}$$

The BO function in GP+ uses two simple convergence criteria to stop the optimization process: overall data collection costs and the maximum number of iterations without improvement. The former is a rather generic metric but it can result in a considerably high number of iterations in the context of MF problems if an LF source is extremely inexpensive to query. The second metric avoids this issue by putting an upper bound on the maximum number of iterations. These convergence criteria can be easily modified in GP+.

To demonstrate the effectiveness of GP+ in MFBO we evaluate it against two alternatives: (1) an SF method (denoted by SFBO_{EI}) that uses the same emulator as GP+ but expected improvement as its AF, (2) BoTorch with STMF-GP and knowledge gradient (KG) as the emulator and AF, respectively (see Appendix H for more details). BoTorch is not applicable to the engineering examples studied below since it cannot handle categorical variables and reports y^* values that optimize the learned posterior (rather than the directly sampled data). In the following simulations we denote our methods via MFBO.

We utilize an analytic (Borehole) and two engineering (HOIP and HEA) examples for comparison. We use all 4 LF sources of Borehole whose formulation, initialization, and source-dependent sampling costs are provided in Appendix A.2. HOIP has 2 LF sources and the sampling costs are $40-10-1$ where 40 is associated with the HF source. We initialize MFBO with $15-20-15$ samples and note that the relative accuracy of the LF sources is unknown a priori. HEA is a 5-dimensional bi-fidelity problem where we start the optimization with $5-20$ HF-LF samples with sampling costs of $50-10$, respectively (see Appendix A.1 for more details on HOIP and HEA). In the two engineering problems, GP+ excludes the best compound from the HF dataset and then builds the initial data by randomly sampling from the MF datasets. In all the examples, the maximum number of iterations without improvement is 50 and a maximum budget 10 000 and 40 000 are used for the engineering and analytic examples, respectively. We report the results for 10 random initializations.

The results of our comparison studies are summarized in Figs. 16 and 17 where Fig. 16 illustrates the fidelity embeddings learned for

```
from gpplus.test_functions.physical import Borehole_MF_BO
from gpplus.utils import set_seed
from gpplus.preprocessing.normalizeX import standard
from gpplus.bayesian_optimizations.BO_GP_plus import BO, Visualize_BO

set_seed(4)                                                          #Set random seed for reproducibility
qual_dict = {8:5}                                                    #Categorical input indices
n_train_init = {"0": 5, "1": 5, "2": 50, "3": 5, "4": 50}            #Number of samples from each source
variable_lower_bounds = [100,990, 700,100,0.05,10,1000,6000]         #Lower bounds of variables
variable_upper_bounds = [1000,1110,820,10000,0.15,500,2000,12000]    #Upper bounds of variables
costs = {"0": 1000, "1": 100, "2": 10, "3":100, "4":10}              #Source- dependent sampling cost
U_init, y_init = Borehole_MF_BO(True,n_train_init)                   #Generate initial HF and LF data
U_init,umean, ustd = standard(U_init,qual_dict)                      #Normalize initial input
bestf, cost = BO(U_init,y_init,costs,variable_lower_bounds,variable_upper_bounds,umean,ustd,qual_dict,Borehole_MF_BO)
Visualize_BO(bestf,cost)                                             #Visualize performance
```

Fig. 15. Multi-fidelity Bayesian optimization in GP+: With just a few lines of code, we solve the Borehole problem where the goal is to optimize the HF source (denoted by "0") while leveraging four LF sources.

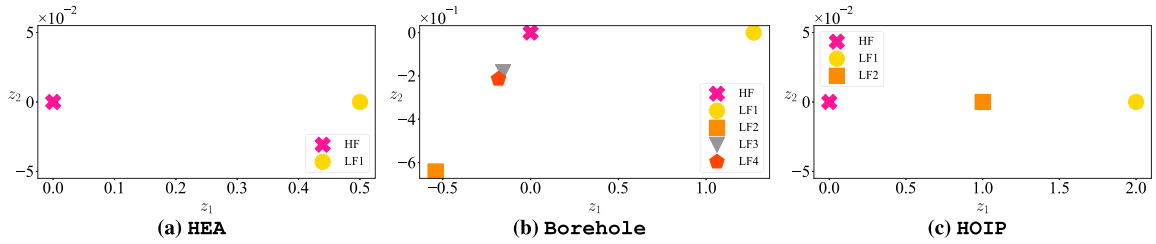


Fig. 16. Fidelity manifolds in BO problems: The figures display the fidelity embeddings learned by GP+ based on the initial data in each example. These embeddings quantify the global correlation among the LF sources and the HF source. Based on these plots, while the LF sources in Borehole and HEA are sufficiently correlated with the HF source, both LF sources of HOIP reveal low correlation with the HF source. This can cause premature convergence of GP+ due to its second stop condition that limits the number of iterations.

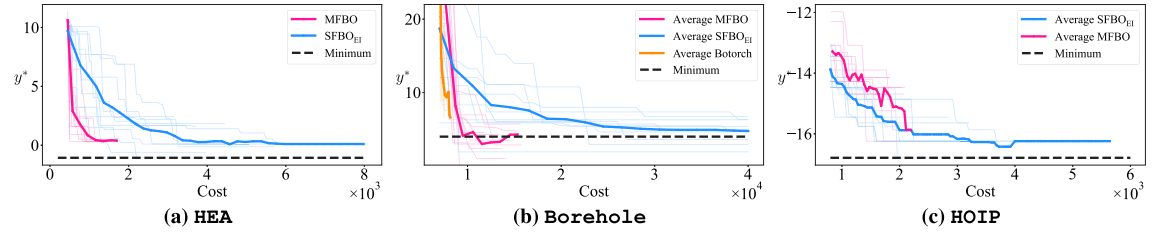


Fig. 17. Convergence histories in BO problems: The figures show the convergence history of each method versus their accumulated cost. The thin curves illustrate each repetition and the thick curves indicate the average behavior across the 10 repetitions. Based on these plots, GP+ significantly outperforms $SFBO_{EI}$ when LF sources are sufficiently correlated with the HF source (see also Fig. 16(a)). However, in scenarios without any correlated sources (HOIP, see Fig. 16(c)), GP+ may converge prematurely due to its second stop condition that limits the number of iterations.

each example by GP+. These embeddings are learnt based on the initial MF data and provide a quantitative metric for assessing the relative accuracy of each LF source with respect to the HF source. Based on these embeddings, while all the fidelity sources are globally correlated in HEA, two of the LF sources in HOIP and Borehole are highly biased and have a limited potential to improve optimization. The effects of these correlated/uncorrelated sources on the optimization are illustrated in Fig. 17 where the convergence histories are provided by tracking the best HF estimate found by each method (i.e., y_l^* in Eq. (32)) as a function of the accumulated sampling cost. Specifically, as shown in Fig. 17(a), the inexpensive correlated LF sources of HEA significantly improve the BO where MFBO finds the same compound as $SFBO_{EI}$ but at a much lower cost. We note that both MFBO and $SFBO_{EI}$ converge before finding the smallest HF value primarily because both the HF and LF data are inherently noisy.

As shown in Fig. 17(b), the superior performance of MFBO is more evident in the Borehole example which has two informative LF sources (LF3 and LF4 based on Fig. 16(b)). In this case, MFBO is able to effectively leverage the LF sources in exploring the input space and occasionally samples from the HF source even though the sampling cost associated with it is very high. This infrequent sampling reduces the overall cost but is necessary for converging to the true minimum. Unlike MFBO, BoTorch fails in this example since (1) its MF emulator is inaccurate, and (2) its AF incorrectly quantifies the information value to the extent that it cannot find an HF candidate whose value warrants its high sampling cost. As expected, $SFBO_{EI}$ converges to a value that is quite close to the minimum but at a much higher cost than MFBO.

HOIP is an example where the interpretable diagnostic tools of GP+ prove useful. Specifically, the fidelity embedding in Fig. 16(c) indicates that both LF sources are highly biased compared to the HF one and hence optimizing the latter may not benefit from sampling from the LF sources especially if the cost ratios are unbalanced. By investigating the optimization histories in Fig. 17(c) we realize that MFBO is unable to provide the same improvements as in the other two examples. Specifically, MFBO finds an optimum that is quite close to the one found by $SFBO_{EI}$ (−15.87 vs. −16.2) but it does so at a much lower cost (2000 vs. 6000) since it primarily samples from the LF sources, see Fig. 18. The performance of MFBO can be improved in such applications with highly biased LF sources by initializing the optimization process with more HF data.

6. Conclusions and future directions

In this paper, we introduce GP+ which is an open-source Python library that systematically integrates nonlinear manifold learning techniques with GPs. As demonstrated with the examples in Section 5, this

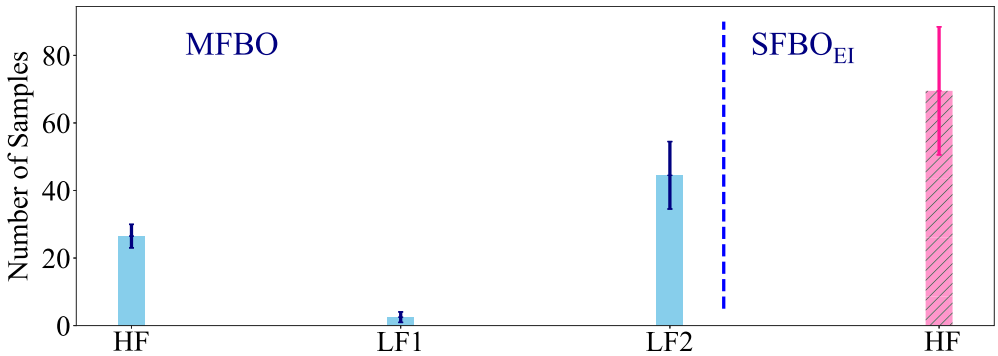


Fig. 18. Sampling history in HOIP: We report the number of samples taken from each source in MFBO and $SFBO_{EI}$ across the 10 repetitions. While MFBO converges with 25 HF samples on average, $SFBO_{EI}$ requires more than 70 HF samples.

integration provides a unified platform for studying a broad range of problems ranging from MF emulation to probabilistic calibration and BO.

In this paper, we primarily focused on applications where data (especially high-fidelity data) is scarce and hence probabilistic modeling has an edge over a deterministic one. We achieve this probabilistic nature in GP+ via variational approaches and plan to extend it in our future works based on MCMC. We anticipate this extension to more accurately quantify uncertainties but with a noticeable increase in computational costs. Additionally, all the problems analyzed in this paper were single-output. Currently, GP+ manages multi-output problems by concatenating the input space with categorical features to distinguish among the outputs. We aim to add more options for handling multi-output problems in future versions of GP+. Another interesting future direction is integrating our contributions with techniques that extend GPs to solve partial differential equations, classification tasks, or big data problems.

A unique advantage of GP+ over other GP modeling libraries is the interpretability of its learnt latent spaces. One example is the fidelity embedding that GP+ constructs for MF data which provides a visualizable metric that quantifies the relative similarity of the data sources. Based on the formulations in Sections 4.1 and 4.2, this fidelity embedding quantifies the global correlations among the data sources since it is not a function of the input features $\boldsymbol{x}$ or $\boldsymbol{t}$. We plan to extend this approach to learn local correlations but note that this extension will rely on larger training datasets since the required embedding functions will have more parameters. With a somewhat similar approach, we believe the calibration scheme introduced in Section 4.4 can be generalized to cases where not all the unknown parameters are shared across the computer models. This generalization is tightly connected to the so-called non-identifiability issues which are both worthy of in-depth future investigations.

CRediT authorship contribution statement

Amin Yousefpour: Writing – review & editing, Software, Methodology, Formal analysis, Conceptualization. **Zahra Zanjani Foumani:** Writing – review & editing, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Mehdi Shishehbor:** Software, Methodology. **Carlos Mora:** Methodology. **Ramin Bostanabad:** Writing – original draft, Supervision, Resources, Project administration, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data is available via GitHub. Link is in the paper.

Acknowledgments

We appreciate the support from NASA's Space Technology Research Grants Program, United States of America (grant number $80NSSC21K1809$), Office of the Naval Research, United States of America (grant number $N000142312485$), National Science Foundation, United States of America (grant number 2238038), and the UC National Laboratory Fees Research Program of the University of California, United States of America (Grant Number $L22CR4520$).

Appendix A. Details on data and benchmark problems

In this section, we provide details on the engineering (Appendix A.1) and analytic (Appendix A.2) examples that are used in Section 5 of the paper.

A.1. Engineering examples

Hybrid organic–inorganic perovskite (HOIP) is a material design problem where the goal is to identify the composition with the smallest inter-molecular binding energy [138]. The dataset used here is generated via three distinct sources that simulate the band gap property of HOIP as a function of composition based on the density functional theory (DFT). These compositions are characterized via three categorical variables that have 10, 3, and 16 levels (i.e., there are 480 unique compositions in total). The major differences between the datasets generated by the three sources are their fidelity (or accuracy) and size. Specifically, dataset generated by Source 1 is the most accurate among the three and contains 480 samples while the datasets generated by Source 2 and Source 3 contain 179 and 240 samples, respectively, and have lower levels of accuracy compared to Source 1. In Section 5.1 we exclusively use the HF data but use all three sources with the corresponding costs of 40, 10, and 1 in Section 5.4.

High entropy alloy (HEA) is another alloy design problem with the goal of finding the compound with the lowest thermal expansion coefficient [139]. This is a 5-dimensional example where the features show the percentage of each element (Fe, Ni, Co, Cr, V, Cu) in the alloy. It has one high- and one low-fidelity data sources whose corresponding costs are 50 and 10, respectively. Both datasets have 700 samples and are single-response.

The DNS-ROM dataset [125] aims to enhance the speed of multi-scale damage simulations for cast aluminum alloys. The acceleration process involves replacing the direct numerical simulations (DNS) at the microscale with reduced-order models (ROMs). The ROMs used in DNS-ROM have three distinct cluster counts: $(800, 1600, 3200)$, where a greater cluster count offers outcomes closer to DNS at a higher computational cost. DNS-ROM is a 6-dimensional problem where the features are all quantitative and characterize pore volume fraction, pore count, pore aspect ratio, average distance between neighboring pores, evolutionary rate parameter, and the critical effective plastic strain, with the latter two determining the material's damage behavior under stress. The number of samples from the highest to the lowest fidelities are $70, 110, 170,$ and 250, respectively.

NTA is the problem of designing a nanolaminate ternary alloy that is used in applications such as high-temperature structural materials [140]. These alloys have compositions of the form M_2AX where M is an early transition metal, A is a main group element, and X is either carbon or nitrogen. NTA is 3−dimensional with just categorical features which have 10, 12, and 2 levels. The dataset has 224 samples and the response is the bulk modulus of the alloy.

A.2. Analytic examples

The mathematical formulations are provided in Table 4 which also includes details on initializations and source-dependent sampling costs used in Section 5.4 for BO. Table 4 also shows the error of each LF source with respect to the corresponding HF source based on NRMSE in Eq. (30). That is:

$$NRMSE = \sqrt{\frac{(\boldsymbol{y}_l - \boldsymbol{y}_h)^T(\boldsymbol{y}_l - \boldsymbol{y}_h)}{10000 \times var(\boldsymbol{y}_h)}} \tag{A.1}$$

where $\boldsymbol{y}_l$ and $\boldsymbol{y}_h$ are vectors of size 10000×1 that store random samples taken from the low and high-fidelity sources, respectively.

Borehole is an 8-dimensional single-response example whose input space only has quantitative features. Only the Hf source is noisy in this case and there are 4 LF sources two of which are highly biased based on the NRMSEs. Note that the relative accuracy levels of the LF sources are calculated based on large data in Table 4 but interestingly these are consistent with the latent distances learnt by GP+ based on small data, see Fig. 16(b).

Table 4
Analytic examples: The examples have a diverse degree of dimensionality, number of sources, and complexity. n denotes the number of initial samples used in BO and the NRMSE of an LF source is calculated by comparing its output to that of the HF source at 10 000 random points, see Eq. (A.1).

Name	Source ID	Formulation	n	NRMSE	Cost	Noise
Borehole	HF	$\frac{2\pi T_u(H_u-H_l)}{\ln(\frac{r}{r_w})(1+\frac{2LT_u}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{T_l})}$	5	–	1000	2
	LF1	$\frac{2\pi T_u(H_u-0.8H_l)}{\ln(\frac{r}{r_w})(1+\frac{1LT_u}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{T_l})}$	5	4.40	100	–
	LF2	$\frac{2\pi T_u(H_u-H_l)}{\ln(\frac{r}{r_w})(1+\frac{8LT_u}{\ln(\frac{r}{r_w})r_w^2k_w}+0.75\frac{T_u}{T_l})}$	50	1.54	10	–
	LF3	$\frac{2\pi T_u(1.09H_u-H_l)}{\ln(\frac{2r}{r_w})(1+\frac{3LT_u}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{T_l})}$	5	1.30	100	–
	LF4	$\frac{2\pi T_u(1.05H_u-H_l)}{\ln(\frac{2r}{r_w})}\times\left(1+\frac{3LT_w}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{T_l}\right)$	50	1.3	10	–
Wing	HF	$0.036s_w^{0.758}w_{fw}^{0.0035}(\frac{A}{\cos^2(\Lambda)})^{0.6}\times q^{0.006}\times\lambda^{0.04}(\frac{100t_c}{\cos(\Lambda)})^{-0.3}(N_zW_{dg})^{0.49}+s_w w_p$	10	–	–	1
	LF1	$0.036s_w^{0.758}w_{fw}^{0.0035}(\frac{A}{\cos^2(\Lambda)})^{0.6}\times q^{0.006}\times\lambda^{0.04}(\frac{100t_c}{\cos(\Lambda)})^{-0.3}(N_zW_{dg})^{0.49}+w_p$	20	0.19	–	1
	LF2	$0.036s_w^{0.8}w_{fw}^{0.0035}(\frac{A}{\cos^2(\Lambda)})^{0.6}\times q^{0.006}\times\lambda^{0.04}(\frac{100t_c}{\cos(\Lambda)})^{-0.3}(N_zW_{dg})^{0.49}+w_p$	20	1.14	–	1
	LF3	$0.036s_w^{0.9}w_{fw}^{0.0035}(\frac{A}{\cos^2(\Lambda)})^{0.6}\times q^{0.006}\times\lambda^{0.04}(\frac{100t_c}{\cos(\Lambda)})^{-0.3}(N_zW_{dg})^{0.49}$	20	5.75	–	1
Sinusoidal	HF	$2\sin(x)$	4	–	–	1
	LF1	$2\sin(x)+0.3x^2-0.7x+1$	20	0.11	–	1

`Borehole-Mixed` refers to the `Borehole` example whose first and sixth features are converted into categorical variables with 5 distinct levels each. To achieve this, we first sample 5 values within the upper and lower bounds of these features. Subsequently, we randomly assign these sampled values to their corresponding variables and calculate the outputs using the formulation outlined in Table 4 for `Borehole`. Then, the categorical conversion is done by sorting the sampled values for the first and sixth features and substituting each feature's sampled point with its index in the sorted list.

To demonstrate the interpretability of embeddings learnt by `GP+`, we now use `Borehole-Mixed` to generate a training dataset of size 400 and then fit an emulator to it. Upon training, we visualize the latent points learnt for the two categorical variables and color-code them based on either the combination or response magnitude, see Figs. 19(a) and 19(b), respectively. As it can be observed in these figures, the learnt embeddings preserve the underlying numerical relations even though `GP+` does not have access to the numerical values used in data generation. For instance, all the variability in the latent space is in two directions with is in line with the fact that all combinations of the categorical data can be quantified via the two underlying numerical features. We also observe that close-by (distant) latent points have similar (different) response values which increases as we move from the top-right corner to the bottom left corner of the latent space, see Fig. 19(b). These embeddings also indicate variable importance. For instance, based on Fig. 19(a) we observe that changing the levels of the first categorical variable from "a" to "e" results in larger latent movements than changing the levels of the second categorical variable. Such a behavior indicates the underlying function is more sensitive to the former variable and we validate this argument in Appendix B by conducting sensitivity analyses.

`Wing` is a 10-dimensional single-response example with one HF and 3 LF sources. Based on the NRMSE values shown in Table 4, the source ID, true fidelity level, and sampling costs follow the same trend (unlike `Borehole`). For instance, the first LF source is the most accurate and most expensive among all the LF sources. Additionally, a Gaussian noise with a standard deviation of 1 is added to all the fidelity sources.

`Sinusoidal` is a 1-dimensional bi-fidelity problem where there are high correlations between the two sources as indicated by the NRMSE value in Table 4. We use a standard Gaussian noise to corrupt the data from both sources.

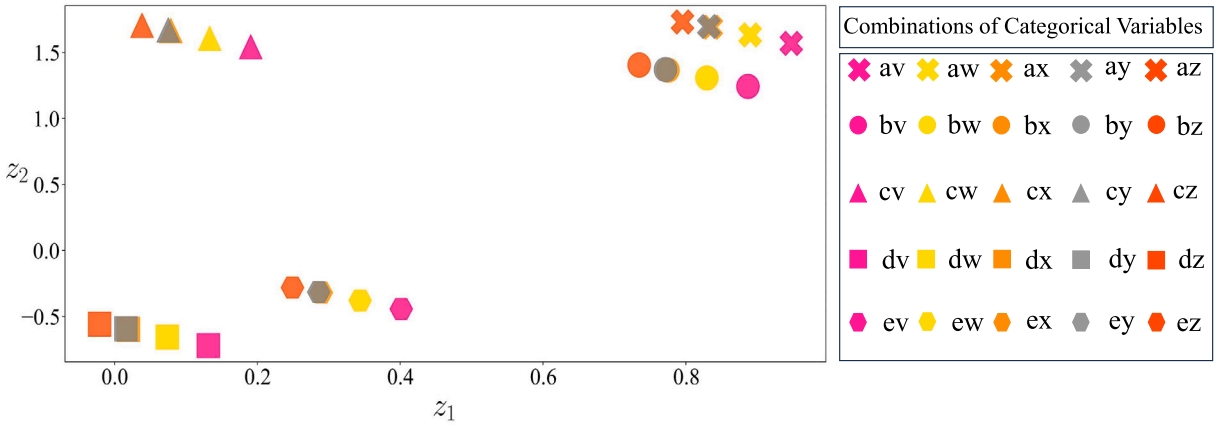
A.3. Calibration examples

`Wing`, `Borehole` and `beam deflection` are used in Section 5.3 to illustrate the performance of `GP+` in calibration. Their formulation and details are provided in Table 5. As explained in Section 5.3, there are 2 and 4 calibration parameters in `Borehole` and `Wing`, respectively. Also, `beam deflection` is a bi-fidelity 5-dimensional problem [126] whose features are a constant distributed load applied to the beam ($p = 12000$ N/m), width ($b = 0.15$ m), height ($h = 0.3$ m), and the length of the beam ($L = 5$ m). The last feature is the unknown material Young's modulus (ζ) which we aim to estimate. The analytic formulation and more details regarding `beam deflection` is also presented in Table 5.

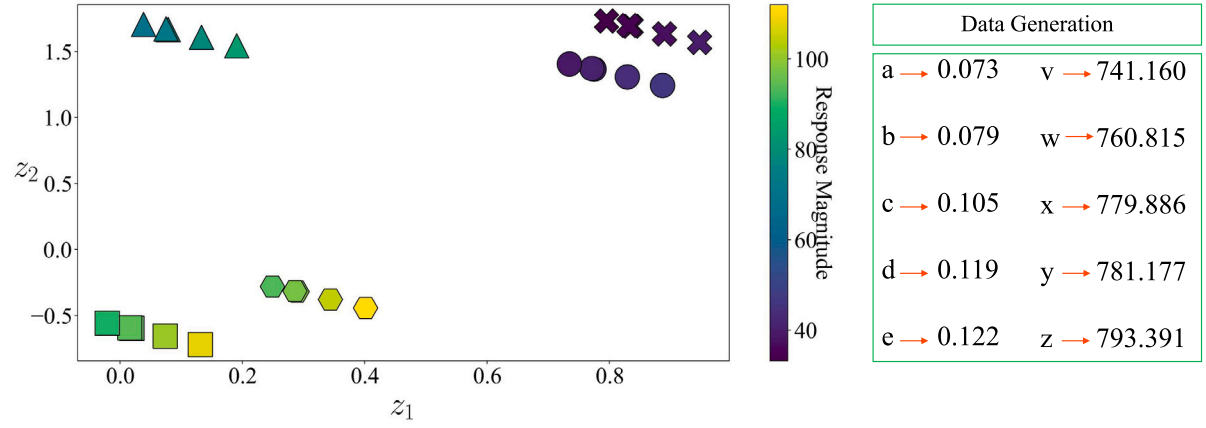
Appendix B. Sensitivity analysis

Sobol sensitivity analysis is a global variance-based method used for quantifying each input's main and total contribution to the output variance [141]. While main-order Sobol indices (SIs) reveal the individual contributions of input variables, total-order indices capture both the individual and interaction effects of inputs on the output. We highlight that while Sobol indices are typically applied to quantitative features, we extend the idea to qualitative features in `GP+` by sampling random quantitative values and mapping them to the unique levels of a categorical variable. This functionality is accessible with the `model.Sobol()` command in `GP+`.

The $\boldsymbol{\omega}$ parameter in GP-based emulation (see Section 2 for details) plays a similar role to the Sobol indices in that it reveals the sensitivity of the emulator to the quantitative features where a smaller ω_i value indicates that the output is less sensitive to the ith feature. Since $\boldsymbol{\omega}$ are not defined for categorical inputs, we propose to measure their sensitivities based on the average distance among the learnt encoded points. Specifically, we encode the l_i categories of variable t_i to l_i latent points whose average inter-distances is then used to measure the sensitivity of the output to t_i. We denote this metric by S_{cat} and highlight that we calculate it by endowing each categorical variable with its own latent space (this is in contrast to the examples in Sections 5.1 and 5.4 where we encode all the combinations of all categorical variables into



(a) Colors correspond to the unique combinations of levels of the two categorical variables.



(b) Colors are based on the response magnitude.

Fig. 19. Latent positions learned via GP+ for `Borehole-Mixed`: We use the numerical values shown in **(b)** to generate 400 training samples via `Borehole`. These values are then replaced with the corresponding categorical features before GP+ is used for emulation. Upon training, we visualize the embedding learnt by GP+ while color-coding it based on either the variable combinations or response magnitude.

Table 5
Analytic examples used for calibration: n denotes the number of samples used in Section 5.3 and the NRMSE of an LF source is calculated by comparing its output to that of the HF source at 10 000 random points, see Eq. (A.1).

Name	Source ID	Formulation	n	NRMSE	Noise
`Borehole`	HF	$\frac{2\pi T_u(H_u-H_l)}{\ln(\frac{r}{r_w})(1+\frac{2\times1500T_u}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{250})}$	20	–	2
	LF1	$\frac{2\pi T_u(H_u-0.8H_l)}{\ln(\frac{r}{r_w})(1+\frac{1.5T_u}{\ln(\frac{r}{r_w})r_w^2k_w}+\frac{T_u}{\zeta_2})}$	100	4.40	–
	LF2	$\frac{2\pi T_u(H_u-H_l)}{\ln(\frac{r}{r_w})(1+\frac{8\zeta_1T_u}{\ln(\frac{r}{r_w})r_w^2k_w}+0.75\frac{T_u}{\zeta_2})}$	100	1.54	–
`Wing`	HF	$0.036s_w^{0.758}w_{fw}^{0.0035}(\frac{A}{\cos^2(A)})^{0.6}\times40^{0.006}$ $\times0.85^{0.04}(\frac{100\times0.17}{\cos(A)})^{-0.3}(3W_{dg})^{0.49}+s_ww_p$	25	–	1
	LF1	$0.036s_w^{0.758}w_{fw}^{0.0035}(\frac{A}{\cos^2(A)})^{0.6}\times\zeta_1^{0.006}$ $\times\zeta_2^{0.04}(\frac{100\zeta_3}{\cos(A)})^{-0.3}(\zeta_4W_{dg})^{0.49}+w_p$	40	0.19	1
	LF2	$0.036s_w^{0.8}w_{fw}^{0.0035}(\frac{A}{\cos^2(A)})^{0.6}\times\zeta_1^{0.006}$ $\times\zeta_2^{0.04}(\frac{100\zeta_3}{\cos(A)})^{-0.3}(\zeta_4W_{dg})^{0.49}+w_p$	50	1.14	1
	LF3	$0.036s_w^{0.9}w_{fw}^{0.0035}(\frac{A}{\cos^2(A)})^{0.6}\times\zeta_1^{0.006}$ $\times\zeta_2^{0.04}(\frac{100\zeta_3}{\cos(A)})^{-0.3}(\zeta_4W_{dg})^{0.49}$	60	5.75	1
`Beam Deflection`	HF	$\frac{5}{32}\frac{pL^4}{3\times10^{10}bh^3}$	1	–	0.05
	LF1	$\frac{5}{32}\frac{pL^4}{\zeta_1bh^3}$	200	–	0

a single latent space). We adopt this approach primarily for ease of implementation and increasing interpretability.

To demonstrate the interpretability of a GP's parameters, we compare them against main and total SIs in `Borehole`, `Borehole-Mixed`, and `Wing` problems. To this end, we train three GPs via GP+ to emulate these functions (we use sufficient samples to ensure the trained GPs accurately learn the underlying functions) and then compare the estimated parameters of these GPs to main and total SIs. The results are enumerated in Tables 6–8 and indicate that there is a good agreement between the two different metrics, that is, important features that have

Table 6
Sensitivity analysis of Borehole: Sensitivity analysis of Borehole using Sobol indices and GP+ emulator. Based on Eq. (13). In Section 5.3, $\zeta = [T_l, L]^T$ are treated as calibration parameters.

Metric	Features							
	r_w	r	T_u	H_u	T_l	H_l	L	k_w
Main SI	0.830	1.57e−7	2.34e−8	0.042	2.09e−6	0.041	0.039	0.009
Total SI	0.868	3.65e−6	4.07e−8	0.054	1.25e−5	0.054	0.051	0.013
10^{ω_i}	0.125	1.00e−4	2.45e−7	0.009	2.95e−5	0.013	0.029	0.003

Table 7
Sensitivity analysis of Borehole-Mixed: Sensitivity analysis of Borehole-Mixed using Sobol indices and GP+ emulator. The ω parameters in GP+ are only learned for quantitative features.

Metric	Features							
	r_w	r	T_u	H_u	T_l	H_l	L	k_w
Main SI	0.541	6.67e−6	1.68e−7	0.153	1.92e−5	0.091	0.153	0.037
Total SI	0.562	3.98e−6	6.95e−8	0.163	2.83e−5	0.096	0.164	0.037
10^{ω_i}	–	4.00e−4	5.06e−6	0.013	3.00e−4	–	0.029	0.004
S_{cat}	0.115	–	–	–	–	0.019	–	–

large SIs also have large 10^{ω_i} or S_{cat}. For instance, r_w, H_u, H_l, and L are the most sensitive features of Borehole and Borehole-Mixed as indicated by both Sobol and GP+. We note that reported metrics in Tables 6 and 7 are slightly different since the latter is affected by categorization of two of its inputs.

Appendix C. Emulation and optimization options

GP+ offers a wide range of options that streamline its adoption for a wide range of applications that involve emulation, MF modeling, identification of model form errors, inverse parameter estimation, and BO. In Tables 9 and 10 we provide a comprehensive list of options related to model initialization and training, respectively. We have chosen the default values of these options based on the most common uses of GPs while striking a balance between accuracy and cost.

Appendix D. Mixed single-task GP (MST-GP)

As mentioned in Section 5.1, BoTorch employs MST-GP to model problems with categorical variables. MST-GP defines two distinct correlation functions for numerical and categorical features. The final correlation function is the combination of these two:

$$r\left(\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{t}\end{bmatrix}, \begin{bmatrix}\boldsymbol{x}'\\ \boldsymbol{t}'\end{bmatrix}; \boldsymbol{\Omega}\right) = r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}_1) + r(\boldsymbol{t}, \boldsymbol{t}'; \boldsymbol{\omega}_2) + r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}_3) \times r(\boldsymbol{t}, \boldsymbol{t}'; \boldsymbol{\omega}_4) \tag{D.2}$$

where $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2, \boldsymbol{\omega}_3, \boldsymbol{\omega}_4\} \in \boldsymbol{\Omega}$ are the distinct length scale parameters for each correlation function. Specifically, $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_3$ are length scale parameters associated with quantitative features while $\boldsymbol{\omega}_2$ and $\boldsymbol{\omega}_4$ scale the categorical features ($\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_3$ are dx-dimensional while $\boldsymbol{\omega}_2$ and $\boldsymbol{\omega}_4$ are of dimension dt). $r(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\omega}_j)$ is the Matèrn correlation function with $\upsilon = 2.5$ while $r(\boldsymbol{t}, \boldsymbol{t}'; \boldsymbol{\omega}_j)$ is formulated as:

$$r\left(\boldsymbol{t}, \boldsymbol{t}'; \boldsymbol{\omega}_j\right) = \exp\left\{(-\sum_{i=1}^{dt} \frac{(t_i - t_i')}{\omega_{ji}})/dt\right\} \tag{D.3}$$

where ω_{ji} is the length scale parameter estimated for the i(th) feature through correlation function j. Also, $(t_i - t_i')$ is the Hamming distance which is 0 when the two categorical variables are the same and 1 otherwise.

Appendix E. Single-task multi-fidelity GP (STMF-GP)

The STMF-GP modifies the correlation function of GP for MF hierarchical MF modeling. Specifically, it adopts an additive covariance function that relies on introducing two user-defined quantitative features [4,30]. The first feature, denoted by x_a, is restricted to the $[0, 1]$ range and assigns a fidelity value to a source based on the user's belief (larger values correspond to higher fidelities). This assigned fidelity value directly affects the correlation and cost function. The second feature, denoted by x_b, is the iteration fidelity parameter and benefits MF BO specifically in the context of hyperparameter tuning of large machine learning models. These two features are used in three user-defined functions defined as follows. $e_1(\cdot)$ and $e_3(\cdot)$ are bias kernels that aim to take the discrepancies among the sources into account:

$$e_1(x_a, x_a') = (1 - x_a)(1 - x_a')(1 + x_a x_a')^p \tag{E.4}$$

$$e_3(x_b, x_b') = (1 - x_b)(1 - x_b')(1 + x_b x_b')^p \tag{E.5}$$

where p is the degree of polynomial (which needs to be estimated) and has a Gamma prior. $e_2(\cdot)$ is the interaction term with four deterministic terms and one polynomial kernel:

$$\begin{aligned} e_2([x_a, x_b]^T, [x_a', x_b']^T) = (1 - x_b)(1 - x_b')(1 - x_a)(1 - x_a') \\ \times (1 + [x_a, x_b]^T [x_a', x_b']^T)^p \end{aligned} \tag{E.6}$$

Finally, the modified covariance function is [142]:

$$\begin{aligned} \mathrm{cov}(\boldsymbol{x}, \boldsymbol{x}') = & c(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_0, {\sigma_0}^2) + e_1(x_a, x_a') c(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_1, {\sigma_1}^2) \\ & + e_2([x_a, x_b]^T, [x_a', x_b']^T) c(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_2, {\sigma_2}^2) \\ & + e_3(x_b, x_b') c(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_3, {\sigma_3}^2) \end{aligned} \tag{E.7}$$

where $c(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}_i, {\sigma_i}^2)$ is the Matern kernel that characterize the spatial correlations across the numerical inputs (the parameters of these kernels are endowed with Gamma priors in BoTorch).

In Section 5.2 we use STMF-GPs as one of the baselines for evaluating the performance of GP+ in MF emulation. Therein, we assing two sets of values to the fidelity indices of STMF-GP to quantify their effect on the results. These two sets of values are enumerated in Table 11.

Appendix F. Neural network architectures for multi-fidelity modeling

In Section 5.2, FFNNs are employed both as MF emulators and as basis functions in GP+. We design the architecture of these models as detailed in Table 12.

Appendix G. Bayesian optimization (BO)

In this section, we first explain how GP+ handles highly biased sources in MFBO and then provide a few options available in GP+ for SFBO and MFBO.

G.1. BO improvements

As mentioned in Section 5.4, the accuracy of the emulator in quantifying uncertainties significantly affects the performance of BO. This impact is more noticeable in MF problems where biased LF sources can misguide the optimization process. To address this challenge, we employ interval scores (IS) to penalize the objective function. We choose IS since it is robust to outliers, rewards narrow prediction intervals, and is flexible in the choice of desired coverage levels [143,144]. IS is a special case of quantile prediction that penalizes the emulator for each observation that is not inside the $(1-\upsilon)\times 100\%$ prediction interval and is calculated as:

$$\begin{aligned} IS = & \frac{1}{n}\sum_{i=1}^{n} (\mathcal{U}^{(i)} - \mathcal{L}^{(i)}) + \frac{2}{\upsilon}(\mathcal{L}^{(i)} - y^{(i)}) \mathbb{1}\{y^{(i)} < \mathcal{L}^{(i)}\} \\ & + \frac{2}{\upsilon}(y^{(i)} - \mathcal{U}^{(i)}) \mathbb{1}\{y^{(i)} > \mathcal{U}^{(i)}\} \end{aligned} \tag{G.8}$$

where $y^{(i)} = y(\boldsymbol{u}^{(i)})$ is the response of the ith training sample. $\mathcal{U}^{(i)}$, $\mathcal{L}^{(i)}$, υ and $\mathbb{1}$ are defined in Section 5.

Table 8
Sensitivity analysis of `Wing`: Sensitivity analysis of `Wing` using Sobol indices and GP+emulator. In Section 5.3, $\boldsymbol{\zeta} = [q, \lambda, t_c, N_z]^T$ are treated as calibration parameters.

Metric	Features									
	S_w	w_{fw}	A	Λ	q	λ	t_c	N_z	w_{dg}	w_p
Main SI	0.125	2.3e−6	0.220	4.8e−4	8.4e−5	1.8e−3	0.142	0.412	0.085	3.3e−3
Total SI	0.128	2.4e−6	0.225	5.1e−4	8.7e−5	1.8e−3	0.146	0.420	0.087	3.3e−3
10^{ω_i}	0.005	8.7e−7	0.010	0.006	2.0e−4	5.0e−4	0.027	0.023	0.005	2.0e−4

Table 9
Model options: We provide a range of options that streamline the adoption of GP+ in many applications with just a few lines of code.

Option name	Description	Type/Default setting
`train_x`	Input	Tensor
`train_y`	Output (response)	Tensor
`dtype`	Data type of the model and data	`torch.float`
`device`	Device to build the model (CPU or CUDA)	`"cpu"`
`qual_dict`	Column index and number of levels of categorical variables	{}
`multiple_noise`	Modeling separate noise for each data source	`False`
`lb_noise`	Lower bound for noise	$1e-8$
`fix_noise`	Flag for estimating noise via the nugget parameter	`False`
`fix_noise_val`	Fixed noise value if `fix_noise = True`	$1e-5$
`quant_correlation_class`	Kernel of numerical variables	`"Rough_RBF"`
`fixed_length_scale`	Flag to fix the length scale	`False`
`fixed_length_scale_val`	Fixed length scale value	`torch.tensor([1.0])`
`encoding_type`	Type of $f_{\boldsymbol{x}}(t)$	`"one-hot"`
`embedding_dim`	Dimension of embedding (manifold) to be learnt	2
`separate_embedding`	Which categorical features be learned separately	[ ]
`embedding_type`	Type of embedding	`"deterministic"`
`NN_layers_embedding`	Network structure of $f_h(\boldsymbol{x}_t, \boldsymbol{\theta}_h)$	[ ]
`m_gp`	Type of mean function for GP	`"single"`
`m_gp_ref`	Mean function for reference source (ID = 0)	`"zero"`
`NN_layers_m_gp`	Structure of neural network for mean function	[4, 4]
`calibration_type`	Deterministic or probabilistic calibration	`"deterministic"`
`calibration_id`	Index of the parameter to be calibrated	[ ]
`mean_prior_cal`	Mean prior for calibration parameter	0
`std_prior_cal`	Standard deviation prior for calibration parameter	1
`interval_score`	Interval scoring during optimization	`False`
`num_pass_train`	Number of training passes; deterministic/probabilistic	1/20
`num_pass_pred`	Number of prediction passes; deterministic/probabilistic	1/30

Table 10
Options for model training: These options control the optimization process and their default values are selected to strike a balance between accuracy and cost.

Option name	Description	Type/Default setting
`add_prior`	Flag for using MAP instead of MLE	`True`
`jac`	Flag for using Jacobian	`True`
`num_restarts`	Number of optimization restarts	32
`method`	Optimization method	`"L-BFGS-B"`
`options`	Optional parameters	{}
`n_jobs`	Number of cores (uses all cores if `n_jobs` = −1)	−1
`constraint`	Flag for adding constraints	`False`
`bounds`	Flag for adding bounds on parameters	`False`
`regularization_parameter`	Regularization coefficients	[0, 0]

Table 11
Fidelity indices of STMF-GP: We use these fidelity indices in Section 5.2 to demonstrate their effect of MF emulation. Even though the two sets of numbers are close, the performance of the corresponding emulators are quite different.

Model	Fidelity parameters ($\boldsymbol{x}_a$)		
	Sinusoidal	Wing-weight	DNS-ROM
$STMF-GP_1$	[1, 0.25]	[1, 0.96, 0.83, 0.49]	[1, 0.96, 0.83, 0.49]
$STMF-GP_2$	[1, 0.5]	[1, 0.75, 0.5, 0.25]	[1, 0.6, 0.4, 0.2]

Table 12
Network architectures of feed-forward neural networks: We design different architectures for our MF emulation studies in Section 5.2.

Method	Option	Problems		
		`Sinusoidal`	`Wing`	`DNS-ROM`
GP+	Small FFNN as $m(\boldsymbol{u}; \boldsymbol{\beta})$	[1, 2]	[4, 4]	[4, 4]
	Medium FFNN as $m(\boldsymbol{u}; \boldsymbol{\beta})$	[2, 2, 2]	[8, 8, 8]	[8, 8, 8]
FFNN	Small	[4, 4]	[8, 8, 8]	[8, 8, 8]
	Medium	[16, 16]	[4, 16, 32]	[32, 32, 32]
	Large	[16, 32]	[4, 16, 128]	[128, 128, 32]

Using the IS in Eq. (G.8) we now formulate the new objective function for emulation within BO where IS is used as a penalty term. Since the effectiveness of this penalization mechanism depends on the value of the posterior, we introduce an adaptive coefficient whose magnitude depends on the posterior value. With this penalty term, the modified objective function for the GP emulator is:

$$[\hat{\boldsymbol{\beta}}, \hat{\sigma}^2, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\delta}}] = \underset{\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}, \boldsymbol{\delta}}{\operatorname{argmin}} \; L_{MAP} + \epsilon |L_{MAP}| \times IS \tag{G.9}$$

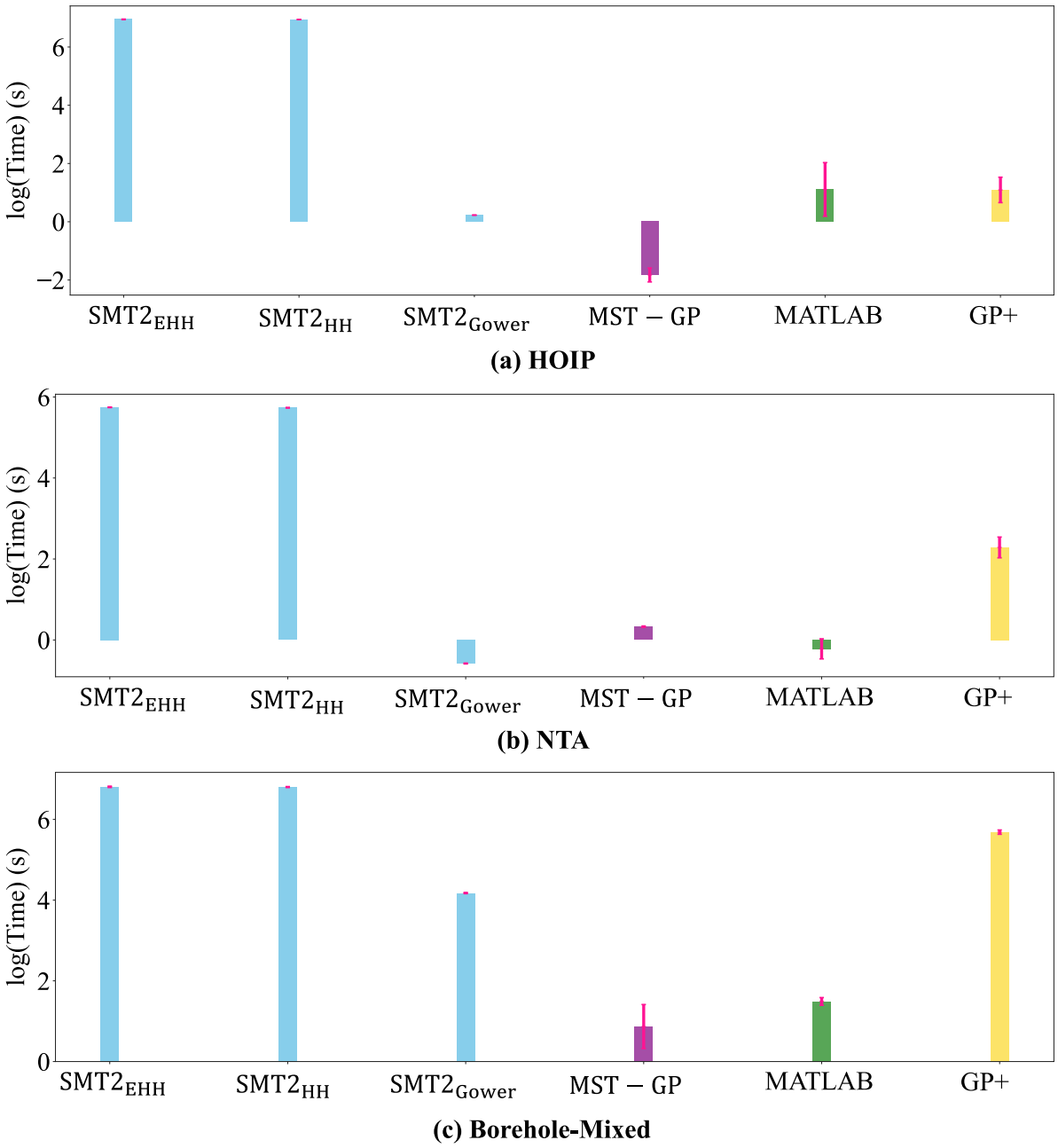


Fig. 20. Training cost in examples with categorical inputs: Each bar in the graph represents the computational costs associated with different methods for various examples. Since EHH, HH, and Gower are from SMT2, we use the same color to represent them. In the context of mixed-input problems, the manifold learning process of GP+ leads to a slight increase in its computational cost. However, given the significantly higher accuracy it achieves, this difference in time is justified.

where $|\cdot|$ denotes the absolute function and ϵ is a user-defined scaling parameter which is set to 0.08 by default in GP+. We refer the reader to [92] for more in-depth information.

G.2. BO options

The BO options of GP+ are summarized in Table 13. We note that GP+ is able to handle both analytic functions and datasets for BO. This versatility is achieved by specifying the analytic function for the former (see Fig. 15 for an example) and utilizing datasets for the latter through the `data_func` option.

Appendix H. BoTorch

`BoTorch` is an MF cost-aware BO package that employs STMF-GP (explained in Appendix E) as the emulator and leverages the knowledge gradient (KG) as the AF. KG is a look-ahead AF that chooses the next sampling point ($\boldsymbol{x}^{(k+1)}$) based on the effect of the yet-to-be-seen observation (i.e., y^{k+1} which follows a normal distribution) on the optimum value predicted by the emulator. This AF quantifies the expected utility of $\boldsymbol{x}$ at iteration $k+1$ as:

$$\gamma_{KG}(\boldsymbol{x}) = \mathbb{E}_{p(y|\boldsymbol{x},D^k)}[\max \mu^{(k+1)}] - \max \mu^{(k)} \tag{H.10}$$

Table 13
Options provided by GP+ for Bayesian optimization: GP+ accommodates both single- and multi-fidelity BO. The user can provide both analytic functions and datasets where the former is typically used in comparison studies while the latter is used in real world applications.

Option name	Description	Type/Default setting
`U_init`	Initial input	Tensor
`y_init`	Initial output (response)	Tensor
`costs`	Source-dependent sampling costs	{}
`l_bound`	Lower bound of the variables	[ ]
`u_bound`	Upper bound of the variables	[ ]
`U_mean`	Mean of the initial inputs	[ ]
`U_std`	Standard deviation of the initial inputs	[ ]
`qual_dict`	Column index and number of levels of categorical variables	{}
`data_func`	Data (function/dataset)	–
`n_train_init`	Number of initial data	{}
`maximize_flag`	Flag for maximization	`False`
`one_iter`	Flag for suggesting only one new sample	`False`
`max_cost`	Maximum budget for optimization	40 000
`MF`	Flag for doing MFBO	`True`
`AF_hf`	AF of HF source	`AF_HF`
`AF_lf`	AF of LF source	`AF_LF`
`IS`	Flag for penalizing the objective function with scoring rules	`True`

where $\mathcal{D}^k = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^k$ is the training data in iteration k and $\max \mu^{(k)} = \max \mu^{(k)}(\boldsymbol{x})$ denotes the maximum mean prediction of the emulator trained on $\mathcal{D}^k$. The expectation operation in Eq. (H.10) appears due to the fact that $y^{(k+1)}$ is not observed yet and $\alpha_{KG}(\boldsymbol{x})$ is relying on the predictive distribution provided by the emulator that is trained on $\mathcal{D}^k$. This expectation cannot be calculated analytically and hence a Monte Carlo estimate is used in practice:

$$\gamma_{KG}(\boldsymbol{x}) \approx \frac{1}{M} \sum_{m=1}^{M} \max \mu^{(k+1)^m} - \max \mu^{(k)} \tag{H.11}$$

where $\max \mu^{(k+1)^m} = \max \mu^{(k+1)^m}(\boldsymbol{x})$ is calculated by first drawing a sample at $\boldsymbol{x}$ from the emulator that is trained on $\mathcal{D}^k$ and then retraining the emulator on $\mathcal{D}^k \cup (\boldsymbol{x}, y^m)$ where y^m is response of the drawn sample. In practice, a small value must be chosen for M since maximizing $\gamma_{KG}(\boldsymbol{x})$ over the input space at each iteration of BO is very expensive. Refer to [145,146] for more information on KG and its implementation.

Appendix I. Computational costs

In this section, we compare the computational costs of various baselines discussed in Section 5.1 for the examples outlined in Table 1. The results are summarized in Figs. 20 and 21. Fig. 20 is for problems whose input space has categorical variables while Fig. 21 is for problems that only have numerical inputs. We observe in Fig. 20 that

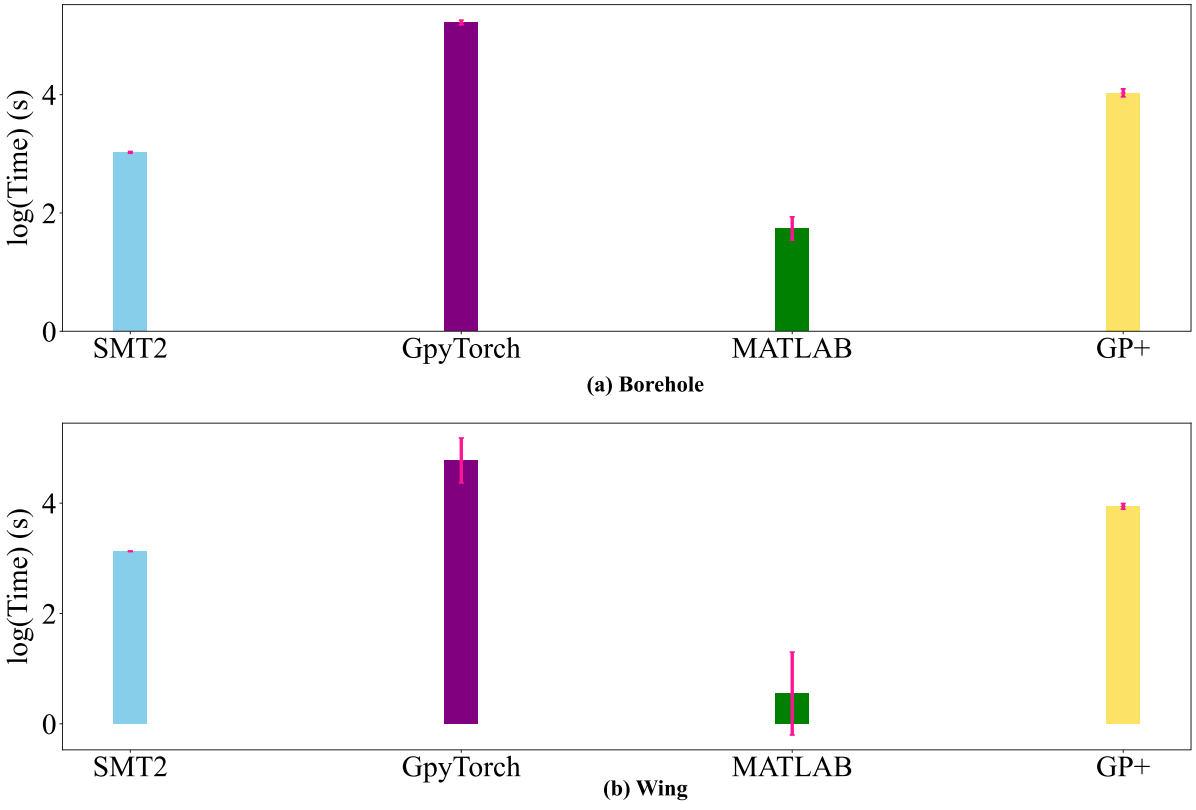


Fig. 21. Training cost in examples with all-numerical inputs: Each bar in the graph represents the computational costs associated with different methods for various examples. In the context of numerical problems, all the benchmarks have the same formulation and they just differ in their optimization. This difference causes slightly different computational costs.

GP+ is slightly more expensive than Gower, MST-GP, and MATLAB. This is because, unlike packages that convert categorical features to numerical ones in a naive manner (MATLAB, Gower, and MST-GP), GP+ nonlinearly learns the underlying characteristics of the categorical inputs. Additionally, we find that the larger number of parameters used in $SMT2_{EHH}$ and $SMT2_{HH}$ not only fails to improve accuracy, but also leads to significantly higher computational costs for SMT2.

In numerical examples (Wing and Borehole, see Fig. 21), the formulations of all methods are the same and they only differ in the parameter estimation and optimization. Specifically, for MATLAB, we employ default settings that utilize MLE for parameter estimation and the BFGS algorithm for optimization. The lower computational costs observed in MATLAB are attributed to its optimization process and streamlined implementation. SMT2 exhibits slightly lower computational costs compared to GP+ and GPytorch which is due to its use of the profiling method. Furthermore, while GP+ and GPytorch are quite similar, the minor difference in their computational costs stems from the different optimizers they employ (GP+ uses L-BFGS from SciPy by default, while GPytorch uses Adam[21] from PyTorch).

References

[1] Balachandran PV, Xue D, Theiler J, Hogden J, Lookman T. Adaptive strategies for materials design using uncertainties. Sci Rep 2016;6:19660. http://dx.doi.org/10.1038/srep19660.

[2] Kusne AG, Yu H, Wu C, Zhang H, Hattrick-Simpers J, DeCost B, Sarker S, Oses C, Toher C, Curtarolo S, Davydov AV, Agarwal R, Bendersky LA, Li M, Mehta A, Takeuchi I. On-the-fly closed-loop materials discovery via Bayesian active learning. Nature Commun 2020;11(1):5966. http://dx.doi.org/10.1016/j.cma.2023.115937.

[3] Zhang Y, Apley DW, Chen W. Bayesian optimization for materials design with mixed quantitative and qualitative variables. Sci Rep 2020;10(1):4924. http://dx.doi.org/10.1038/s41467-020-19597-w.

[4] Balandat Maximilian, Karrer Brian, Jiang Daniel, Daulton Samuel, Letham Ben, Wilson Andrewa G, Bakshy Eytan. Botorch: A framework for efficient Monte-Carlo Bayesian optimization. Adv Neural Inf Process Syst 2020;33.

[5] Astudillo Raul, Frazier Peter. Bayesian optimization of composite functions. In: International Conference on Machine Learning. 2019.

[6] Wu Jian, Toscano-Palmerin Saul, Frazier Peter I, Wilson Andrew Gordon. Practical multi-fidelity bayesian optimization for hyperparameter tuning. 2020.

[7] Herbol Henrya C, Poloczek Matthias, Clancy Paulette. Cost-effective materials discovery: Bayesian optimization across multiple information sources. Mater Horiz 2020;7(8):2113–23.

[8] Wang Yifan, Chen Tai-Ying, Vlachos Dionisiosa G. Nextorch: a design and Bayesian optimization toolkit for chemical sciences and engineering. J Chem Inf Model 2021;61(11):5312–9.

[9] Takeno Shion, Fukuoka Hitoshi, Tsukada Yuhki, Koyama Toshiyuki, Shiga Motoki, Takeuchi Ichiro, Karasuyama Masayuki. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In: International conference on machine learning. PMLR; p. 9334–45.

[10] Tran Anh, Tranchida Julien, Wildey Tim, Thompson Aidana P. Multi-fidelity machine-learning with uncertainty quantification and Bayesian optimization for materials design: Application to ternary random alloys. J Chem Phys 2020;153(7):074705. http://dx.doi.org/10.1063/5.0015672.

[11] Zanjania Foumani Zahra, Shishehbor Mehdi, Yousefpour Amin, Bostanabad Ramin. Multi-fidelity cost-aware Bayesian optimization. Comput Methods Appl Mech Engrg 2023;407:115937.

[12] Rasmussen Carla Edward. Gaussian processes for machine learning. 2006.

[13] Batlle Pau, Darcy Matthieu, Hosseini Bamdad, Owhadi Houman. Kernel methods are competitive for operator learning. 2023, arXiv preprint arXiv:2304.13202.

[14] Chen Yifan, Hosseini Bamdad, Owhadi Houman, Stuart Andrewa M. Solving and learning nonlinear PDEs with Gaussian processes. J Comput Phys 2021;447:110668.

[15] Meng Rui, Yang Xianjin. Sparse Gaussian processes for solving nonlinear PDEs. J Comput Phys 2023;490:112340.

[16] Oune N, Bostanabad R. Latent map Gaussian processes for mixed variable metamodeling. Comput Methods Appl Mech Engrg 2021;387:114128. http://dx.doi.org/10.1016/j.cma.2021.114128.

[17] Planas R, Oune N, Bostanabad R. Evolutionary Gaussian processes. J Mech Des 2021;143(11):111703. http://dx.doi.org/10.1115/1.405074.

[18] Arendt Paula D, Apley Daniela W, Chen Wei, Lamb David, Gorsich David. Improving identifiability in model calibration using multiple responses. J Mech Des 2012;134(10):100909.

[19] Arendt Paula D, Apley Daniela W, Chen Wei. Quantification of model uncertainty: Calibration, model discrepancy, and identifiability. J Mech Des 2012;134(10):100908. http://dx.doi.org/10.1115/1.4007390.

[20] Loeppky J, Bingham Derek, Welch W. Computer model calibration or tuning in practice. University of British Columbia, Vancouver, BC, Canada; 2006, Citeseer.

[21] Bayarri MJ, Berger JO, Liu F. Modularization in Bayesian analysis, with emphasis on analysis of computer models. Bayesian Anal 2009;4(1):119–50. http://dx.doi.org/10.1214/09-ba404.

[22] Kennedy Marca C, O'Hagan Anthony. Bayesian calibration of computer models. J R Stat Soc Ser B Stat Methodol 2001;63(3):425–64.

[23] Smith Ralpha C. Uncertainty quantification: theory, implementation, and applications, vol. 12, Siam; 2013.

[24] Fernández-Godino MGiselle, Park Chanyoung, Kim Nam-Ho, Haftka Raphaela T. Review of multi-fidelity models. 2016, arXiv preprint arXiv:1609.07196.

[25] Cutajar Kurt, Pullin Mark, Damianou Andreas, Lawrence Neil, González Javier. Deep Gaussian processes for multi-fidelity modeling. 2019, arXiv preprint arXiv:1903.07320.

[26] Eweis-Labolle Jonathana Tammer, Oune Nicholas, Bostanabad Ramin. Data fusion with latent map Gaussian processes. J Mech Des 2022;144(9). http://dx.doi.org/10.1115/1.4054520.

[27] Deng Shiguang, Mora Carlos, Apelian Diran, Bostanabad Ramin. Data-driven calibration of multifidelity multiscale fracture models via latent map Gaussian process. J Mech Des 2023;145(1):1–15. http://dx.doi.org/10.1115/1.4055951.

[28] Zhang WZ, Bostanabad R, Liang B, Su XM, Zeng D, Bessa MA, Wang YC, Chen W, Cao J. A numerical Bayesian-calibrated characterization method for multiscale prepreg preforming simulations with tension-shear coupling. Compos Sci Technol 2019;170:15–24. http://dx.doi.org/10.1016/j.compscitech.2018.11.019.

[29] Matthews Alexanderb Gb dea G, Vanb Dera Wilk Mark, Nickson Tom, Fujii Keisuke, Boukouvalas Alexis, León-Villagrá Pablo, Ghahramani Zoubin, Hensman James. Gpflow: A Gaussian process library using TensorFlow. J Mach Learn Res 2017;18(40):1–6.

[30] Gardner Jacoba R, Pleiss Geoff, Bindel David, Weinberger Kiliana Q, Wilson Andrewa Gordon. Gpytorch: Blackbox matrix-matrix Gaussian process inference with gpu acceleration. 2018, arXiv preprint arXiv:1809.11165.

[31] Ulaganathan S, Couckuyt I, Dhaene T, Degroote J, Laermans E. High dimensional kriging metamodelling utilising gradient information. Appl Math Model 2016;40(9–10):5256–70. http://dx.doi.org/10.1016/j.apm.2015.12.033.

[32] Bouhlel Mohameda Amine, Martins Joaquima RRA. Gradient-enhanced kriging for high-dimensional problems. 2017, arXiv preprint arXiv:1708.02663.

[33] Thimmisetty Charanraja A, Ghanem Rogera G, White Joshuaa A, Chen Xiao. High-dimensional intrinsic interpolation using Gaussian process regression and diffusion maps. Math Geosci 2017;50(1):77–96. http://dx.doi.org/10.1007/s11004-017-9705-y.

[34] Tripathy Rohit, Bilionis Ilias, Gonzalez Marcial. Gaussian processes with built-in dimensionality reduction: Applications to high-dimensional uncertainty propagation. J Comput Phys 2016;321:191–223.

[35] Giovanis Dimitrisa G, Shields Michaela D. Data-driven surrogates for high dimensional models using Gaussian process regression on the grassmann manifold. Comput Methods Appl Mech Engrg 2020;370:113269.

[36] Damianou Andreas, Lawrence Neil. Deep Gaussian processes. In: Artificial intelligence and statistics. p. 207–15.

[37] Hensman James, Fusi Nicolo, Lawrence Neila D. Gaussian processes for big data. 2013, arXiv preprint arXiv:1309.6835.

[38] Gramacy Robert B, Apley Daniela W. Local Gaussian process approximation for large computer experiments. J Comput Graph Statist 2015;24(2):561–78. http://dx.doi.org/10.1080/10618600.2014.914442.

[39] Guhaniyogi R, Banerjee S. Meta-kriging: Scalable Bayesian modeling and inference for massive spatial datasets. Technometrics 2018;60(4):430–44. http://dx.doi.org/10.1080/00401706.2018.1437474.

[40] Park C, Apley D. Patchwork kriging for large-scale Gaussian process regression. J Mach Learn Res 2018;19(1):269–311.

[41] Liu H, Ong YS, Shen X, Cai J. When Gaussian process meets big data: A review of scalable GPs. IEEE Trans Neural Netw Learn Syst 2020;31(11):4405–23. http://dx.doi.org/10.1109/TNNLS.2019.2957109.

[42] Conference paper. 2021.

[43] Wang Liwei, Yerramilli Suraj, Iyer Akshay, Apley Daniel, Zhu Ping, Chen Wei. Scalable Gaussian processes for data-driven design using big data with categorical factors. J Mech Des 2021;144(2). http://dx.doi.org/10.1115/1.4052221.

[44] Gramacy Robert B, Lee Herbert KH. Bayesian treed Gaussian process models with an application to computer modeling. J Amer Statist Assoc 2012;103(483):1119–30. http://dx.doi.org/10.1198/016214508000000689.

[45] Zhang Yichi, Tao Siyu, Chen Wei, Apley Daniela W. A latent variable approach to Gaussian process modeling with qualitative and quantitative factors. Technometrics 2019;62(3):291–302. http://dx.doi.org/10.1080/00401706.2019.1638834.

[46] Zhang Qiong, Chien Peter, Liu Qing, Xu Li, Hong Yili. Mixed-input Gaussian process emulators for computer experiments with a large number of categorical levels. J Qual Technol 2020;1–11. http://dx.doi.org/10.1080/00224065.2020.

[21] Adaptive Moment Estimation.

[47] Roustant Olivier, Padonou Esperan, Deville Yves, Clément Aloï s, Perrin Guillaume, Giorla Jean, Wynn Henry. Group kernels for Gaussian process metamodels with categorical inputs. SIAM/ASA J Uncertain Quant 2020;8(2):775–806.

[48] Qian Peterb ZG, Wu Huaiqing, Wu CFJeff. Gaussian process models for computer experiments with qualitative and quantitative factors. Technometrics 2008;50(3):383–96.

[49] Mobahi Hossein, Fishera III Johna W. A theoretical analysis of optimization by Gaussian continuation. In: Twenty-ninth AAAI conference on artificial intelligence.

[50] Bonilla Edwina V, Chai Kian, Williams Christopher. Multi-task Gaussian process prediction. Adv Neural Inf Process Syst 2007;20.

[51] Conti S, Gosling JP, Oakley JE, O'Hagan A. Gaussian process emulation of dynamic computer codes. Biometrika 2009;96(3):663–76. http://dx.doi.org/10.1093/biomet/asp028.

[52] Conti Stefano, O'Hagan Anthony. Bayesian emulation of complex multi-output and dynamic computer models. J Statist Plann Inference 2010;140(3):640–51. http://dx.doi.org/10.1016/j.jspi.2009.08.006.

[53] Bernardo J, Berger J, Dawid APAFMS, Smith A. Regression and classification using Gaussian process priors. Bayes Statist 1998;6:475.

[54] MacKay Davida JC. Introduction to Gaussian processes. NATO ASI Ser F Comput Syst Sci 1998;168:133–66.

[55] Gramacy Robert B, Lee Herbert KH. Cases for the nugget in modeling computer experiments. Stat Comput 2010;22(3):713–22. http://dx.doi.org/10.1007/s11222-010-9224-x.

[56] Bostanabad Ramin, Kearney Tucker, Tao Siyu, Apley Daniela W, Chen Wei. Leveraging the nugget parameter for efficient Gaussian process modeling. Int J Numer Methods Eng 2018;114(5):501–16.

[57] MacDonald Blake, Ranjan Pritam, Chipman Hugh. GPfit: An R package for fitting a Gaussian process model to deterministic simulator outputs. J Stat Softw 2015;64:1–23.

[58] O'Hagan A. Curve fitting and optimal design for prediction. J R Stat Soc Ser B Stat Methodol 1978;40(1):1–24. http://dx.doi.org/10.1111/j.2517-6161.1978.tb01643.x.

[59] Murphy Kevina P. Machine learning: a probabilistic perspective. MIT Press; 2012.

[60] Gramacy Robert B. tgp: an R package for Bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models. J Stat Softw 2007;19:1–46.

[61] Chipman HA, George EI, McCulloch RE, Zhang H, Knight K, Kustra R. Bayesian CART model search. Commentaries. Authors' reply. J Amer Statist Assoc 1998;93(443):935–60.

[62] Härkönen Teemu, Wade Sara, Law Kody, Roininen Lassi. Mixtures of Gaussian process experts with SMC^2. 2022, arXiv preprint arXiv:2208.12830.

[63] Candelieri Antonio, Pedrielli Giulia. Treed-Gaussian processes with support vector machines as nodes for nonstationary Bayesian optimization. In: 2021 winter simulation conference. WSC, IEEE; 2021, p. 1–12.

[64] GPy. GPy: A Gaussian process framework in python. 2012, http://github.com/SheffieldML/GPy.

[65] Azevedo-Filho Adriano, Shachter Rossa D. Laplace's method approximations for probabilistic inferencein belief networks with continuous variables. In: Proceedings of the tenth international conference on uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc.; p. 28–36.

[66] Wilson Andrewa Gordon, Hu Zhiting, Salakhutdinov Ruslan, Xing Erica P. Deep kernel learning. In: Artificial intelligence and statistics. PMLR; p. 370–8.

[67] Titsias Michalis, Lawrence Neila D. Bayesian Gaussian process latent variable model. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop Conference Proceedings; p. 844–51.

[68] Lawrence Neil. Gaussian process latent variable models for visualisation of high dimensional data. Adv Neural Inf Process Syst 2003;16.

[69] Bingham Eli, Chen Jonathana P, Jankowiak Martin, Obermeyer Fritz, Pradhan Neeraj, Karaletsos Theofanis, Singh Rohit, Szerlip Paul, Horsfall Paul, Goodman Noaha D. Pyro: Deep Universal Probabilistic Programming. J Mach Learn Res 2018.

[70] Ambikasaran Sivaram, Foreman-Mackey Daniel, Greengard Leslie, Hogg Davida W, O'Neil Michael. Fast direct methods for Gaussian processes and the analysis of NASA Kepler mission data. 2014, arXiv preprint arXiv:1403.6015.

[71] Vanhatalo Jarno, Riihimäki Jaakko, Hartikainen Jouni, Jylänki Pasi, Tolvanen Ville, Vehtari Aki. Gpstuff: Bayesian modeling with Gaussian processes. J Mach Learn Res 2013;14(1):1175–9.

[72] Hensman James, Fusi Nicolo, Lawrence Neila D. Gaussian processes for big data. 2013, arXiv preprint arXiv:1309.6835.

[73] Bengio Yoshua, Delalleau Olivier, Roux Nicolas. The curse of highly variable functions for local kernel machines. Adv Neural Inf Process Syst 2005;18.

[74] Eweis-Labolle Jonathana Tammer, Oune Nicholas, Bostanabad Ramin. Data fusion with latent map Gaussian processes. J Mech Des 2022;144(9):091703.

[75] Foumani Zahraa Zanjani, Shishehbor Mehdi, Yousefpour Amin, Bostanabad Ramin. Multi-fidelity cost-aware Bayesian optimization. Comput Methods Appl Mech Engrg 2023;407:115937.

[76] Oune Nicholas, Bostanabad Ramin. Latent map Gaussian processes for mixed variable metamodeling. Comput Methods Appl Mech Engrg 2021;387:114128.

[77] Tao Siyu, Apley Daniela W, Plumlee Matthew, Chen Wei. Latent variable Gaussian process models: A rank-based analysis and an alternative approach. Internat J Numer Methods Engrg 2021;122(15):4007–26.

[78] Bonilla Edwina V, Chai Kian, Williams Christopher. Multi-task Gaussian process prediction. Adv Neural Inf Process Syst 2007;20.

[79] Poloczek Matthias, Wang Jialei, Frazier Peter. Multi-information source optimization. Adv Neural Inf Process Syst 2017;30.

[80] Chakraborty Souvik, Chatterjee Tanmoy, Chowdhury Rajib, Adhikari Sondipon. A surrogate based multi-fidelity approach for robust design optimization. Appl Math Model 2017;47:726–44.

[81] Korondi Pétera Zénó, Marchi Mariapia, Parussini Lucia, Poloni Carlo. Multi-fidelity design optimisation strategy under uncertainty with limited computational budget. Optim Eng 2021;22:1039–64.

[82] Dixon Thomasa O, Warner Jamesa E, Bomarito Geoffreya F, Gorodetsky Alexa A. Covariance expressions for multi-fidelity sampling with multi-output, multi-statistic estimators: Application to approximate control variates. 2023, arXiv preprint arXiv:2310.00125.

[83] Absi Ghinaa N, Mahadevan Sankaran. Multi-fidelity approach to dynamics model calibration. Mech Syst Signal Process 2016;68:189–206.

[84] Sobol' Il'yaa Meerovich. On sensitivity estimation for nonlinear mathematical models. Mat Model 1990;2(1):112–8.

[85] Gorodetsky AA, Jakeman JD, Geraci G. MFNets: data efficient all-at-once learning of multifidelity surrogates as directed networks of information sources. Comput Mech 2021;68(4):741–58. http://dx.doi.org/10.1007/s00466-021-02042-0.

[86] Mora Carlos, Eweis-Labolle Jonathana Tammer, Johnson Tyler, Gadde Likith, Bostanabad Ramin. Probabilistic neural data fusion for learning from an arbitrary number of multi-fidelity data sets. Comput Methods Appl Mech Engrg 2023;415:116207. http://dx.doi.org/10.1016/j.cma.2023.116207.

[87] Tuo Rui, Wu CF. Prediction based on the kennedy-o'hagan calibration model: asymptotic consistency and other properties. 2017, arXiv preprint arXiv:1703.01326.

[88] Qian Peterb ZG, Wu CFJeff. Bayesian hierarchical modeling for integrating low-accuracy and high-accuracy experiments. Technometrics 2008;50(2):192–204. http://dx.doi.org/10.1198/004017008000000082.

[89] McFarland John, Mahadevan Sankaran, Romero Vicente, Swiler Laura. Calibration and uncertainty analysis for computer simulations with multivariate output. AIAA J 2008;46(5):1253–65. http://dx.doi.org/10.2514/1.35288.

[90] Bayarri MJ, Walsh D, Berger JO, Cafeo J, Garcia-Donato G, Liu F, Palomo J, Parthasarathy RJ, Paulo R, Sacks J. Computer model validation with functional output. Ann Statist 2007;35(5):1874–906. http://dx.doi.org/10.1214/009053607000000163.

[91] Higdon Dave, Kennedy Marc, Cavendish Jamesa C, Cafeo Johna A, Ryne Robert D. Combining field data and computer simulations for calibration and prediction. SIAM J Sci Comput 2004;26(2):448–66. http://dx.doi.org/10.1137/s1064827503426693.

[92] Zanjania Foumani Zahra, Yousefpour Amin, Shishehbor Mehdi, Bostanabad Ramin. Safeguarding multi-fidelity Bayesian optimization against large model form errors and heterogeneous noise. J Mech Des 2023;1–23.

[93] Schaden Daniel, Ullmann Elisabeth. On multilevel best linear unbiased estimators. SIAM/ASA J Uncertain Quant 2020;8(2):601–35.

[94] Gorodetsky Alexa A, Geraci Gianluca, Eldred Michaela S, Jakeman Johna D. A generalized approximate control variate framework for multifidelity uncertainty quantification. J Comput Phys 2020;408:109257.

[95] Ba Shan, Joseph VRoshan. Composite Gaussian process models for emulating expensive functions. Ann Appl Stat 2012;6(4):1838–60. http://dx.doi.org/10.1214/12-aoas570.

[96] Le Quoca V, Smola Alexa J, Canu Stéphane. Heteroscedastic Gaussian process regression. In: Proceedings of the 22nd international conference on machine learning. ACM; p. 489–96.

[97] Kingma Diederika P, Welling Max. Auto-encoding variational bayes. 2013, arXiv preprint arXiv:1312.6114.

[98] Wolpert Robert L. Conditional expectation. In: University lecture. 2010.

[99] Rudary Matthewa R. On predictive linear Gaussian models. University of Michigan; 2009.

[100] Lee Jaehoon, Bahri Yasaman, Novak Roman, Schoenholz Samuela S, Pennington Jeffrey, Sohl-Dickstein Jascha. Deep neural networks as gaussian processes. 2017, arXiv preprint arXiv:1711.00165.

[101] Al-Shedivat Maruan, Wilson Andrewa Gordon, Saatchi Yunus, Hu Zhiting, Xing Erica P. Learning scalable deep kernels with recurrent structure. J Mach Learn Res 2017;18(1):2850–86.

[102] Planas Robert, Oune Nick, Bostanabad Ramin. Evolutionary Gaussian processes. J Mech Des 2021;143(11):111703.

[103] Belytschko Ted, Liu Winga Kam, Moran Brian, Elkhodary Khalil. Nonlinear finite elements for continua and structures. John Wiley & Sons; 2013.

[104] Zhang Weizhao, Ren Huaqing, Wang Zequn, Liu Winga K, Chen Wei, Zeng Danielle, Su Xuming, Cao Jian. An integrated computational materials engineering method for woven carbon fiber composites preforming process. AIP Conf Proc 2016;1769(1):170036. http://dx.doi.org/10.1063/1.4963592.

[105] Botelho EC, Figiel Ł, Rezende MC, Lauke B. Mechanical behavior of carbon fiber reinforced polyamide composites. Compos Sci Technol 2003;63(13):1843–55. http://dx.doi.org/10.1016/S0266-3538(03)00119-2.
[106] Gao Jiaying, Shakoor Modesar, Domel Gino, Merzkirch Matthias, Zhou Guowei, Zeng Danielle, Su Xuming, Liu Winga Kam. Predictive multiscale modeling for unidirectional carbon fiber reinforced polymers. Compos Sci Technol 2020;186:107922. http://dx.doi.org/10.1016/j.compscitech.2019.107922.
[107] Deng SG, Soderhjelm C, Apelian D, Bostanabad R. Reduced-order multiscale modeling of plastic deformations in 3D alloys with spatially varying porosity by deflated clustering analysis. Comput Mech 2022;1–32. http://dx.doi.org/10.1007/s00466-022-02177-8.
[108] Deng Shiguang, Apelian Diran, Bostanabad Ramin. Adaptive spatiotemporal dimension reduction in concurrent multiscale damage analysis. Comput Mech 2023. http://dx.doi.org/10.1007/s00466-023-02299-7.
[109] Dvorak Georgea J. Transformation field analysis of inelastic composite materials. Proc R Soc Lond Ser A Math Phys Eng Sci 1992;437(1900):311–27.
[110] Roussette Sophie, Michel Jean-Claude, Suquet Pierre. Nonuniform transformation field analysis of elastic–viscoplastic composites. Compos Sci Technol 2009;69(1):22–7.
[111] Tuo Rui, Jeffa Wu CF. A theoretical framework for calibration in computer models: Parametrization, estimation and convergence properties. SIAM/ASA J Uncertain Quant 2016;4(1):767–95. http://dx.doi.org/10.1137/151005841.
[112] Plumlee Matthew. Bayesian calibration of inexact computer models. J Amer Statist Assoc 2016. just-accepted.
[113] Bayarri Mariaa J, Berger Jamesa O, Paulo Rui, Sacks Jerry, Cafeo Johna A, Cavendish James, Lin Chin-Hsu, Tu Jian. A framework for validation of computer models. Technometrics 2007;49(2):138–54. http://dx.doi.org/10.1198/004017007000000092.
[114] Apley Daniela W, Liu Jun, Chen Wei. Understanding the effects of model uncertainty in robust design with computer experiments. 2006.
[115] Oakley J. Bayesian inference for the uncertainty distribution of computer model outputs. Biometrika 2002;89(4):769–84. http://dx.doi.org/10.1093/biomet/89.4.769.
[116] Arendt Paula D, Apley Daniela W, Chen Wei. A preposterior analysis to predict identifiability in the experimental calibration of computer models. IIE Trans 2016;48(1):75–88.
[117] Carvalho Carlosa M, Polson Nicholasa G, Scott Jamesa G. The horseshoe estimator for sparse signals. Biometrika 2010;97(2):465–80.
[118] Mathworks. Gaussian process regression models. 2023.
[119] Balandat Maximilian, Karrer Brian, Jiang Daniela R, Daulton Samuel, Letham Benjamin, Wilson Andrewa Gordon, Bakshy Eytan. Botorch models. 2020.
[120] Saves Paul, Lafage Rémi, Bartoli Nathalie, Diouane Youssef, Bussemaker Jasper, Lefebvre Thierry, Hwang Johna T, Morlier Joseph, Martins Joaquima RRA. SMT 2.0: A surrogate modeling toolbox with a focus on hierarchical and mixed variables Gaussian processes. Adv Eng Softw 2024;188:103571.
[121] Zhou Qiang, Qian Petera ZG, Zhou Shiyu. A simple approach to emulation for computer models with qualitative and quantitative factors. Technometrics 2011;266–73.
[122] Saves Paul, Diouane Youssef, Bartoli Nathalie, Lefebvre Thierry, Morlier Joseph. A mixed-categorical correlation kernel for Gaussian process. Neurocomputing 2023;126472.
[123] Halstrup Momchil. Black-box optimization of mixed discrete-continuous optimization problems. 2016.
[124] Bostanabad R, Kearney T, Tao SY, Apley DW, Chen W. Leveraging the nugget parameter for efficient Gaussian process modeling. Internat J Numer Methods Engrg 2018;114(5):501–16. http://dx.doi.org/10.1002/nme.5751.
[125] Deng Shiguang, Mora Carlos, Apelian Diran, Bostanabad Ramin. Data-driven calibration of multifidelity multiscale fracture models via latent map Gaussian process. J Mech Des 2023;145(1):011705.
[126] Marelli Stefano, Sudret Bruno. UQLab: A framework for uncertainty quantification in Matlab. In: Vulnerability, uncertainty, and risk: quantification, mitigation, and management. 2014, p. 2554–63.
[127] Turner Ryan, Eriksson David, McCourt Michael, Kiili Juha, Laaksonen Eero, Xu Zhen, Guyon Isabelle. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In: NeurIPS 2020 competition and demonstration track. PMLR; 2021, p. 3–26.
[128] Frazier Petera I, Wang Jialei. Bayesian optimization for materials design. In: Information science for materials discovery and design. Springer; 2015, p. 45–75.
[129] Couckuyt Ivo, Gonzalez Sebastiana Rojas, Branke Juergen. Bayesian optimization: tutorial. In: Proceedings of the genetic and evolutionary computation conference companion. 2022, p. 843–63.
[130] Nguyen Loc. Tutorial on Bayesian optimization. 2023.
[131] Brochu Eric, Cora Vlada M, Dea Freitas Nando. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010, arXiv preprint arXiv:1012.2599.
[132] Kopsiaftis George, Protopapadakis Eftychios, Voulodimos Athanasios, Doulamis Nikolaos, Mantoglou Aristotelis. Gaussian process regression tuned by Bayesian optimization for seawater intrusion prediction. Comput Intell Neurosci 2019;2019.
[133] Binois Mickael, Wycoff Nathan. A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization. ACM Trans Evol Learn Optim 2022;2(2):1–26.
[134] Sui Yanan, Zhuang Vincent, Burdick Joel, Yue Yisong. Stagewise safe Bayesian optimization with gaussian processes. In: International conference on machine learning. PMLR; 2018, p. 4781–9.
[135] Morita Yuki, Rezaeiravesh Saleh, Tabatabaei Narges, Vinuesa Ricardo, Fukagata Koji, Schlatter Philipp. Applying Bayesian optimization with Gaussian process regression to computational fluid dynamics problems. J Comput Phys 2022;449:110788.
[136] McIntire Mitchell, Ratner Daniel, Ermon Stefano. Sparse Gaussian processes for Bayesian optimization. In: UAI. 2016.
[137] Rana Santu, Li Cheng, Gupta Sunil, Nguyen Vu, Venkatesh Svetha. High dimensional Bayesian optimization with elastic Gaussian process. In: International conference on machine learning. PMLR; 2017, p. 2883–91.
[138] Egger Davida A, Rappe Andrewa M, Kronik Leeor. Hybrid organic–inorganic perovskites on the move. Acc Chem Res 2016;49(3):573–81.
[139] Rao Ziyuan, Tung Po-Yen, Xie Ruiwen, Wei Ye, Zhang Hongbin, Ferrari Alberto, Klaver TPC, Körmann Fritz, Sukumar Prithiva Thoudden, Kwiatkowskib daa Silva Alisson. Machine learning–enabled high-entropy alloy discovery. Science 2022;378(6615):78–85.
[140] Cover MF, Warschkow O, Bilek MMM, McKenzie DR. A comprehensive survey of M2AX phase elastic properties. J Phys: Condens Matter 2009;21(30):305403.
[141] Saltelli Andrea, Annoni Paola, Azzini Ivano, Campolongo Francesca, Ratto Marco, Tarantola Stefano. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. Comput Phys Commun 2010;181(2):259–70.
[142] Renganathan Sa Ashwin, Rao Vishwas, Navon Ionela M. CAMERA: A method for cost-aware, adaptive, multifidelity, efficient reliability analysis. 2022, arXiv preprint arXiv:2203.01436.
[143] Bracher Johannes, Ray Evana L, Gneiting Tilmann, Reich Nicholasa G. Evaluating epidemic forecasts in an interval format. PLoS Comput Biol 2021;17(2):e1008618.
[144] Mitchell K, Ferro CAT. Proper scoring rules for interval probabilistic forecasts. Q J R Meteorol Soc 2017;143(704):1597–607.
[145] Frazier Petera I, Powell Warrena B, Dayanik Savas. A knowledge-gradient policy for sequential information collection. SIAM J Control Optim 2008;47(5):2410–39.
[146] Balandat Maximilian, Karrer Brian, Jiang Daniel, Daulton Samuel, Letham Ben, Wilson Andrewa G, Bakshy Eytan. BoTorch: a framework for efficient Monte-Carlo Bayesian optimization. Adv Neural Inf Process Syst 2020;33:21524–38.