Gaussian processes for predictive modeling in the low-data regime in materials science

Gonzalo Ramírez López-Osa
(Dated: June 17, 2025)

Gaussian Processes (GPs) have emerged as powerful predictive modeling tools in a variety of scientific contexts, particularly valuable due to their non-parametric nature, inherent uncertainty quantification, and flexibility in modeling complex functions. This Master thesis explores the application of Gaussian Processes—both Full and Sparse variants—to predictive modeling problems in materials science, focusing on two representative systems: a simplified water molecule model and a large-scale dataset from methane oxidative coupling reactions. Initially, a comprehensive theoretical framework of Gaussian Processes is established, highlighting critical components such as kernel functions and hyperparameter optimization methods. Subsequently, we examine Full Gaussian Processes on a small dataset derived from quantum simulations of the water molecule, evaluating different kernels' performance and robustness under data scarcity. The analysis reveals excellent predictive capabilities, emphasizing the importance of kernel choice and the GPs' stability even with limited data. Next, we shift focus to Sparse Variational Gaussian Processes (SVGPs), analyzing their effectiveness on a larger, high-dimensional catalytic dataset. Through systematic benchmarking against Full GPs and other established regression techniques, such as Random Forests and XGBoost, the thesis demonstrates the computational efficiency and competitive predictive accuracy of SVGPs. Although Sparse GPs slightly compromise predictive accuracy compared to Full GPs, they provide substantial computational advantages for large datasets. Overall, this thesis validates Gaussian Processes as versatile and robust modeling approaches in materials science, capable of balancing predictive performance with computational feasibility across diverse datasets and applications.

I. INTRODUCTION

The rational design and accelerated discovery of novel materials and molecules are central to solving pressing technological challenges in areas such as energy storage, environmental sustainability, electronics, and biomedicine. However, these efforts are often hindered by the intrinsic complexity of material systems and the vastness of their chemical and structural design spaces. Exhaustively sampling these high-dimensional spaces through experiments or simulations remains impractical, underscoring the need for efficient and scalable modeling approaches [1].

Computational materials science, particularly through quantum mechanical simulations and high-throughput screening protocols, has significantly expanded our ability to predict material properties from first principles. Yet, such approaches incur steep computational costs when applied to large systems, disordered materials, or iterative tasks such as global optimization and uncertainty quantification [2]. As a result, there is increasing demand for data-driven surrogate models that retain accuracy while offering improved scalability.

Machine learning (ML) has emerged as a compelling framework for this purpose [3, 4]. Among ML methods, Gaussian Process Regression (GPR), or Gaussian Processes (GPs), occupies a unique position due to its non-parametric nature, well-defined probabilistic interpretation, and strong performance in the low-data regimes typical of materials science [2, 5, 6]. Unlike black-box models, GPs yield both predictive means and calibrated uncertainties, making them particularly well-suited for risk-aware applications such as Bayesian Optimization (BO), active learning, and materials screening [7, 8].

In recent years, GPs have been employed for a wide range of atomistic and molecular tasks, including the prediction of potential energies, forces, dipole moments, and electronic structure observables. The Gaussian Approximation Potential (GAP) framework, in particular, has demonstrated that GPs can serve as accurate surrogates to Density Functional Theory (DFT) for modeling complex systems, including amorphous solids and multicomponent alloys [9, 10].

Despite these successes, the applicability of GPs to large-scale problems is constrained by their computational cost, which scales cubically with the number of training points ($\mathcal{O}(N^3)$). To mitigate this bottleneck, Sparse approximations such as Sparse Variational Gaussian Processes (SVGPs) have been developed. Most of these methods introduce a tractable set of $M \ll N$ inducing variables to approximate the full GP Posterior, reducing the training cost to $\mathcal{O}(NM^2)$ while preserving key uncertainty quantification capabilities [11, 12].

High dimensionality poses another significant challenge for GPs. As the number of input features grows, the volume of the input space expands exponentially—a phenomenon often referred to as the "curse of dimensionality" [13]. In such cases, kernel-based methods may struggle to maintain predictive accuracy due to deteriorating distance metrics and overfitting to irrelevant input dimensions [14]. Kernel design, dimensionality reduction, and Automatic Relevance Determination (ARD) therefore play critical roles in maintaining model performance [15].

Beyond regression, GPs are foundational to Bayesian

Optimization (BO) frameworks, where they serve as surrogates for computationally expensive black-box objective functions [7, 8]. BO leverages both the mean and uncertainty of the GP Posterior to sequentially select candidate points for evaluation, enabling data-efficient global optimization [7, 8]. This combination of interpretability and probabilistic rigor makes GPs particularly attractive for guiding materials discovery and experimental design.

This Master thesis investigates the application of both Full and Sparse Gaussian Processes to regression tasks in materials science, focusing on two limiting cases: small data and high-dimensionality. The first case involves a low-dimensional system derived from a simplified water molecule model [2], used to analyze the influence of kernel choice and training dataset size on GP performance. The second case examines a high-dimensional dataset describing oxidative coupling of methane (OCM) reactions [16], consisting of over 4000 samples and 289 features, and is used to benchmark SVGPs against established ensemble-based ML models.

The primary objectives are to (i) characterize the trade-offs between Full and Sparse GPs in terms of predictive accuracy and computational efficiency across varying regimes, and (ii) compare their performance with widely-used alternatives such as Random Forests and XGBoost [17–20]. The results demonstrate that Full GPs achieve superior accuracy and uncertainty quantification in small-data settings, while SVGPs offer favorable scalability and competitive performance in high-dimensional contexts. Additionally, the study highlights the importance of kernel choice, with the Matern class exhibiting greater robustness than the RBF kernel in data-scarce regimes. Ensemble models like XGBoost, while effective in predictive accuracy, do not inherently provide uncertainty estimates, limiting their utility in uncertainty-sensitive applications.

The remainder of this Master thesis is structured as follows. Section II introduces the theoretical foundations of Gaussian Processes, including weight-space and function-space perspectives, and describes hyperparameter optimization strategies. Section III outlines the datasets and preprocessing techniques. Section IV(A) presents the water molecule case study, emphasizing model calibration and data efficiency. Section IV(B) evaluates SVGPs on the OCM dataset, with comparative analysis against the other considered predictive models. Section V concludes with key findings and future directions for scalable GP-based modeling in materials science.

II. THEORETICAL FOUNDATIONS OF GAUSSIAN PROCESSES

Gaussian Processes (GPs) provide a principled, flexible framework for non-parametric Bayesian regression. Formally, a GP is defined as a collection of random variables, any finite number of which follows a joint Gaussian distribution [21]. A GP is fully specified by its mean function, $m(\boldsymbol{x})$, and covariance function, or kernel, $k(\boldsymbol{x}, \boldsymbol{x}')$, such that:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')). \tag{1}$$

The mean function encodes prior expectations about the global trend of the target function, while the kernel captures statistical correlations between function values at different input points, and thus governs the expressivity of the model [21]. In most practical settings, the mean is set to zero without loss of generality, as deviations can be modeled via the kernel alone [2, 5, 21].

Among the two components, the kernel plays a central role in GP modeling. It encodes prior assumptions about smoothness, periodicity, and other functional properties of the target, and it determines the shape and complexity of the functions that the GP can model [5, 21]. The kernel function depends on a set of hyperparameters that govern properties such as length scale, amplitude, and noise level. These hyperparameters are typically optimized via marginal likelihood maximization or cross-validation to best match the training data [21].

Once the GP has been trained, predictions at unseen inputs can be made in closed form. These predictions are themselves Gaussian distributed and defined by a Posterior mean and variance, which quantify both the expected function value and its uncertainty. This built-in uncertainty quantification makes GPs particularly attractive in applications such as Bayesian optimization and active learning [7, 8].

Despite their conceptual elegance and strong predictive performance in low-data regimes, the application of GPs to large datasets is computationally challenging. The training process of an exact GP, often called a Full GP, requires inversion of an $N \times N$ covariance matrix, leading to a computational cost that scales as $\mathcal{O}(N^3)$ with the number of training points N [11, 12]. This cubic complexity renders Full GPs impractical for datasets larger than a few thousand samples.

To address this, Sparse Gaussian Processes (SGPs) have been developed. These methods introduce a set of $M \ll N$ inducing variables (or representative points) that serve as a low-rank approximation to the full covariance structure [11, 22]. The resulting approximate posterior distribution can be computed at a reduced cost of $\mathcal{O}(NM^2)$, making SGPs suitable for large-scale problems [11, 12]. While this approximation introduces a trade-off in terms of predictive accuracy, empirical results show that it remains effective, particularly when the number of inducing points is chosen appropriately.

Figure 1 illustrates the predictive performance of both Full and Sparse GP models on the function $y = \sin(\pi x/5)$. As expected, the Full GP closely matches the target function across the domain. The Sparse GP model improves progressively as more inducing points are added, demonstrating convergence toward the Full

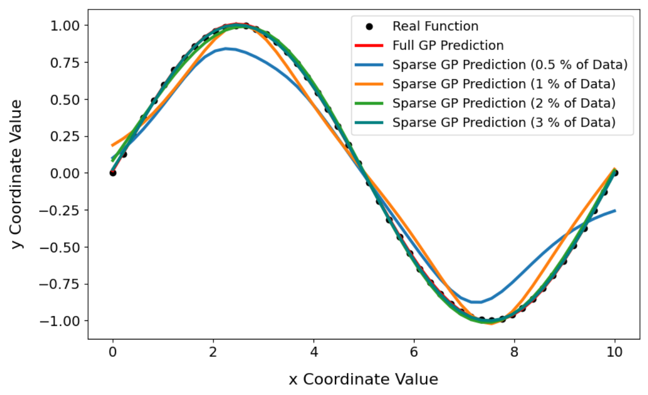


FIG. 1. Modeling of the function $y = \sin(\frac{\pi x}{5})$ using a Full GP and a Sparse GP with varying percentages of representative points. The Full GP was trained on 1000 points in the interval $[0, 50]$. As the number of inducing points increases, the Sparse GP approaches the accuracy of the Full GP.

GP behavior with just a small fraction (e.g., 3%) of representative data.

The predictive framework of GPs can be derived from two complementary perspectives: the weight-space view and the function-space view [2, 5, 21]. The weight-space view assumes a fixed basis function expansion and derives the posterior by optimizing weights, offering an intuitive and deterministic picture of learning. In contrast, the function-space view is more general and describes the GP directly as a distribution over functions, which is analytically tractable under the Gaussian prior. This perspective underpins the derivation of the Sparse Variational Gaussian Process (SVGP), the Sparse model used in this thesis [22].

In both views, it is common practice to assume a zero-mean prior. If the true function has a non-zero mean, it can be removed by preprocessing the data—subtracting the empirical mean or a parametric estimate—before training, and then added back to the posterior mean after prediction. Mathematically, this is equivalent to defining a new target function $g(\boldsymbol{x}) = f(\boldsymbol{x}) - m(\boldsymbol{x})$, where $m(\boldsymbol{x})$ is a known, fixed mean function [2, 5, 21]. This technique simplifies the mathematics while preserving model expressiveness, and is valid as long as $m(\boldsymbol{x})$ has no trainable parameters.

A. Predictive Process

The predictive process in Gaussian Processes (GPs) refers to the derivation of the Posterior distribution over function values at new inputs, conditioned on a set of observed data [21]. This section presents two complementary perspectives for understanding and computing GP predictions: the weight-space view, which provides an interpretable, parametric approximation, and the function-space view, which offers a fully probabilistic, non-parametric formulation. Together, these frameworks form the foundation for both exact and approximate inference schemes used in this work.

1. *Weight Space Perspective*

In the weight-space formulation of Gaussian Processes (GPs), we approximate the function $y(\boldsymbol{x})$ using a finite linear combination of basis functions centered at a subset of M representative points (inducing inputs), $\{\boldsymbol{x}_m\}_{m=1}^{M}$, typically selected from the training dataset. The predictive function takes the form:

$$\widetilde{y}(\boldsymbol{x}) = \sum_{m=1}^{M} c_m \, k(\boldsymbol{x}, \boldsymbol{x}_m), \tag{2}$$

where $k(\boldsymbol{x}, \boldsymbol{x}_m)$ denotes the kernel evaluated between the input and an inducing point, and c_m are the weights to be optimized. To obtain these coefficients, we minimize a regularized least-squares loss function:

$$L = \sum_{n=1}^{N} \frac{[y_n - \widetilde{y}(\boldsymbol{x}_n)]^2}{\sigma_n^2} + \sum_{m,m'=1}^{M} c_m \, k(\boldsymbol{x}_m, \boldsymbol{x}_{m'}) \, c_{m'}, \tag{3}$$

where σ_n^2 denotes a noise variance that can be fixed or learned. The first term ensures data fidelity, while the second introduces Tikhonov regularization that penalizes large coefficients, thereby enforcing smoothness and preventing overfitting.

Assuming homoscedastic noise ($\sigma_n = \sigma$), the loss simplifies to:

$$L = \sum_{n=1}^{N} [y_n - \widetilde{y}(\boldsymbol{x}_n)]^2 + \sigma^2 \sum_{m,m'=1}^{M} c_m \, k(\boldsymbol{x}_m, \boldsymbol{x}_{m'}) \, c_{m'}. \tag{4}$$

This can be rewritten compactly in matrix notation as:

$$L = (\boldsymbol{y} - \boldsymbol{K}_{NM}\boldsymbol{c})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{K}_{NM}\boldsymbol{c}) + \boldsymbol{c}^T \boldsymbol{K}_{MM} \boldsymbol{c}, \tag{5}$$

where $\boldsymbol{K}_{NM}$ is the $N \times M$ matrix of cross-kernel evaluations, $\boldsymbol{K}_{MM}$ is the kernel matrix among inducing points, and $\boldsymbol{\Sigma}$ is diagonal with σ^2 on the diagonal. Taking the derivative with respect to $\boldsymbol{c}$ and setting it to zero yields:

$$\boldsymbol{c} = (\boldsymbol{K}_{MM} + \boldsymbol{K}_{MN} \boldsymbol{\Sigma}^{-1} \boldsymbol{K}_{NM})^{-1} \boldsymbol{K}_{MN} \boldsymbol{\Sigma}^{-1} \boldsymbol{y}, \tag{6}$$

which gives the optimal weights. In the special case where $M = N$ and the inducing points are the training data, we recover the exact GP (Full GP) expression:

$$\boldsymbol{c} = (\boldsymbol{K}_{NN} + \boldsymbol{\Sigma})^{-1}\boldsymbol{y}. \tag{7}$$

Once $\boldsymbol{c}$ is known, predictions at any test point $\boldsymbol{x}_*$ are obtained via:

$$\widetilde{y}(\boldsymbol{x}_*) = \boldsymbol{c}^T\boldsymbol{k}(\boldsymbol{x}_*), \tag{8}$$

where $\boldsymbol{k}(\boldsymbol{x}_*)$ is a column vector of kernel evaluations between $\boldsymbol{x}_*$ and the inducing inputs. This weight-space formulation provides an intuitive bridge to sparse approximations, where $M \ll N$.

2. Function Space Perspective

The function-space view of Gaussian Processes offers a complementary probabilistic formulation by interpreting GPs as distributions over functions, rather than weight combinations of basis expansions. Consider a stochastic process where the function value $\widetilde{y}(\boldsymbol{x})$ is expressed as a linear combination of deterministic basis functions $\phi_h(\boldsymbol{x})$:

$$\widetilde{y}(\boldsymbol{x}) = \sum_{h=1}^{H} w_h\, \phi_h(\boldsymbol{x}), \tag{9}$$

where $\{\phi_h\}$ represent fixed, pre-defined basis functions (e.g., polynomials, sinusoids), and the weights $\{w_h\}$ are treated as independent Gaussian random variables drawn from $\mathcal{N}(0, \sigma_w^2)$. This construction induces a Gaussian Prior over functions $\widetilde{y}(\boldsymbol{x})$, with zero mean and covariance given by:

$$\mathbb{E}[\widetilde{y}(\boldsymbol{x})\widetilde{y}(\boldsymbol{x}')] = \sigma_w^2 \sum_{h=1}^{H} \phi_h(\boldsymbol{x})\phi_h(\boldsymbol{x}'), \tag{10}$$

which defines the kernel as a weighted inner product in function space:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_w^2 \sum_{h=1}^{H} \phi_h(\boldsymbol{x})\phi_h(\boldsymbol{x}'). \tag{11}$$

This Gram matrix formulation of the kernel function guarantees positive semi-definiteness and symmetry $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$, fundamental properties required for the Gaussian prior [23].

When noise-corrupted observations are considered, i.e., $y_n = f(\boldsymbol{x}_n) + \epsilon_n$ with $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$, the joint distribution over the observed outputs $\boldsymbol{y} = [y_1, \dots, y_N]^T$ is multivariate Gaussian:

$$P(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{NN} + \sigma^2\boldsymbol{I}), \tag{12}$$

where $\boldsymbol{K}_{NN}$ is the kernel matrix over training inputs. Conditioning on this prior using Bayes' theorem, the predictive distribution at a test point $\boldsymbol{x}_*$ is also Gaussian with the Posterior mean and variance:

$$\mu(\boldsymbol{x}_*) = \boldsymbol{k}_*^T(\boldsymbol{K}_{NN} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}, \tag{13}$$

$$\mathrm{var}(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) + \sigma^2 - \boldsymbol{k}_*^T(\boldsymbol{K}_{NN} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{k}_*, \tag{14}$$

where $\boldsymbol{k}_*$ is the vector of kernel evaluations between $\boldsymbol{x}_*$ and the training inputs. In the noise-free limit ($\sigma^2 \to 0$), these expressions reduce accordingly.

This function-space derivation provides the theoretical basis for exact GPs. However, scaling GPs to large datasets necessitates approximations. The Sparse Variational Gaussian Process (SVGP) reformulates the Posterior via variational inference, introducing a set of M inducing variables $\boldsymbol{f}_m$ at inputs $\boldsymbol{X}_m$, where $M \ll N$ [22]. The true Posterior over test values $\boldsymbol{z}$ is:

$$p(\boldsymbol{z}|\boldsymbol{y}) = \int p(\boldsymbol{z}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{y})\, d\boldsymbol{f}, \tag{15}$$

which is computationally intractable. SVGP introduces a variational distribution $q(\boldsymbol{f}_m) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{A})$ to approximate the Posterior [22], yielding:

$$q(\boldsymbol{z}) = \int p(\boldsymbol{z}|\boldsymbol{f}_m)\, q(\boldsymbol{f}_m)\, d\boldsymbol{f}_m. \tag{16}$$

The predictive Posterior then has closed-form expressions:

$$\mu(\boldsymbol{x}) = \boldsymbol{K}_{xM}\boldsymbol{K}_{MM}^{-1}\boldsymbol{\mu}, \tag{17}$$

$$k'(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{K}_{xM}\boldsymbol{K}_{MM}^{-1}\boldsymbol{K}_{Mx'} \tag{18}$$

$$+ \boldsymbol{K}_{xM}\boldsymbol{K}_{MM}^{-1}\boldsymbol{A}\boldsymbol{K}_{MM}^{-1}\boldsymbol{K}_{Mx'}, \tag{19}$$

where $\boldsymbol{K}_{xM}$ and $\boldsymbol{K}_{Mx'}$ are cross-covariance vectors. The optimal variational parameters $\boldsymbol{\mu}$ and $\boldsymbol{A}$ are obtained by maximizing the Evidence Lower Bound (ELBO) [22]:

$$\mathcal{F}_{\text{ELBO}} = \int p(\boldsymbol{f}|\boldsymbol{f}_m)q(\boldsymbol{f}_m)\, \log \frac{p(\boldsymbol{y}|\boldsymbol{f})\, p(\boldsymbol{f}_m)}{q(\boldsymbol{f}_m)}\, d\boldsymbol{f}\, d\boldsymbol{f}_m. \tag{20}$$

This variational objective balances model fit and complexity. Closed-form expressions for the optimal parameters are given by:

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{K}_{MM}(\boldsymbol{K}_{MM} + \sigma^{-2}\boldsymbol{K}_{MN}\boldsymbol{K}_{NM})^{-1}\boldsymbol{K}_{MN}\boldsymbol{y}, \tag{21}$$

$$\boldsymbol{A} = \boldsymbol{K}_{MM}(\boldsymbol{K}_{MM} + \sigma^{-2}\boldsymbol{K}_{MN}\boldsymbol{K}_{NM})^{-1}\boldsymbol{K}_{MM}. \tag{22}$$

B. Training and Hyperparameter Optimization

The predictive power of Gaussian Processes (GPs) critically depends on the appropriate selection of their hyperparameters, which govern the behavior of the kernel function and noise model. Training a GP entails the optimization of these hyperparameters by maximizing the marginal likelihood (or its variational lower bound in the case of SVGPs approximations). This step ensures that the resulting Posterior distribution is statistically consistent with the observed data and capable of generalizing to unseen inputs [21].

In exact or Full GPs, hyperparameter learning is achieved by minimizing the Negative Log Marginal Likelihood (NLML), a convex objective under certain kernels. For a training dataset of inputs $\boldsymbol{X} = \{\boldsymbol{x}_n\}_{n=1}^N$ and targets $\boldsymbol{y} = \{y_n\}_{n=1}^N$, the marginal likelihood under a GP Prior with hyperparameters θ is given by:

$$P(\boldsymbol{y}|\boldsymbol{X}, \theta) = \frac{1}{\sqrt{(2\pi)^N \det(\boldsymbol{K}_{NN} + \boldsymbol{\Sigma})}} \tag{23}$$

$$\times \exp\left(-\frac{1}{2}\boldsymbol{y}^T(\boldsymbol{K}_{NN} + \boldsymbol{\Sigma})^{-1}\boldsymbol{y}\right), \tag{24}$$

where $\boldsymbol{K}_{NN}$ is the kernel matrix, and $\boldsymbol{\Sigma} = \sigma^2\boldsymbol{I}$ denotes the noise covariance. Taking the negative logarithm yields the NLML:

$$\mathcal{L}_{\text{NLML}} = \frac{1}{2}\boldsymbol{y}^T(\boldsymbol{K}_{NN} + \boldsymbol{\Sigma})^{-1}\boldsymbol{y} + \frac{1}{2}\log\det(\boldsymbol{K}_{NN} + \boldsymbol{\Sigma}) + \frac{N}{2}\log(2\pi). \tag{25}$$

This formulation assumes a zero mean function; in practice, a mean vector $\boldsymbol{m}$ can be incorporated, modifying the first term to $\frac{1}{2}(\boldsymbol{y}-\boldsymbol{m})^T(\boldsymbol{K}_{NN}+\boldsymbol{\Sigma})^{-1}(\boldsymbol{y}-\boldsymbol{m})$. Minimizing this expression yields hyperparameters that balance data fit and model complexity, with the log-determinant term acting as an Occam's razor penalty [5, 21].

For large-scale problems, the Sparse Variational Gaussian Process (SVGP) provides a scalable alternative by introducing a variational distribution over a set of M inducing points. Hyperparameter optimization is then performed by maximizing the Evidence Lower Bound (ELBO) on the marginal likelihood [22]. The ELBO trades off data fidelity and complexity via a Kullback-Leibler divergence term, allowing efficient minibatch-based optimization:

$$\mathcal{F}_{\text{ELBO}} = \int q(\boldsymbol{f}_m)\, p(\boldsymbol{f}|\boldsymbol{f}_m)\, \log\frac{p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}_m)}{q(\boldsymbol{f}_m)}\, d\boldsymbol{f}\, d\boldsymbol{f}_m. \tag{26}$$

In SVGPs, the locations of the inducing points $\boldsymbol{X}_m$ are often treated as additional variational parameters. Initial values may be sampled randomly from the training data or selected via clustering algorithms such as KMeans [11, 12]. While the positions can be refined during training, the number of inducing points must be chosen carefully: too few degrade the approximation, while too many undermine the computational benefits of sparsity [11, 12].

The optimization of the ELBO or NLML is typically performed using gradient-based methods. Hyperparameters are updated iteratively via stochastic gradient descent or variants such as Adam, following the general update rule:

$$\theta^{(t+1)} = \theta^{(t)} \pm \eta\, \nabla_\theta J(\theta^{(t)}), \tag{27}$$

where $J(\theta)$ is the chosen objective and η the learning rate. Careful tuning of η is essential: overly large values can destabilize convergence, while small values lead to prohibitively slow learning.

The computational complexity per training iteration scales as $\mathcal{O}(N^3)$ for Full GPs and $\mathcal{O}(NM^2 + M^3)$ for SVGPs [11, 12]. The latter offers a substantial reduction for $M \ll N$, making it viable for applications involving tens of thousands of datapoints. However, total training time also depends on the number of hyperparameters and convergence properties of the optimizer.

C. Kernels

The kernel, or covariance function, lies at the core of Gaussian Process (GP) modeling, as it encodes the prior assumptions about the smoothness, stationarity, and general structure of the underlying function. The choice of kernel directly determines the expressiveness and inductive bias of the GP and therefore has a profound impact on both predictive performance and uncertainty quantification [5, 21].

In this work, we focus on two widely used stationary kernels: the Radial Basis Function (RBF) kernel and the Matern kernel. Both belong to the class of translation-invariant covariance functions, but differ in their assumptions about function smoothness [24].

The RBF kernel, also known as the squared exponential or Gaussian kernel, assumes infinitely differentiable functions and is defined as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\ell^2}\right), \tag{28}$$

where σ^2 is the signal variance controlling the amplitude of the function, and ℓ is the characteristic length scale that determines how quickly correlations decay with distance. The RBF kernel produces very smooth functions and is often the default choice when no prior knowledge is available, due to its strong generalization capacity under mild assumptions [24].

In contrast, the Matern kernel introduces an additional hyperparameter ν that governs the differentiability of the resulting functions [24]. It is given by:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}||\boldsymbol{x} - \boldsymbol{x}'||}{\ell} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}||\boldsymbol{x} - \boldsymbol{x}'||}{\ell} \right), \tag{29}$$

where K_ν is the modified Bessel function of the second kind, $\Gamma(\cdot)$ is the gamma function, and $\nu > 0$ is the smoothness parameter. For specific half-integer values of ν, such as $\nu = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}$, the kernel simplifies to expressions involving elementary functions, which are computationally advantageous [24].

As $\nu \to \infty$, the Matern kernel converges to the RBF kernel, thereby encompassing it as a special case. Small values of ν yield rougher functions with fewer derivatives, making the Matern kernel particularly suitable for modeling physical processes exhibiting non-smooth or noisy behavior. For example, $\nu = \frac{1}{2}$ corresponds to the exponential kernel (absolute exponential), while $\nu = \frac{5}{2}$ yields twice-differentiable functions [24].

In practice, the choice between RBF and Matern kernels should be guided by the nature of the target function. When high smoothness is expected or preferred, the RBF is often appropriate. In cases involving irregular or structured noise, Matern kernels offer greater flexibility by tuning ν to match the observed functional variability. This adaptability is particularly valuable in scientific applications where domain knowledge informs smoothness priors.

III. BENCHMARK PROBLEMS AND DATASET CHARACTERIZATION

This section presents the two benchmark datasets used to evaluate Gaussian Process (GP) models under distinct modeling challenges: a low-dimensional, physically interpretable dataset derived from a molecular system, and a high-dimensional, heterogeneous dataset stemming from catalysis. For both problems, all features and function values were standardized via z-score normalization to ensure numerical stability and facilitate model convergence during training.

A. Distorted Water Molecule: Low-Dimensional Surrogate Modeling

The first dataset is based on a simplified quantum-mechanical model of a distorted water molecule, originally proposed by Deringer et al. [2]. As illustrated in Figure 2, the system is described using two internal degrees of freedom: the bending angle ω and the asymmetric O–H stretch $\nu' = d_{\mathrm{OH}_1} - d_{\mathrm{OH}_2}$, with the constraint that the average bond length remains fixed at $\frac{1}{2}(d_{\mathrm{OH}_1} + d_{\mathrm{OH}_2}) = 0.95$ Å. For each combination of these

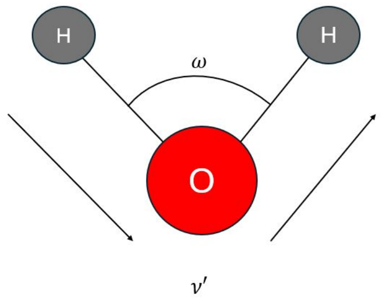


FIG. 2. Schematic representation of the distorted water molecule model showing the bending angle ω and asymmetric stretch ν' used as features in the water molecule dataset.

coordinates, the Partridge–Schwenke potential was used to compute reference values for the molecular energy and dipole moments along the x and y axes.

The resulting dataset contains 121 data points and serves as a controlled environment to assess the predictive performance and calibration quality of Full GP models. Two kernel classes were considered: the Radial Basis Function (RBF) and the Matern family with smoothness parameters $\nu \in \{1/2, 3/2, 5/2\}$. All kernels were composed with a ScaleKernel wrapper to independently learn signal variance and were paired with a constant mean function. Detailed formulations for these models are provided in the appendix.

Each model was trained on 80% of the data and validated on the remaining 20%. Performance was assessed via the mean squared error (MSE) and the coefficient of determination (R^2). Additionally, predicted values were plotted against ground truth values to visualize the calibration of posterior means and variances. Finally, a sensitivity analysis was conducted by evaluating R^2 scores as a function of training set size, providing insights into data efficiency.

B. Oxidative Coupling of Methane: High-Dimensional Modeling with Sparse GPs

The second benchmark task leverages a high-dimensional dataset describing 4715 oxidative coupling of methane (OCM) reactions, obtained from Mine et al. [16]. Each reaction is associated with 30 primary features, encompassing both continuous variables (e.g., molar concentrations, temperature, pressure, contact time) and categorical descriptors (e.g., catalyst composition, support material, synthesis method). One-hot encoding of the categorical variables resulted in a final input dimensionality of 289.

The target variable is the reaction Yield. Given the scale and complexity of the dataset, we employed a Sparse Variational Gaussian Process (SVGP) model, using the same kernel and mean function combination that yielded the best performance on the water molecule task. The SVGP was enhanced with Automatic Relevance Determination (ARD), enabling feature-wise length scale tuning. This regularizes the model by suppressing uninformative dimensions during training.

To evaluate SVGP initialization strategies, we tested both random and KMeans-based placement of inducing points. Using 50% of the data for training, we studied the impact of inducing points count and initialization method on both predictive accuracy (measured via R^2) and runtime. The findings informed the selection of an initialization scheme and an appropriate number of inducing points.

We validated these selections with a second experiment using 95% of the dataset for training. This larger experiment confirmed the robustness of the earlier observations and supported the scalability of SVGP to larger datasets. With the SVGP fully characterized, we conducted a direct comparison with a Full GP, assessing training and prediction combined runtimes across dataset sizes. The computational cost of the process is dominated by training, with the predictive contribution considered negligible. As expected, the SVGP significantly reduced computational costs, scaling as $\mathcal{O}(NM^2+M^3)$ per epoch versus $\mathcal{O}(N^3)$ for the Full GP.

Lastly, we benchmarked GP models against popular ensemble methods, including K-Nearest Neighbors (KNN), Gradient Boosted Regression (GBR), Extreme Gradient Boosting (XGBoost), and Random Forest Regression (RFR). These models provided additional context on the predictive strengths and limitations of GP methods in the high-dimensional regime.

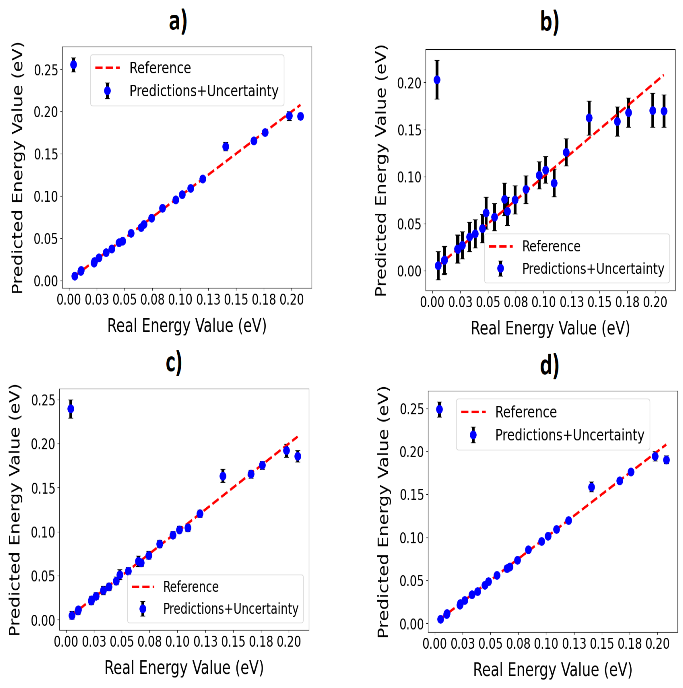


FIG. 3. Energy predictions for the four studied models versus their real values. Panel a) shows the results of a model with an RBF kernel, whereas panels b), c), and d) correspond to the results for models with a Matern Kernel with smoothness factor $\nu = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}$ respectively. The error bars denote the uncertainty associated with the predicted value by the Gaussian Process, calculated from the variance.

IV. RESULTS AND MODEL EVALUATION

A. Low-Dimensional Regression: Water Molecule Case Study

To benchmark the performance of different Full Gaussian Process (GP) configurations, we begin with a simplified yet representative regression task: predicting the energy of a distorted water molecule using two input coordinates. This low-dimensional system offers an ideal testbed to assess kernel behavior and model generalization under data-limited conditions. Each model was trained on 80% of the data over 600 epochs with a fixed learning rate of $\eta = 0.1$, which ensured stable convergence across all kernel choices. Preliminary tests showed that small variations in η had negligible impact on the final Negative Log Marginal Likelihood (NLML) value, allowing us to focus analysis on kernel-specific effects.

Figures 3(a)–(d) show the predicted versus actual energy values for four Gaussian Process (GP) models, revealing detailed differences in their performance and calibration. The Radial Basis Function (RBF) kernel and the Matern kernel with $\nu = 5/2$ provide the most accurate and tightly clustered predictions along the identity line, indicating excellent predictive accuracy and model calibration. The narrow uncertainty intervals associated with these models underscore their reliability and robustness, reinforcing their suitability for modeling the complex relationships within the studied system.

In contrast, the Matern kernel with $\nu = 1/2$ displays predictions that deviate significantly from the identity line, especially at higher energy values, and presents substantially broader uncertainty bands. This behavior suggests challenges in generalization, which align well with the higher Negative Log Marginal Likelihood (NLML) values seen during training (see Figure 5(a)). Such deviations and uncertainties illustrate that this kernel, characterized by its sharpness and limited smoothness, is less appropriate for capturing the smooth energy surface of the system under study.

Meanwhile, the Matern kernel with $\nu = 3/2$ shows intermediate behavior, achieving predictions closer to the identity line compared to the $\nu = 1/2$ case, yet with broader uncertainty intervals than those observed in the RBF and $\nu = 5/2$ models. This highlights the nuanced trade-off between kernel smoothness and predictive performance, emphasizing the importance of choosing an appropriate kernel smoothness factor. Overall, these results reinforce the critical role of kernel choice in Gaussian Process modeling, emphasizing that

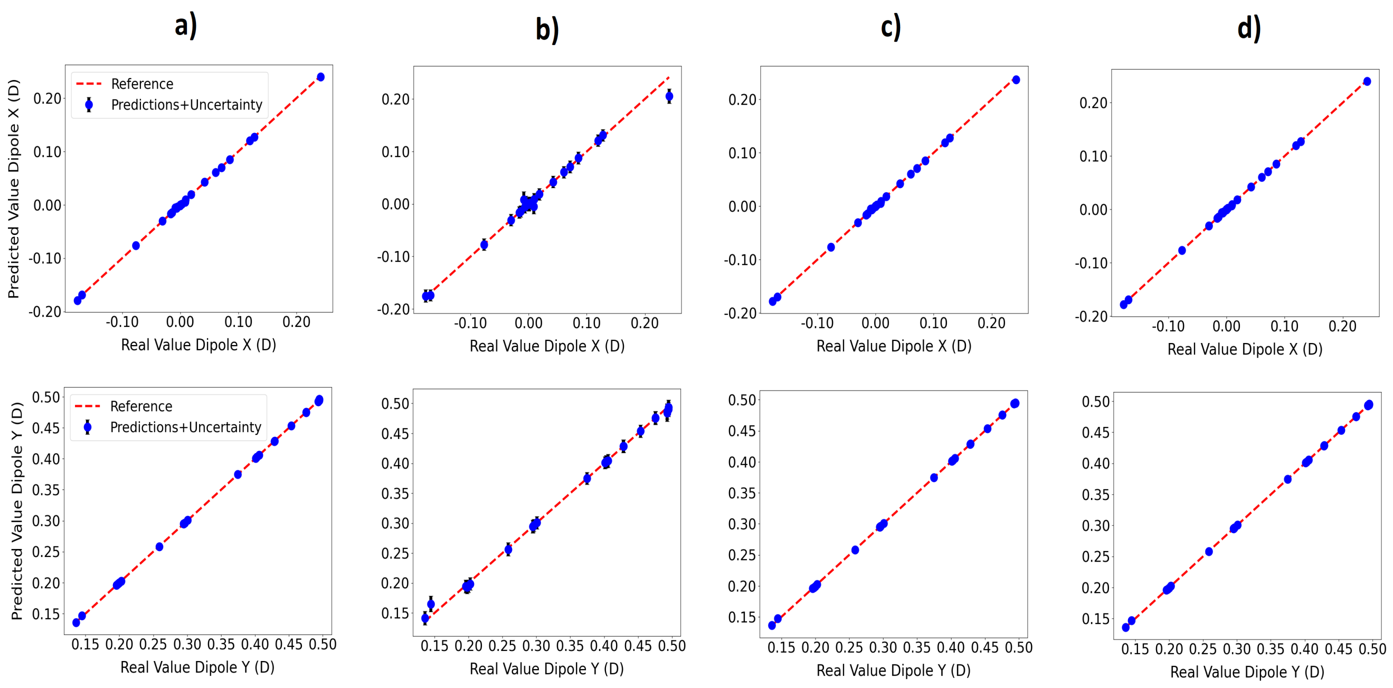


FIG. 4. Predicted vs actual dipole moments for the x (top row) and y (bottom row) directions. Columns correspond to different kernels: (a) RBF, (b) Matern $\nu = 1/2$, (c) Matern $\nu = 3/2$, (d) Matern $\nu = 5/2$. Error bars reflect model uncertainty.

smoother kernels (such as RBF and Matern ν= 5/2) are more effective in accurately capturing and predicting energy-related properties in this model. In contrast, less smooth kernels can introduce significant predictive uncertainty and reduced generalization capability, especially at higher energy values.

Quantitative performance metrics for each model are presented in Table I. The inclusion and exclusion of an outlier, data point located in the upper left corner of Figures 3(a)–(d), provides insight into robustness. With the outlier included, the Matern $\nu = 1/2$ model surprisingly shows the highest R^2, but this is misleading — the outlier aligns more closely with that model's prediction, artificially inflating the metric. Once removed, the Matern $\nu = 5/2$ and RBF models clearly outperform the others in both MSE and R^2, corroborating visual observations and training loss trends.

We further evaluated model performance on predicting the molecular dipole moment components along the x and y axes. These continuous physical properties provide a complementary test of model smoothness and generalization. Figure 4 shows predicted versus actual dipole moment values across a range of kernels for both x (top row) and y (bottom row) components. Overall, all kernel models demonstrate uniformly high predictive accuracy, with points closely aligned along the identity line. This underscores the relative simplicity and consistency of dipole moment predictions across kernel choices.

Upon detailed inspection, however, subtle differences emerge. The Radial Basis Function (RBF) and Matern kernels with higher ν values (ν= 3/2 and ν= 5/2) exhibit exceptionally tight clustering of predictions and narrow uncertainty intervals, reflecting strong model calibration and reliability. In particular, the Matern kernel with ν= 5/2 closely matches the accuracy of the RBF kernel, suggesting that these kernels capture the underlying physical relationships in dipole moment predictions exceptionally well.

In contrast, the Matern kernel with ν= 1/2 again shows slightly broader uncertainty intervals and a more pronounced residual spread, especially visible in the x-dipole predictions. These characteristics align with earlier observations of slower convergence and higher uncertainty for this kernel in energy prediction, further reinforcing the conclusion that kernels with lower

Kernel	**MSE** (eV^2)	R^2	**MSE** (eV^2) **without the point**	R^2 **without the point**
RBF	0.0025	0.30	$2.2 \cdot 10^{-5}$	0.993
Matern, $\nu = 1/2$	0.0017	0.52	$14.5 \cdot 10^{-5}$	0.959
Matern, $\nu = 3/2$	0.0022	0.37	$4.5 \cdot 10^{-5}$	0.987
Matern, $\nu = 5/2$	0.0024	0.33	$2.7 \cdot 10^{-5}$	0.992

TABLE I. MSE and R^2 values for energy predictions, with and without the identified outlier, using a training dataset size of 80% of the total. All models trained over 600 epochs with $\eta = 0.1$.

smoothness (low ν values) may be less effective in modeling smoothly varying physical properties.

Interestingly, predictions for the y-dipole component are slightly more accurate across all kernels, displaying tighter clustering around the reference line than the corresponding x-dipole predictions. This minor discrepancy may be related to intrinsic directional differences in the dipole moment behavior or data distribution within the training dataset. In summary, while all kernels effectively capture dipole moment behaviors, the RBF and higher-order Matern kernels demonstrate superior accuracy, narrower uncertainty intervals, and better predictive stability, suggesting their greater suitability for modeling dipole moments in similar molecular systems.

The corresponding metrics are listed in Table II. All models achieve near-perfect R^2 values ($\sim$ 0.99) for both components, but again, the Matern $\nu = 5/2$ and RBF models consistently yield the lowest MSEs. The x-component shows greater kernel sensitivity, while the y-component is uniformly well-predicted. This suggests some anisotropy in the dipole response surface, better captured by smoother kernels.

Training curves shown in Figure 5 further highlight differences in optimization dynamics among kernel choices. The Matern kernel with $\nu = 1/2$ consistently demonstrates slow convergence and stabilizes at significantly higher Negative Log Marginal Likelihood (NLML) values, which aligns with its comparatively weaker performance observed in predictive tasks. This slower convergence and higher final NLML underscore its limited suitability for modeling the complexities of the energy and dipole moment predictions presented in this work.

Conversely, the Matern kernels with higher ν values (ν= 3/2 and $\nu = 5/2$) and the Radial Basis Function (RBF) kernel exhibit rapid and robust convergence behavior. Specifically, the Matern $\nu = 5/2$ and RBF kernels converge quickly and reach very low NLML values across energy and both dipole moment components, suggesting their ability to efficiently capture the relevant physical interactions and provide well-calibrated uncertainty estimates. Notably, the RBF kernel shows slightly lower final NLML values, particularly in dipole moment predictions (Figures 5(b) and 5(c)), reinforcing the kernel's overall robustness and suitability.

Kernel	**Component**	**MSE (D^2)**	R^2
RBF	x	$1.3 \cdot 10^{-6}$	0.9998
Matern, $\nu = 1/2$	x	$7.3 \cdot 10^{-5}$	0.9892
Matern, $\nu = 3/2$	x	$1.7 \cdot 10^{-6}$	0.9997
Matern, $\nu = 5/2$	x	$5.3 \cdot 10^{-7}$	0.9999
RBF	y	$4.2 \cdot 10^{-7}$	0.99997
Matern, $\nu = 1/2$	y	$2.6 \cdot 10^{-5}$	0.99820
Matern, $\nu = 3/2$	y	$4.9 \cdot 10^{-7}$	0.99996
Matern, $\nu = 5/2$	y	$3.0 \cdot 10^{-7}$	0.99997

TABLE II. MSE and R^2 values for dipole moment predictions along the x and y axes, using a training dataset size of 80% of the total. All models trained over 600 epochs with $\eta = 0.1$.

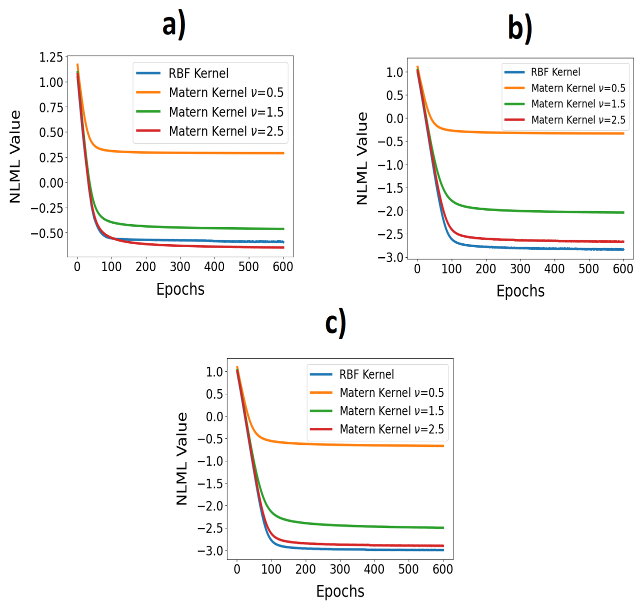


FIG. 5. NLML convergence curves for models predicting (a) energy, (b) x- and (c) y-dipole moments. Lower values indicate better fit and model calibration. Training was performed using 80% of the total data.

The distinction between kernels becomes clearer when comparing energy predictions (Figure 5(a)) and dipole moment predictions (Figures 5(b) and 5(c)). Dipole moments, representing simpler or more linear interactions, enable convergence to lower NLML values for most kernel choices, whereas energy predictions involve greater complexity and variability, evident from their higher NLML values. This difference emphasizes the importance of selecting appropriate kernel functions depending on the predictive task, with higher-order Matern and RBF kernels demonstrating a versatile and effective choice for both energy and dipole moment modeling tasks.

In summary, the observed differences in NLML convergence across kernel types underline the crucial interplay between kernel smoothness, complexity, and optimization efficiency. These insights offer practical guidance for future implementations of Gaussian process regression models in computational materials science and molecular simulations.

Finally, Figure 6 presents the R^2 curves as a function of training dataset size, offering insight into data efficiency.

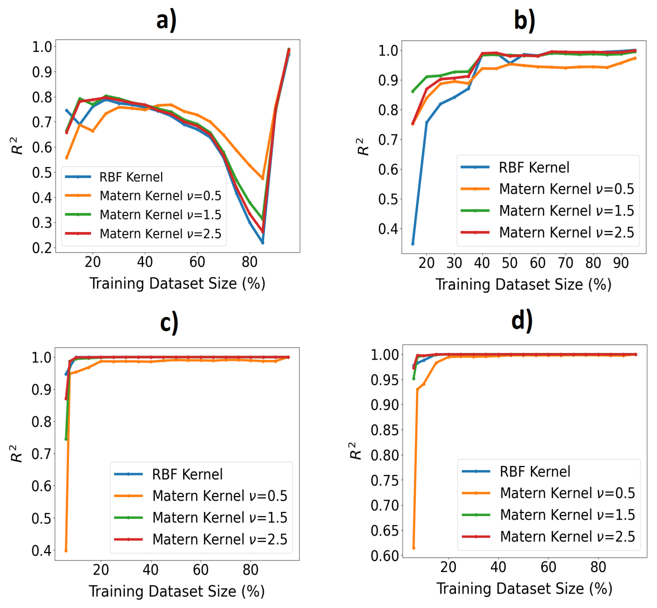


FIG. 6. Evolution of R^2 versus training dataset size. (a) Energy with outlier included; (b) Energy without outlier; (c) Dipole moment x; (d) Dipole moment y. All models trained over 600 epochs with $\eta = 0.1$.

Notably, dipole predictions remain highly accurate ($R^2 > 0.9$) with as few as 9 training points, highlighting the strength of Gaussian processes (GPs) in data-scarce regimes. In particular, Figures 6(c) and 6(d) illustrate that all kernel choices—Radial Basis Function (RBF) and Matern kernels with ν values of 1/2, 3/2, and 5/2—quickly reach near-perfect predictive performance even at very small training set sizes, demonstrating remarkable data efficiency and robustness across all kernel types for dipole predictions.

In contrast, energy predictions depicted in Figures 6(a) and (b) exhibit significant variability and a stronger dependence on kernel choice. Especially at smaller dataset sizes, the Matern kernel with ν=5/2 consistently outperforms the RBF kernel, underscoring its superior inductive bias suited for the particular physical system under consideration. Additionally, the sudden improvement of all kernel performances at larger dataset sizes (80–90%) in Figure 6(a) reflects specific dataset characteristics related to the presence of the outlier point in the training dataset instead of the testing dataset.

Interestingly, the difference between kernels is more pronounced at low data counts for energy prediction, but diminishes as the dataset size increases, approaching similar high levels of accuracy, except for the Matern kernel with ν=1/2 in Figure 6(b), for training sizes above 80%. Conversely, the dipole moment predictions demonstrate consistently high accuracy and limited kernel dependency, suggesting that dipole-related features exhibit simpler or more linear dependencies compared to energy-related properties in this context. The results emphasize the nuanced interplay between kernel choice and dataset size, guiding future efforts in optimizing Gaussian process regression models for specific predictive tasks in materials and molecular simulations.

In summary, this case study demonstrates the power of Full GPs for low-dimensional regression in materials modeling. The Matern $\nu = 5/2$ kernel consistently emerges as the top performer, balancing accuracy, convergence speed, and uncertainty calibration. These results validate the use of GPs for small-data, high-precision applications common in quantum chemistry and materials science.

B. OCM Reactions Database

Having analyzed the results obtained for the water molecule, we now extend our investigation to a second dataset. For this system, both a Sparse Variational Gaussian Process (SVGP) and a Full Gaussian Process (Full GP) were employed, each constructed with the same mean function and kernel that yielded the best results for water—namely, the Matern kernel with $\nu = \frac{5}{2}$. The decision to apply both models is justified by their complementary strengths: while the Full GP offers high fidelity in modeling due to its complete treatment of the kernel matrix, its training scales poorly with dataset size ($\mathcal{O}(N^3)$ time complexity), making it impractical for larger datasets. SVGP, by contrast, approximates the posterior using a representation based on inducing points, providing a tractable alternative that scales more favorably ($\mathcal{O}(NM^2 + M^3)$, with $M \ll N$) while still capturing the essential features of the function.

To ensure comparability between the models and promote stable convergence, both were trained for 900 epochs using a learning rate of $\eta = 0.01$. This setup proved effective in minimizing the negative log marginal likelihood (NLML) for the Full GP and maximizing the evidence lower bound (ELBO) for the SVGP. In the case of Full GPs, convergence was achieved for any dataset size; however, we can distinguish two distinct behaviors, as seen in Figures 7 (a) and (b). For training dataset sizes smaller than approximately 70%, the NLML value converged smoothly to its minimum. For larger sizes, we can observe that the NLML reaches a minimum and then increases again, converging to a slightly higher value. The differences between these two values are very small and were not significant for the predictions made with models trained either up to the minimum or to convergence. For this reason, it was decided to train the model until convergence, a common practice that allows for a fair comparison with SVGP in terms of execution time. Additionally, this jump could only be considered noticeable for a size of 95%. In the case of SVGP, spikes in the ELBO were observed during training, such as those in Figure 7 (c). These spikes occurred randomly and were not affected by changes in the learning rate or the number of training epochs. However, these spikes do not prevent

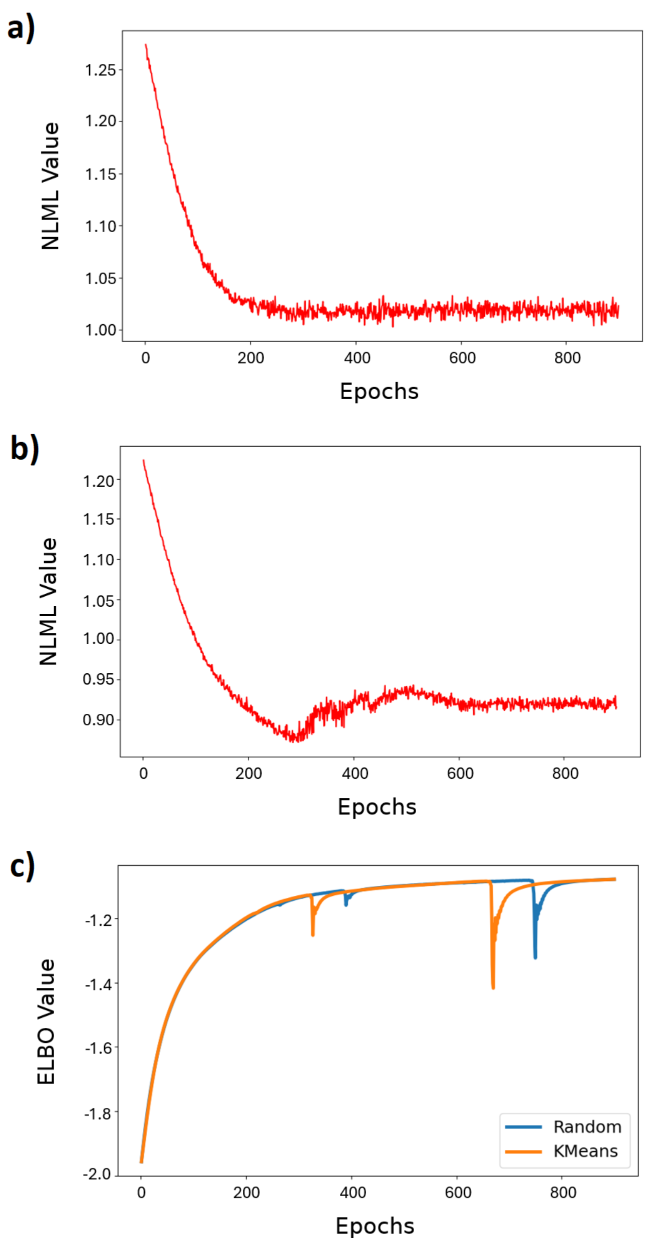


FIG. 7. Evolution of the NLML value during training for training dataset sizes of 50% (a) and 95% (b) of the total dataset. Panel (c) shows the evolution of the ELBO value during SVGP training with random initialization and KMeans, using 11% inducing points on a training dataset representing 50% of the total data. The curves were obtained over 900 training epochs with a learning rate of $\eta = 0.01$.

the attainment of coherent results in line with theoretical expectations, as we will see.

As mentioned in the previous sections, for this problem a comparative study was first carried out between different initialization methods for the position of the inducing points. In our case, the methods were random selection of points from the training dataset and initialization using the KMeans algorithm. The results obtained are shown in Figure 8.

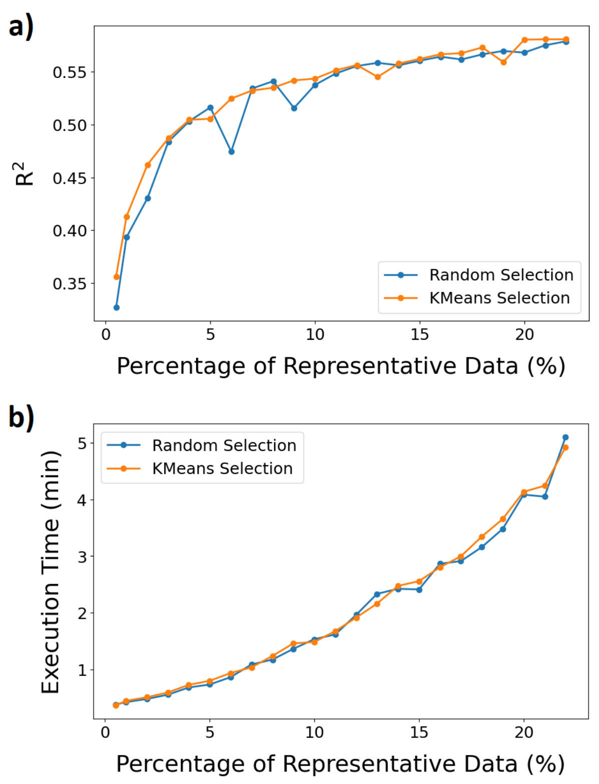
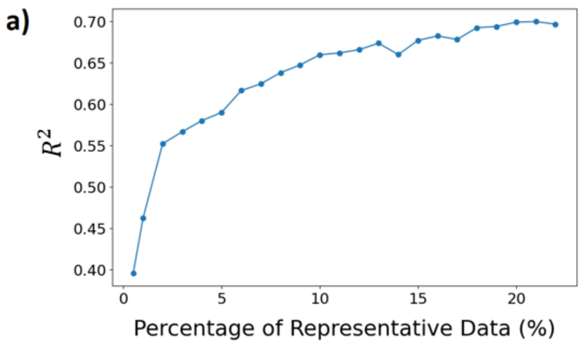


FIG. 8. Evolution of the R^2 coefficient a) and of the combined execution time of training and predictions b) with the number of representative (inducing) points used for the two initialization methods of their positions. The number of representative points ranges from 0.5% to 22% of the total number of data points in the training dataset. The training dataset used constituted 50% of the total dataset.

Starting with the evolution of the R^2 coefficient, we can see that both methods yield practically identical results. The only notable differences occur at the lowest values of the number of inducing points and some peaks where the R^2 value drops suddenly. For small numbers of inducing points, we are not able to adequately cover all relevant regions of the input space. In these cases, the initial choice of inducing points can influence the quality of the prediction. Random selection may cause the initial positions to be clustered around a specific region. During optimization, although the positions are adjusted to converge to the maximum of the ELBO, the maximum reached may be a local one rather than a global one. On the other hand, initialization using the KMeans algorithm can yield better results by offering a better representation of the data structure. This may allow the SVGP, during ELBO optimization, to converge to a higher local maximum or to the global maximum, improving the results of random initialization in this range of values.

Once a sufficient number of points is reached to adequately cover the input space, we can observe how



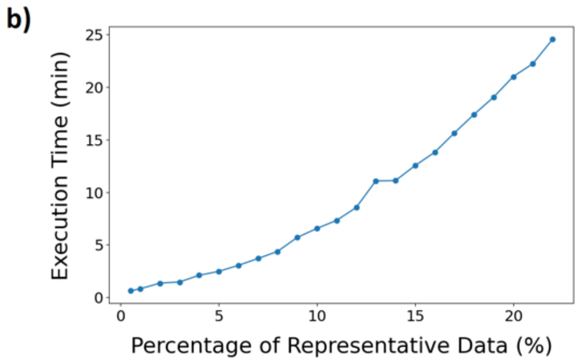


FIG. 9. Evolution of the R^2 coefficient a) and of the combined execution time of training and predictions b) with the number of representative (inducing) points used. The number of representative points ranges from 0.5% to 22% of the total number of data points in the training dataset. The training dataset used constituted 95% of the total dataset.

both methods begin to exhibit nearly identical R^2 values, as one would expect. The main differences occur in the peaks observed in the evolution of the R^2 coefficient, especially with random initialization. These are due to the aforementioned random peaks observed during training, like the ones in Figure 7 (c), which could not be controlled. In these peaks, the ELBO value drops suddenly, resulting in a loss of the SVGP's predictive capability. Finally, we can also observe that for the largest numbers of inducing points, the R^2 associated with KMeans initialization slightly outperforms random initialization, but not significantly.

Focusing now on the total execution time of training and the predictive process, we can see that there are no notable differences. The times obtained in all cases are practically identical for both initializations and increase with the number of inducing points considered. This is expected as the computational cost per training epoch, which dominates the total cost, and prediction process are independent of how the inducing points are initialize. Additionally, the number of parameters to be optimized is the same in both SVGPs, so there cannot be a time increase due to this either.

Taking these results into account, the KMeans algorithm was selected as the initialization method.

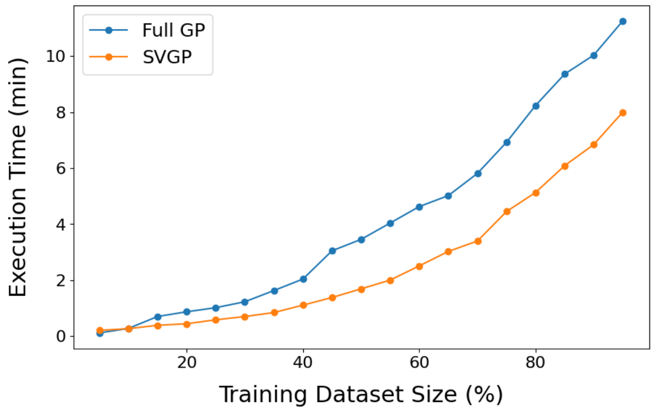


FIG. 10. Evolution of the combined execution time of training and prediction process of the Full GP and the SVGP as a function of training dataset size. The training dataset size is given as a percentage of the total dataset size. The number of inducing points used to construct the SVGP equals a 11% of each training dataset size.

Although the results obtained are practically identical to those from random initialization, the results for few inducing points clearly show that the KMeans algorithm initializes the position of the inducing points more appropriately, improving convergence toward the global maximum of the ELBO.

Once the method for initializing the inducing points was selected, the next step was to choose the number of these points to use for the construction of the SVGP. Although the evolution of both the R^2 coefficient and the execution time can already be inferred from Figure 8, and with this a coherent number of inducing points to use, a new study was conducted on these same variables using a training dataset composed of 95% of the full dataset to have more data in order to decide. This study is shown in Figure 9, and as we can see, in both cases the behavior obtained is practically identical to that for the 50% case.

The R^2 coefficient initially experiences a pronounced increase in its value. This is because we are initially considering very few inducing points. As a result, we are not able to adequately cover all the relevant areas of the space, leading to very low prediction accuracy. Thus, a small addition of new inducing points helps to mitigate this, producing a rapid increase in the R^2 value. After this initial rapid increase, once around 9–10% of inducing points is reached, the growth begins to slow significantly. In fact, the increase in R^2 obtained by increasing the number of inducing points up to 22% is around 0.05 in both cases. This is because the SVGP can only provide an approximation of the exact Posterior distribution of a Full GP. Consequently, once a certain number of inducing points is reached, the improvement in predictive capability becomes practically negligible. Thus, toward the end of the graph, when considering the largest sets of inducing points, the improvements are almost imperceptible for both the 50% and 95% cases.

In addition, the largest quantities of inducing points considered begin to violate the theoretical condition of Sparse GPs, namely that $M \ll N$. For example, 22% of inducing points corresponds to 985 points when the total training data is 4479 (95% of the total). This is a high number of inducing points, which may no longer fall within the previously mentioned condition. Furthermore, it should be noted that increasing the number of inducing points not only increases the theoretical computational cost of training per epoch, thereby increasing execution time, but also increases the number of hyperparameters to optimize, such as the positions of these inducing points. This is reflected in Figure 9 (b), which exhibits the same behavior as observed in Figure 8 (b). In both cases, the training and predictive process execution time increases with the number of inducing points. This increase is much more significant in Figure 9 (b), where the final execution time is slightly below 25 minutes.

Taking into account these results and the fact that when using the SVGP we are not aiming to obtain results similar to a Full GP, but rather to apply it to large datasets, a percentage of 11% of inducing points was selected for the comparative study with the Full GP. This is because, for this percentage, we can see that for both 95% and 50% training dataset sizes, the growth in R^2 begins to stagnate, with no further significant increases in its value, although execution time continues to increase.

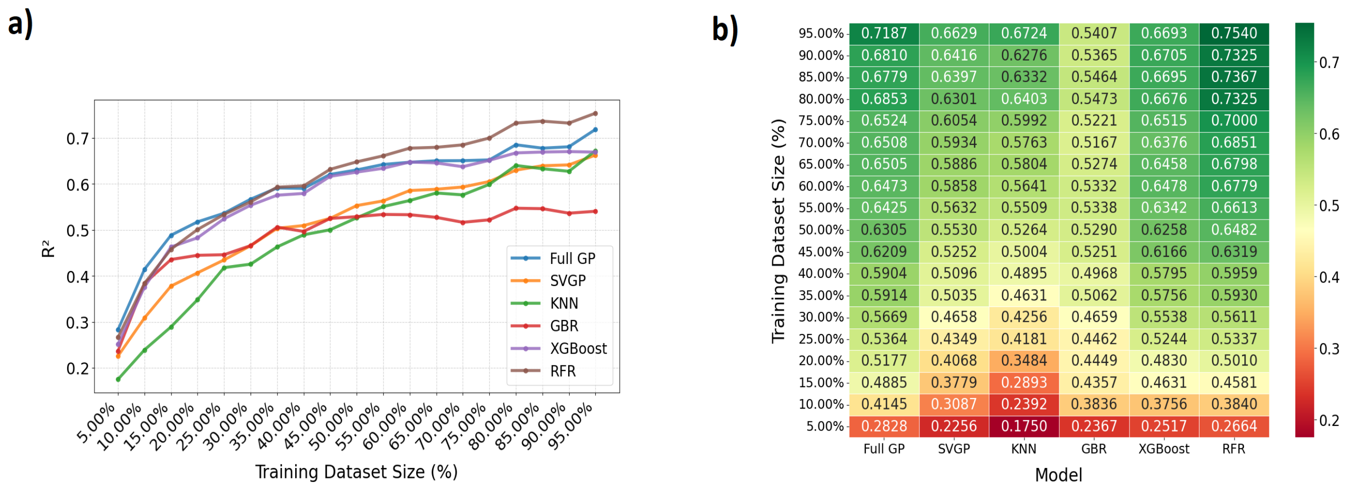
With this value, the comparative study between the SVGP and the Full GP began, starting with a comparison of the combined execution time of training and prediction processes for both models with respect to the size of the training dataset. The results obtained are shown in Figure 10. As we can observe, for both Gaussian Processes there is an increase in execution time with the number of training data points, N. This is expected, since the theoretical computational cost of training increases with N for each training epoch. However, the growth in computational cost with N is significantly higher for the Full GP than for the SVGP, which results in a much longer execution time for the Full GP. This is achieved despite the larger number of hyperparameters that need to be optimized in the SVGP. This time difference becomes increasingly significant as larger datasets are considered, as one would expect.

For small datasets, the execution time is nearly identical. This is because the computational costs of both models, although higher for the Full GP which scales with N^3, are still quite similar. Therefore, the difference in execution time can become negligible when taking into account the greater number of hyperparameters to be optimized in the SVGP. However, when considering the largest training dataset size, the execution time difference between the two is approximately 3 minutes, a considerable reduction. This would not have been possible if a larger number of inducing points had been selected in an attempt to match the predictive performance of a Full GP. Recall that for a training dataset size of 95%, using 22% inducing points resulted in a total time of a little under 25 minutes, much longer than that required by the Full GP. This is because the increase in training time due to the number of hyperparameters to optimize is not negligible. Thus, with 11% of inducing points, we are considering about 492 points, along with the corresponding optimization of their positions, whereas with 22% we are working with 985, representing a significantly greater number of hyperparameters. Taking all this into account, SVGPs represent a strong option for working with Gaussian Processes on large datasets, significantly reducing execution time at the cost of some loss in prediction accuracy.

The reduction in predictive accuracy becomes particularly evident when analyzing the evolution of the R^2 coefficient as a function of training dataset size, as illustrated in Figure 11. This figure provides a comprehensive comparison between the Full Gaussian Process (Full GP), the Sparse Variational Gaussian Process (SVGP), and several other commonly used regression models, including k-Nearest Neighbors (KNN), Gradient Boosting Regressor (GBR), XGBoost, and Random Forest Regressor (RFR).

Panel (a) of Figure 11 presents the R^2 scores obtained by each model across a wide range of training dataset sizes, from 5% to 95%. This line plot reveals several important trends. Notably, the Full GP exhibits consistently strong performance across almost the entire range, often outperforming all other models except the RFR. In particular, for small to moderate training sizes—up to approximately 30%—the Full GP surpasses all other methods, including RFR, achieving R^2 values exceeding 0.5 with as little as 20% of the data. This highlights the remarkable data efficiency and robustness of Full GPs, even in the context of modeling a high-dimensional and complex target function such as the Yield in this study. The RFR ultimately takes the lead at higher data availability, but the Full GP remains highly competitive.

The SVGP, while generally trailing behind the Full GP, still demonstrates a solid predictive capacity. Its curve consistently improves with increasing data, reaching R^2 values around 0.66 with 95% of the data—ranking fith overall, behind the Full GP and RFR and achieving similar results to KNN and XGBoost. This confirms that despite its approximate nature, SVGP maintains a high degree of predictive reliability, especially in large-data regimes. Its somewhat lower performance relative to the Full GP is expected: SVGP approximates the Posterior distribution of the Full GP by relying on a finite set of inducing points. In this work, we intentionally limited the number of inducing points to preserve the computational advantages that make SVGP scalable to large datasets—unlike the computationally expensive Full GP. Thus, while SVGP sacrifices some accuracy, it remains an excellent compromise between predictive power and scalability.



Training Dataset Size (%)	Full GP	SVGP	KNN	GBR	XGBoost	RFR
95.00%	0.7187	0.6629	0.6724	0.5407	0.6693	0.7540
90.00%	0.6810	0.6416	0.6276	0.5365	0.6705	0.7325
85.00%	0.6779	0.6397	0.6332	0.5464	0.6695	0.7367
80.00%	0.6853	0.6301	0.6403	0.5473	0.6676	0.7325
75.00%	0.6524	0.6054	0.5992	0.5221	0.6515	0.7000
70.00%	0.6508	0.5934	0.5763	0.5167	0.6376	0.6851
65.00%	0.6505	0.5886	0.5804	0.5274	0.6458	0.6798
60.00%	0.6473	0.5858	0.5641	0.5332	0.6478	0.6779
55.00%	0.6425	0.5632	0.5509	0.5338	0.6342	0.6613
50.00%	0.6305	0.5530	0.5264	0.5290	0.6258	0.6482
45.00%	0.6209	0.5252	0.5004	0.5251	0.6166	0.6319
40.00%	0.5904	0.5096	0.4895	0.4968	0.5795	0.5959
35.00%	0.5914	0.5035	0.4631	0.5062	0.5756	0.5930
30.00%	0.5669	0.4658	0.4256	0.4659	0.5538	0.5611
25.00%	0.5364	0.4349	0.4181	0.4462	0.5244	0.5337
20.00%	0.5177	0.4068	0.3484	0.4449	0.4830	0.5010
15.00%	0.4885	0.3779	0.2893	0.4357	0.4631	0.4581
10.00%	0.4145	0.3087	0.2392	0.3836	0.3756	0.3840
5.00%	0.2828	0.2256	0.1750	0.2367	0.2517	0.2664

FIG. 11. Evolution of R^2 for the considered models as a function of training dataset sizes. Panel a) shows the evolution graphically, while Panel b) shows the values through a heat map in tabular form. The percentages refer to the portion of the total dataset that makes up the training dataset. For SVGP, a representative data amount equal to 11% of the training dataset size was always used.

Panel (b) of Figure 11 complements this analysis with a heatmap that offers a fine-grained, quantitative view of the R^2 performance across all models and training set sizes. Each cell contains the exact R^2 value corresponding to a given model and data fraction, color-coded to reflect the performance magnitude: deeper green shades indicate higher accuracy, while red corresponds to low predictive power. This panel confirms the trends observed in the line plot: the Full GP and RFR consistently rank among the best-performing models across most training set sizes, with SVGP remaining competitive, especially for larger datasets. Interestingly, while models such as KNN perform relatively poorly at smaller dataset sizes, their performance steadily improves as more data becomes available. However, they do not surpass the Full GP or RFR and have predictive capabilities similar to the SVGP.

Importantly, in contrast to other methods shown, both Full GP and SVGP offer the additional benefit of built-in uncertainty quantification through their probabilistic formulation. This capability allows not only for accurate predictions but also for principled estimation of confidence intervals—an essential feature for downstream applications that require risk-aware decision-making. This, together with their strong performance, further underscores the practical advantages of Gaussian Process models in complex regression settings.

V. CONCLUSIONS

In this work, we have investigated the application of Gaussian Processes (GPs) to two representative problems in materials science, demonstrating their versatility across both low- and high-dimensional regression tasks.

The first part of our study focused on modeling properties of the water molecule, a low-dimensional system with a small dataset. Here, the Full Gaussian Process (Full GP) clearly exhibited excellent predictive accuracy, reinforcing its suitability for problems where data scarcity and low dimensionality are prevalent. Furthermore, this case study highlighted the critical role of kernel selection in the performance of GPs. In particular, the GP with a Matern kernel and smoothness parameter $\nu = \frac{1}{2}$ showed a marked decline in predictive performance compared to other configurations. This was especially evident in the evolution of the R^2 coefficient with increasing training data size. Not only did this kernel underperform in most prediction tasks, but it also exhibited substantially larger predictive uncertainties—an indication of the model's lower confidence and precision.

The second part of the study addressed a more complex and realistic application using the OCM dataset, which involves high-dimensional input features and a large dataset. In this regime, we employed the Sparse Variational Gaussian Process (SVGP), a scalable variant of GP regression. Our results showed that SVGP offers a compelling trade-off between predictive power and computational efficiency. With only 11% of representative (inducing) points, SVGP achieved R^2 values on par with several established machine learning models. While SVGP does not reach the accuracy of the Full GP, its computational cost—particularly in training and inference—scales much more favorably as long as the number of inducing points satisfies the condition $M \ll N$.

Interestingly, our analysis also revealed a saturation effect in SVGP performance: increasing the number of inducing points beyond a certain threshold yields diminishing returns in predictive accuracy while significantly increasing computational cost. In extreme cases, this overhead may offset the efficiency benefits and even exceed the runtime of the Full GP, nullifying the main advantage of sparsity. This observation emphasizes the importance of careful hyperparameter tuning when deploying SVGPs in practice.

Overall, Full GPs proved to be remarkably robust. Even when applied to a high-dimensional problem such as modeling the Yield function from the OCM dataset, the Full GP achieved excellent predictive performance—surpassed only by the Random Forest Regressor (RFR) in some scenarios—despite the increase in computational cost with dataset size.

In summary, Gaussian Processes emerge as a highly effective and adaptable tool for modeling complex functions in materials science. Their ability to deliver high predictive accuracy, quantify uncertainties, and adapt to both small and large datasets—while remaining grounded in sound probabilistic principles—makes them a compelling choice for a wide range of scientific applications. As data-driven approaches continue to advance materials modeling, GPs and their scalable variants like SVGP are likely to play an increasingly important role.

[1] B. Sanchez-Lengeling and A. Aspuru-Guzik, Inverse molecular design using machine learning: Generative models for matter engineering, Science **361**, 360 (2018).

[2] V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti, and G. Csányi, Gaussian process regression for materials and molecules, Chemical Reviews **121**, 10073 (2021).

[3] P. Raccuglia, K. C. Elbert, P. D. Adler, C. Falk, M. B. Wenny, A. Mollo, M. Zeller, S. A. Friedler, J. Schrier, and A. J. Norquist, Machine-learning-assisted materials discovery using failed experiments, Nature **533**, 73 (2016).

[4] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld, Fast and accurate modeling of molecular atomization energies with machine learning, Physical review letters **108**, 058301 (2012).

[5] E. Schulz, M. Speekenbrink, and A. Krause, A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions, Journal of mathematical psychology **85**, 1 (2018).

[6] J. Wang, An intuitive tutorial to gaussian process regression, Computing in Science & Engineering **25**, 4 (2023).

[7] P. I. Frazier, A tutorial on bayesian optimization, arXiv preprint arXiv:1807.02811 (2018).

[8] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, Bayesian optimization for adaptive experimental design: A review, IEEE access **8**, 13937 (2020).

[9] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons, Physical review letters **104**, 136403 (2010).

[10] D. Dragoni, T. D. Daff, G. Csányi, and N. Marzari, Achieving dft accuracy with a machine-learning interatomic potential: Thermomechanics and defects in bcc ferromagnetic iron, Physical Review Materials **2**, 013808 (2018).

[11] F. Leibfried, V. Dutordoir, S. John, and N. Durrande, A tutorial on sparse gaussian processes and variational inference, arXiv preprint arXiv:2012.13962 (2020).

[12] H. J. Cunningham, D. A. de Souza, S. Takao, M. van der Wilk, and M. P. Deisenroth, Actually sparse variational gaussian processes, in *International Conference on Artificial Intelligence and Statistics* (PMLR, 2023) pp. 10395–10408.

[13] M. Binois and N. Wycoff, A survey on high-dimensional gaussian process modeling with application to bayesian optimization, ACM Transactions on Evolutionary Learning and Optimization **2**, 1 (2022).

[14] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, On the surprising behavior of distance metrics in high dimensional space, in *International conference on database theory* (Springer, 2001) pp. 420–434.

[15] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, Vol. 4 (Springer, 2006).

[16] S. Mine, M. Takao, T. Yamaguchi, T. Toyao, Z. Maeno, S. Hakim Siddiki, S. Takakusagi, K.-i. Shimizu, and I. Takigawa, Analysis of updated literature data up to 2019 on the oxidative coupling of methane using an extrapolative machine-learning method to identify novel catalysts. chemcatchem. 2021; 13 (16): 3636-3655, DOI: https://doi. org/10.1002/cctc **202100495**.

[17] J. H. Friedman, Greedy function approximation: a gradient boosting machine, Annals of statistics , 1189 (2001).

[18] T. Hastie, R. Tibshirani, J. Friedman, *et al.*, The elements of statistical learning (2009).

[19] T. Chen and C. Guestrin, Xgboost: A scalable tree boosting system, in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016) pp. 785–794.

[20] L. Breiman, Random forests, Machine learning **45**, 5 (2001).

[21] C. Edward, Rasmussen and christopher ki williams. gaussian processes for machine learning, MIT Press **211**, 212 (2006).

[22] M. Titsias, Variational learning of inducing variables in sparse gaussian processes, in *Artificial intelligence and statistics* (PMLR, 2009) pp. 567–574.

[23] M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur, Gaussian processes and kernel methods: A review on connections and equivalences, arXiv preprint arXiv:1807.02582 (2018).

[24] M. G. Genton, Classes of kernels for machine learning: a statistics perspective, Journal of machine learning research **2**, 299 (2001).

A. ALTERNATIVE PREDICTIVE MODELS CONSIDERED

A. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a supervised, non-parametric, instance-based machine learning model. Thus, the algorithm does not initially assume a predetermined analytical form for the studied function (non-parametric) and learns from the available data about it (instance-based) [18]. The operation of KNN is based on predicting a new function value based on the K closest instances, relying on a distance metric. Among the most commonly used metrics for calculating distance are: Euclidean distance, Manhattan distance, and Minkowski distance [18]. The Euclidean distance, given by equation 30, corresponds to the usual distance measure between two points. The Manhattan distance, sometimes also called city block distance, is given by equation 31 and corresponds to the sum of the differences between the coordinates of the considered vectors. Finally, equation 32 corresponds to the Minkowski distance. For values $p = 1$ the Minkowski distance simplifies to Manhattan distance, and for $p = 2$ it simplifies to Euclidean distance [18].

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \left(\sum_{i=1}^{n} (x_i - x_i')^2 \right)^{\frac{1}{2}} \tag{30}$$

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \sum_{i=1}^{n} |x_i - x_i'| \tag{31}$$

$$d(\boldsymbol{x}, \boldsymbol{x'}) = \left(\sum_{i=1}^{n} |x_i - x_i'|^p \right)^{\frac{1}{p}} \tag{32}$$

Once the distance metric is selected, the K nearest neighbors corresponding to the smallest distances are selected. From these, the predicted value is calculated as the average of the K neighbors:

$$y_{pred} = \frac{1}{K} \sum_{i=1}^{K} y_i \tag{33}$$

In the KNN algorithm, training doesn't involve learning a function. Training actually stores the data, based on what is predicted (lazy learning) [18].

In this method, the choice of the value of K is crucial. Choosing a very small value for K can cause the model to overfit, while selecting a very large K can cause the opposite, underfitting. Additionally, to avoid problems when working with KNN, it is common to perform prior normalization of the data. This prevents some variables from dominating the distance calculation, which could influence the results obtained [18].

B. Gradient Boost Regression

Gradient Boost Regression (GBR) is a machine learning model based on the boosting technique to create a predictive model. Boosting consists of creating a complex model by sequentially combining multiple simple models (weak learners), usually shallow decision trees [17]. This allows each new model to correct the errors of the previous one, improving the prediction result. The goal of GBR is to find a function $F(x)$ that maps points in the function's domain space to predictions. This is achieved by minimizing a loss function, typically the Mean Squared Error (MSE). This function measures the difference between the predicted value and the true function value. Generally, the objective of GBR can be mathematically expressed as [17]:

$$F^* = \arg\min \sum_{i=1}^{n} L(y_i, F(\boldsymbol{x}_i)) \tag{34}$$

Where F^* denotes the optimal prediction function. To achieve this, GBR uses the technique of functional gradient descent. This technique is similar to that used in hyperparameter optimization but is carried out in function space. Using this method, at each iteration the prediction function is updated with a weak learner that approximates the negative gradient of the loss function with respect to the model estimated at that iteration [18]. Considering all this, after M iterations the final predictive model is:

$$F_M^* = F_0 + \sum_{m=1}^{M} \nu \gamma_m h_m(\boldsymbol{x}) \tag{35}$$

Where $h_m(x)$ denotes each of the weak learners, whose contributions are controlled by parameters γ and ν, where the latter corresponds to the learning rate. At each iteration, the residuals, i.e., the negative gradients of the loss function, are computed following the rule:

$$r_i^m = -\left[\frac{\partial L\big(y_i, F(\boldsymbol{x}_i)\big)}{\partial F(\boldsymbol{x}_i)} \right]_{F=F_{m-1}} \tag{36}$$

The new weak learner is trained to predict this gradient, and the optimal coefficient γ_m is obtained simultaneously by applying:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\big(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma h_m(\boldsymbol{x}_i)\big). \tag{37}$$

In this model, therefore, obtaining good predictions depends on the correct tuning of a series of model hyperparameters, among which the learning rate, the number of iterations and thus the number of weak learners used, the loss function, and the subsample or fraction of data used to train each weak model stand out [18].

C. XGBoost

Extreme Gradient Boosting (XGBoost) is a supervised machine learning algorithm that, like GBR, uses gradient boosting as a basis for generating a predictive model. The method is an expansion of traditional gradient boosting methods through the use of an optimized algorithm and a system of levels that allows better model scalability [19].

XGBoost uses sequential boosting where each regression tree is optimized to minimize residual errors of previous decision trees (weak learners). A learning rate is used to control each tree's contribution to the model. Additionally, to avoid overfitting, small learning rates are usually selected or their value is adjusted with techniques such as learning decay, which consists of a progressive drop in the learning rate [19]. At each iteration, a new tree that minimizes a regularized objective function is added. This objective function consists of two components. On one hand, we have a differentiable loss function, typically Mean Squared Error (MSE), which measures how well the predicted data fits the observed data. On the other hand, we have a regularization term that penalizes the complexity of each tree in the model [19]. Mathematically, the objective function is given by:

$$F_{Objective} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t) \tag{38}$$

In this equation, $L(y_i, \hat{y}_i)$ denotes the differentiable loss function evaluated at the observed value y_i and the current model prediction $\hat{y}_i$. On the other hand, $\Omega(f_t)$ denotes the regularization term of the T trees considered by the model. The regularization term of each tree is given by the following expression:

$$\Omega(f_t) = \gamma \cdot N_{leaves}(f_t) + \frac{1}{2}\lambda \sum_{j=1}^{L_t} \omega_j^2 \tag{39}$$

In this equation, $N_{leaves}(f_t)$ denotes the number of leaves of tree f_t, λ is a regularization term, L_t denotes the number of leaves of the tree in iteration t, ω_j is the prediction value of leaf j of the considered tree, and γ is a coefficient responsible for penalizing the addition of new leaves.

The optimization at each iteration of the objective function is obtained through a second-order Taylor series expansion of the differentiable error function around the previous prediction, $\hat{y}_i^{(t-1)}$. This expansion is given by:

$$L(y_i, \hat{y}_i) \approx L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \tag{40}$$

Where $g_i = \frac{\partial L(y_i,\hat{y}_i)}{\partial \hat{y}_i}|_{\hat{y}_i^{(t-1)}}$ and $h_i = \frac{\partial^2 L(y_i,\hat{y}_i)}{\partial \hat{y}_i^2}|_{\hat{y}_i^{(t-1)}}$. Applying this optimization, the objective to minimize becomes:

$$F_{objective}^{(t)} \approx \sum_{i=1}^{n} \left[g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i) \right] + \Omega(f_t) \tag{41}$$

D. RFR

Random Forest Regression (RFR) is a supervised learning model based on the random forest principle. Random forest is an ensemble learning method that combines predictions from multiple decision trees to generate a more accurate and stable prediction [20]. The Random Forest Regression algorithm begins by creating M decision trees. Each tree divides the input function space into different regions. Each of these regions is fitted to a constant value. Thus, the total prediction function of each decision tree is given by:

$$F_m(\boldsymbol{x}) = \sum_{j=1}^{J} c_j 1(\mathbf{x} \in R_j) \tag{42}$$

Where j is an index running over all regions of the space, denoted as R_j. On the other hand, c_j is the predictive constant calculated by the tree for region R_j, and $1(\boldsymbol{x} \in R_j)$ is an indicator function that equals 1 if $\boldsymbol{x}$ is contained in the region and 0 otherwise. Each decision tree is trained using data obtained by bootstrap sampling. This method selects rows of data from the original dataset to form new training datasets for each tree [20]. Additionally, feature sampling is performed, which is based on selecting a random subset of features or input variables to build each tree, ensuring diversity in the model. Considering all this, the final prediction made by RFR is given by:

$$\hat{F}(\boldsymbol{x}) = \frac{1}{M} \sum_{m=1}^{M} F_m(\boldsymbol{x}) \tag{43}$$

Where $F_m(\boldsymbol{x})$ denotes the prediction of each individual decision tree.

B. GAUSSIAN PROCESS CODES

In this section, the codes developed for the Gaussian Processes used to obtain the results are shown and briefly explained. All the codes to be displayed are programmed in Python using the gpytorch and pytorch libraries, which are specially designed for working with Gaussian Processes and tensors, respectively. We can begin with the Full GPs, whose codes are shown below:

```python
class GPModel(ExactGP):
    def __init__(self,train_x,train_y,likelihood):
        super(GPModel, self).__init__(train_x,
            ↪ train_y,likelihood)
```

```
 4        self.mean_module=gpytorch.means.ConstantMean
              ↪ ()
 5        self.covar_module=gpytorch.kernels.
              ↪ ScaleKernel(RBFKernel())
 6
 7    def forward(self, x):
 8        mean_x=self.mean_module(x)
 9        covar_x=self.covar_module(x)
10        return gpytorch.distributions.
              ↪ MultivariateNormal(mean_x,covar_x)
```

```
 1 class GPTmodel_2(ExactGP):
 2   def __init__(self,train_x,train_y,likelihood,nu):
 3     super(GPTmodel_2,self).__init__(train_x,train_y,
           ↪ likelihood)
 4     self.mean_module=gpytorch.means.ConstantMean()
 5     self.covar_module=gpytorch.kernels.ScaleKernel(
           ↪ MaternKernel(nu=nu))
 6
 7   def forward(self, x):
 8     mean_x=self.mean_module(x)
 9     covar_x=self.covar_module(x)
10     return gpytorch.distributions.MultivariateNormal
           ↪ (mean_x,covar_x)
```

The first code corresponds to the Full GP with an RBF kernel, while the second is for a Full GP with a Matern kernel with smoothness factor ν. These models were programmed using classes. These classes were built based on two functions.

On the one hand, the init function (lines 2 to 5) aims to initialize the Gaussian Process. In this function, the two basic structural components of any Gaussian Process are defined: the mean function and the kernel. Through line 4, we define the mean function of the Gaussian Process, in this case, ConstantMean. This is used when the data does not exhibit a dominant trend. Mathematically, it is given by $m(x) = c$, where c is a constant. Thus, c becomes a new hyperparameter of the model that will be learned during its training.

On the other hand, revisiting the codes, we can see in the lines corresponding to the kernel definitions (line 5) that we not only consider the Matern or RBF kernel, but also a ScaleKernel. The Scale Kernel as seen before does not modify the structure of the contained kernel, which remains the primary one. The ScaleKernel is used solely as a modulator of the values provided by the contained kernel. Its inclusion allows the model to learn a global scaling factor for the covariances, which can help improve numerical stability and model fitting.

Additionally, the forward function is used to construct the multivariate Gaussian associated with the Gaussian Process from its two components.

Finally, it is also worth mentioning the likelihood parameter, or likelihood function, responsible for modeling the noise associated with the observations. This was defined as an external parameter to allow flexible definition outside the model. In our case, the function used was GaussianLikelihood, which models the noise using a Gaussian distribution with zero mean and variance σ^2:

```
1 likelihood=gpytorch.likelihoods.GaussianLikelihood()
```

These models were trained using the following function:

```
 1 def train_model(model,likelihood,num_epochs,lr,
       ↪ x_train,y_train):
 2   model_params=list(model.parameters())
 3   likelihood_params=list(likelihood.parameters())
 4   all_params=list(set(model_params +
         ↪ likelihood_params))
 5   optimizer=torch.optim.Adam(all_params, lr=lr)
 6   mll=gpytorch.mlls.ExactMarginalLogLikelihood(
         ↪ likelihood,model)
 7
 8   epocas=[]
 9   perdida=[]
10
11   model.train()
12   likelihood.train()
13
14   for i in range(num_epochs):
15     optimizer.zero_grad()
16     output=model(x_train)
17     loss=-mll(output,y_train)
18     loss.backward()
19     optimizer.step()
20     perdida.append(loss.item())
21     epocas.append(i+1)
22
23   return epocas,likelihood,perdida,model
```

The first thing we do in this function is define a variable for the complete set of the GP's hyperparameters, which we will optimize using the function. These hyperparameters consist of the sum of the hyperparameters from model (a variable designated for the GP constructed with the previous classes) and the hyperparameters of the likelihood function. This entire process is carried out from lines 2 to 4.

After this, we define the optimizer, in our case the Adam optimizer, to automate the optimization process. This optimizer is programmed to optimize hyperparameters by minimizing an objective. We also define the Log Marginal Likelihood (LML). This definitions are done in lines 5 and 6, respectively. As we can see, we are calling the Log Marginal Likelihood (LML), not the Negative Log Marginal Likelihood (NLML) directly. This is because the LML (negative of the NLML) is the function implemented in gpytorch. To correctly maximize the LML using the Adam optimizer we have to take the negative of its value. This is the same as calculating the value of the NLML and minimizing it using the Adam optimizer.

Additionally, before entering the loop, in lines 11 and 12, we activate the training mode for both the model and the likelihood. Once this is done, we can begin the actual training process using the loop that spans from lines 14 to 21. In this loop, all hyperparameters are optimized by minimizing the NLML.

Once completed, we return the trained model and likelihood, as well as a list containing the NLML value for each epoch of training.

Predictions were made using the following code:

```
1  def evaluate_model(model,likelihood,x_test):
2    model.eval()
3    likelihood.eval()
4    with torch.no_grad():
5      test_preds=likelihood(model(x_test))
6      preds=test_preds.mean
7      var=test_preds.variance
8    return preds,var
```

In the first two lines, we switch both the model and the likelihood to evaluation mode. Following this, in line 5, we first evaluate the set of points with the GP model, $model(x_{test})$, which allows us to obtain the true function values at these points. This result is then evaluated with the likelihood function, adding the modeled noise factor, thus obtaining the final multivariate distribution, the Posterior distribution that gives the predictions of the observed function. In line 6, we select only the means of this distribution, which correspond to the observed value for each point. On line 7, we select the variances of the calculated values. Finally, on line 8, we return the predictions and variances.

Once we have covered the codes for generating and training the Full GP models, we now discuss the codes corresponding to the SVGP, the Sparse GP model used:

```
1  class SparseGPModel(ApproximateGP):
2      def __init__(self, inducing_points):
3          variational_distribution=
               ↪ CholeskyVariationalDistribution(
               ↪ inducing_points.size(0))
4          variational_strategy=VariationalStrategy(
               ↪ self, inducing_points,
               ↪ variational_distribution,
               ↪ learn_inducing_locations=True)
5          super().__init__(variational_strategy)
6          self.mean_module=ConstantMean()
7          self.covar_module=ScaleKernel(MaternKernel(
               ↪ nu=2.5,ard_num_dims=inducing_points.
               ↪ size(1)))
8
9
10     def forward(self, x):
11         mean_x=self.mean_module(x)
12         covar_x=self.covar_module(x)
13         return MultivariateNormal(mean_x,covar_x)
```

The first difference is that to create this model we do not need the full training data, we only require the inducing points. Using these, we call CholeskyVariationalDistribution in line 3 which generates a multivariate variational distribution that approximates the Posterior distribution over the latent values at the inducing points.

After this, we define the variational strategy for our model in line 4, which uses the distribution generated with CholeskyVariationalDistribution. Here, through the command `learn_inducing_locations=True`, we specify that the positions of the inducing points should be treated as optimizable variables, allowing for the best possible approximation. The number of inducing points remains fixed as defined by the input set and is used to define the variational distribution in line 3.

Next, we proceed to define the mean function and kernel of the Gaussian Process. As with the Full GP case, we maintain the ConstantMean function. For the kernel, we use a Matern kernel with smoothness factor $\nu = \frac{5}{2}$. The Automatic Relevance Determination (ARD) is also defined in this kernel line. Finally, in the forward function, we define our final multivariate Gaussian distribution.

The function used for training this model is shown below:

```
1  def train_model_Sparse(model,likelihood,num_epochs,
       ↪ lr,x_train,y_train):
2    model_params=list(model.parameters())
3    likelihood_params=list(likelihood.parameters())
4    all_params=list(set(model_params +
         ↪ likelihood_params))
5    optimizer=torch.optim.Adam(all_params, lr=lr)
6    mll=gpytorch.mlls.VariationalELBO(likelihood,model
         ↪ ,num_data=y_train.size(0))
7
8    epocas=[]
9    perdida=[]
10
11   model.train()
12   likelihood.train()
13
14   for i in range(num_epochs):
15     optimizer.zero_grad()
16     output=model(x_train)
17     loss=-mll(output,y_train)
18     loss.backward()
19     optimizer.step()
20     perdida.append(-loss.item())
21     epocas.append(i+1)
22
23   return epocas,likelihood,perdida,model
```

The code used to train the SVGP is the same as that used for the Full GPs, with one key difference. In this case, we use the ELBO, instead of the NLML as was established before.

As we can see, when calculating the ELBO value in line 17, we take its negative. This is because the Adam optimizer minimizes the objective as we have seen before. Thus, we must negate the ELBO to maintain consistency in the optimization process, even though, in essence, we are maximizing the ELBO.