Cost-Effective Empirical Performance Modeling

Marcus Ritter, Benedikt Naumann, Alexandru Calotoiu, Sebastian Rinke, Thorsten Reimann, Torsten Hoefler, and Felix Wolf

***Abstract*—Performance models help us to understand how HPC applications scale, which is crucial for efficiently utilizing HPC resources. They describe the performance (e.g., runtime) as a function of one or more execution parameters (e.g., problem size and the degree of parallelism). Creating one manually for a given program is challenging and time-consuming. Automatically learning a model from performance data is a viable alternative, but potentially resource-intensive. Extra-P is a tool that implements this approach. The user begins by selecting values for each parameter. Each combination of values defines a possible measurement point. The choice of measurement points affects the quality and cost of the resulting models, creating a complex optimization problem. A naive approach takes measurements for all possible measurement points, the number of which grows exponentially with the number of parameters. In our earlier work, we demonstrated that a quasi-linear number of points is sufficient and that prioritizing the least expensive points is a generic strategy with a good trade-off between cost and quality. Here, we present an improved selection strategy based on Gaussian process regression (GPR) that selects points individually for each modeling task. In our synthetic evaluation, which was based on tens of thousands of artificially generated functions, the naive approach achieved 66% accuracy with two model parameters and 5% artificial noise. At only 10% of the naïve approach's cost, the generic approach already achieved 47.3% accuracy, while the GPR-based approach achieved even 77.8% accuracy. Similar improvements were observed in experiments involving different numbers of model parameters and noise levels, as well as in case studies with realistic applications.**

***Index Terms*—Performance analysis, performance modeling, high-performance computing, parallel processing, reinforcement learning, Gaussian process regression.**

I. Introduction

THE design and development of high-performance computing (HPC) applications represent a complex and costly endeavor. To achieve optimal performance on today's large-scale machines, sophisticated parallelization strategies, memory management techniques, and efficient utilization of computational resources are required, necessitating a continuous focus on performance analysis. In this process, performance models can be beneficial because they facilitate performance assessments across a vast space of execution configurations. One example of such a model is the expression of the application's execution time as a function of the number of processes and the input problem size. However, deriving such models analytically is complex and error-prone [1], [2]. Empirical performance models, by contrast, can be automatically derived from performance measurements without expert knowledge or manual analysis. They are effective for identifying scaling bottlenecks in complex codes and making performance predictions with practical accuracy [3], [4].

Extra-P [5] is an empirical modeling tool that uses a set of small-scale experiments to derive performance models for individual functions of a parallel program. In general, the experiment design, the chosen set of execution configurations for measuring performance, determines the resulting models' quality and cost. Therefore, the selection of the execution configurations, or measurement points, is an optimization problem where the search space is defined by the parameters of the execution configuration (e.g., process count and input problem size) and the range of values they can assume. This search space can contain thousands of potential measurement points depending on the number of model parameters. Although each point entails a specific cost and potential accuracy gain, understanding their interaction is crucial to selecting an optimal set that balances cost and model accuracy.

Besides the costs of conducting a performance measurement, one must also consider the effects of system noise on the measurements. For example, concurrently running jobs, network bandwidth limitations, or OS noise can significantly alter performance measurements. Especially runtime, a key performance metric, may vary substantially even between runs with the same execution configuration. To counter noise, the developers of Extra-P recommend repeating each experiment up to five times [6]. Subsequently, Extra-P uses the median of the measured metrics' values to create performance models. The number of repetitions per measurement point introduces another variable into the optimization problem, increasing the search space even further. On a system with minimal run-to-run variation, one might want to repeat each measurement only twice or not at all, whereas, in high-noise-level scenarios, more repetitions of existing measurement points might be advantageous over unseen points.

The current version of Extra-P employs a generic experiment design strategy that prioritizes cost-effectiveness, selecting the least expensive execution configurations for modeling, which we call *cheapest points first* (CPF) [6]. Additionally, each performance experiment is repeated five times to address run-to-run variation. However, the current generic strategy is too rigid to achieve optimal results. It neither trades off the cost with a sample's value nor adjusts the number of repetitions to the actual noise level. For example, how expensive an "expensive" measurement point is depends on the application being modeled. Moreover, the noise level can vary significantly between systems. Therefore, a more intelligent sample selection strategy is needed that exploits application-

Received 19 September 2024; revised 14 November 2025; accepted 13 December 2025. M. Ritter is affiliated with the Corporate Research Center of ABB AG in Mannheim, Germany. B. Naumman, and F. Wolf are with the Department of Computer Science at Technical University of Darmstadt. T. Reimann is with the University Computing Centre at Technical University of Darmstadt. A. Calotoiu and T. Hoefler are with the Department of Computer Science at ETH Zurich. S. Rinke is with the Faculty of Computer Science and Media at HTWK Leipzig.

and system-specific properties.

This paper proposes a novel measurement-point selection strategy based on Gaussian process regression (GPR). Considering variables such as the noise level and the possible parameter value ranges, the GPR-based strategy selects execution configurations to achieve an optimal trade-off between model accuracy and cost for a given modeling task and budget. The new approach replaces the generic cheapest-points-first heuristic [6]. Like the previous strategy, it starts with a predefined minimum configuration but then selects additional points and their repetitions individually for each modeling task. The selection is controlled by a feedback loop considering several problem-specific factors, including the uncertainty left by points already seen and the noise level. Compared to the CPF experiment design strategy, it improves the average model accuracy by up to 64.29% using the same modeling budget. It also outperforms four other strategies that we included in our evaluation.

The remainder of the paper is organized as follows. After introducing Extra-P in Section II, we outline our novel experiment design approach in Section III. In Section IV, we conduct a synthetic evaluation to compare the model accuracy and cost of the new GPR-based experiment design technique with the previous generic strategy. We then present six application case studies in Section V, highlighting the advantages of the GPR approach. Finally, we discuss related work in Section VI and provide a conclusion in Section VII.

TABLE I
LIST OF MATHEMATICAL NOTATIONS.

Symbol	Definition
x_i	Model parameter with index i
$\mathbf{x}_i$, $\mathbf{x}_{i+}$	Set of parameter values for x_i (modeling and evaluation)
P, P_+	Measurement point, evaluation point
$\mathbf{P}$, $\mathbf{P}_+$	Set of measurement, evaluation points
m	Number of model parameters
α_{kl}, β_{kl}	PMNF function term exponents
$\mathcal{A}$, $\mathcal{B}$	Exponent sets for the PMNF
c_k	PMNF function term coefficients
ϵ	Additional points used by the sparse modeler
n_b	Batch size of newly attained points for modeling
λ	Termination criterion for selection strategies
lb_i, ub_i	Model parameter lower and upper bounds
$G_i(x_i)$	Application-specific constraint predicates
Δx_i^k	Step size between successive parameter values
$\mathbf{S}$	Measurement point selection search space
C_w	Objective function of the optimization problem
A	Achieved model accuracy
C	Cost of a set of measurement points
w_a, w_c, w_n, w_r	Weight factors for accuracy, cost, noise, repetitions
$\overline{f}$	Mean function of the Gaussian process (GP)
$k_{\text{Matérn}}$, k_{White}	GP covariance functions (Matérn, White kernel)
N	Normal distribution of the GP
$\text{GP}(t)$	Gaussian process evaluated at point t
t	Measurement point $P(x_i)$ in $\mathbf{S}$
$d(\cdot,\cdot)$	Euclidean distance between two points
ℓ	Length scale of the GP
k_ν	Modified Bessel function of order ν
$\Gamma(\nu)$	Gamma function evaluated for ν
ν	Function smoothness
n	Noise level found in the measurements
r	Number of repetitions of a point P
B	Modeling budget

II. BACKGROUND

Performance models, such as the ones created by Extra-P, are mathematical functions that describe how the performance of a program, expressed in terms of a metric (e.g., execution time), changes as execution parameters (e.g., number of processes) vary. We obtain an empirical data set to learn an application performance model by conducting a series of performance experiments with different configuration parameters. The performance measurements in this set reflect the changes in application performance as configuration parameters x_i are modified, where i is the parameter index, such that Extra-P can discover the underlying function and automatically create a performance model. Furthermore, each of these experiments represents a specific execution configuration defined by a distinct combination of execution parameter values, which we refer to as *measurement point*. For clarity, Table I summarizes the mathematical notation used throughout this paper.

We define a measurement point $P(x_1, x_2, \ldots, x_i)$ as a unique configuration of the application's execution parameters x_i, where m is the number of parameters considered for modeling, e.g. $m = 2$ for the following example. The notation $\mathbf{x}_1 = \{4, 8, 16, 32, 64, \ldots\}$ represents a set of possible parameter values for x_1, while $\mathbf{P}(\mathbf{x}_1)$ represents a set of measurement points: $P(4)$, $P(8)$, $P(16)$, $P(32)$, $P(64)$. To derive the measurement points, we use the Cartesian product of the parameter-value sets $\mathbf{x}_i$ so that $\mathbf{x}_1 \times \mathbf{x}_2 \times \cdots \times \mathbf{x}_i = \mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$. In this process, we create pairs, combining each element from set $\mathbf{x}_1$ with every element from set $\mathbf{x}_2$. We then combine each resulting pair with every element from the next set $\mathbf{x}_i$. We repeat this process $m - 1$ times. Consequently, a set of measurement points with two parameters is noted as $\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2)$. For example, $\mathbf{x}_1 = \{4, 8\}$ and $\mathbf{x}_2 = \{32, 64\}$ would result in the points $\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2) = P(4, 32), P(4, 64), P(8, 32), P(8, 64)$.

To obtain such a set of measurement points, we vary the application's configuration parameters, e.g., the number of processes $\mathbf{x}_1 = \{4, 8, 16, 32, 64, \ldots\}$, or the problem size $\mathbf{x}_2 = \{10, 20, 30, 40, 50, \ldots\}$. Once the experiments for a sufficiently large set of measurement points have been obtained, Extra-P automatically creates a performance model. It makes a separate model for each instrumented application kernel so that scaling issues can be precisely pinpointed to locations in the source code. To create performance models, Extra-P uses the performance model normal form (PMNF) in Equation 1 [7]. It describes the effects of multiple parameters x_i on performance as a sum of terms consisting of products of polynomial and logarithmic expressions. To create a performance model, a search space of possible model hypotheses is generated by instantiating Equation 1 with different exponents α_{kl}, β_{kl} chosen from the sets $\mathcal{A} = \{0, 0.5, 1, 1.5, 2, \ldots\}$ and $\mathcal{B} = \{0, 1, 2, \ldots\}$, respectively. The sets $\mathcal{A}$ and $\mathcal{B}$ are predefined but can be adjusted by the user depending on their modeling requirements. For example, one might want to use negative exponents or prohibit using logarithmic terms. We then calculate the coefficients c_k of the hypothesis using linear regression. Finally, the best model is identified using cross-validation, choosing the hypothesis with the smallest symmetric mean absolute percentage error (SMAPE). For an in-depth explanation of the modeling process, please see Calotoiu et

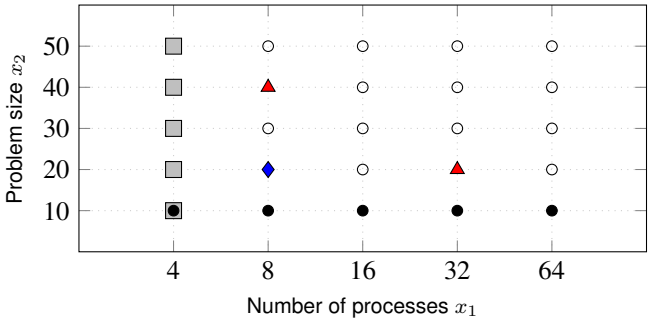

Fig. 1. Illustration of the matrix of measurement points for 2-parameter analysis. The filled black circles represent a set of measurements that would be sufficient to generate a single-parameter model for how the number of processes affects performance. The gray squares are an example of a subset of points that is sufficient to create a performance model for how x_2 affects performance. The blue diamond represents an extra point that would be used by the CPF strategy to increase model accuracy, whereas the red triangles could be points chosen by the GPR strategy.

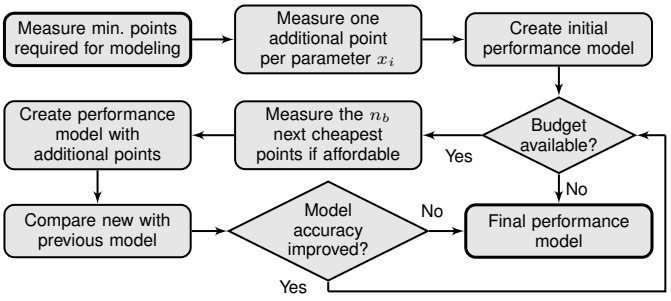

Fig. 2. The measurement-point selection process of the CPF strategy.

al. [7]. Note that the PNMF produces monotonically increasing functions, particularly for weak-style scaling. For strong-style scaling, the modeling problem can be easily adapted to match the PNMF, for example, by modeling resource consumption (e.g., core hours) instead of wall-clock time.

$$f(x_1, \ldots, x_i) = \sum_{k=1}^{h} c_k \cdot \prod_{l=1}^{i} x_l^{\alpha_{kl}} \cdot log_2^{\beta_{kl}}(x_l) \qquad (1)$$

The choice of measurement points determines the extrapolation accuracy of the resulting models. The first version of Extra-P required performance measurements of five unique values per model parameter and all their combinations. Thus, it requires an m-dimensional *dense* matrix of measurement points – with a total of 5^m elements [7]. Because every experiment had to be repeated five times to account for run-to-run variation, an educated guess based on anecdotal evidence, the total number of experiments was $5^{(m+1)}$.

Introducing a new, more cost-efficient measurement-point selection strategy reduced the number of measurement points to roughly $5 \cdot m$ [6]. To set it apart from the previously used dense matrix of measurement points, we coined the overarching term *sparse modeling* for this strategy. Instead of measuring every combination of the five values per parameter, a minimum of five different points per parameter had to be obtained. Those five points varied only a single parameter and kept all others constant – to save cost at their lowest possible values. These correspond to the leftmost column and lowest row in Figure 1. This is enough to create separate models for each parameter. Then, a few extra points had to be collected to establish the relationship among those parameters and improve the accuracy of the resulting multi-parameter models. Including repeated experiments, the total number of experiments dropped from $5^{(m+1)}$ to $5 \cdot (m+1) + 5 \cdot \epsilon$ with ϵ being the number of extra points, again including the five repetitions per point.

The freedom in selecting the extra points beyond the initial column and row in Figure 1 poses a discrete optimization problem, where the search space is defined by the model parameters x_i and their possible values. To find an efficient strategy for selecting the additional measurement points that represent the optimal trade-off between retaining model accuracy and reducing the overall modeling cost, we trained an artificial agent on the task of parameter-value selection. We created a training environment that simulates the task of measurement-point selection as a Markov decision process and uses a reward function that weighs the achieved model accuracy against the cost, which is then presented as a reinforcement signal to the agent. We defined accuracy as the model's extrapolation accuracy at a point outside the scale used for modeling. We defined cost as the total number of core hours needed to run a performance experiment for a specific measurement point. Consequently, the cost for each measurement point is different. To quantify the improvement compared to the dense-matrix approach, we compared the cost of the full matrix of points, which we assume to be 100%, with the cost of the points selected by the artificial agent in percent. We trained the agent using hundreds of thousands of synthetically generated performance functions, considering different types of modeling problems, numbers of parameters, parameter-value series, and different amounts of artificially induced noise. As a result of our training efforts, the agent learned a generic strategy that effectively reduces the modeling cost while retaining accuracy.

Observing the trained agent, we derived a generic measurement point selection heuristic, the cheapest-points-first (CPF) strategy. The CPF strategy can be broken down into a nine-step process outlined in Figure 2. First, each model parameter's five cheapest available measurement points are measured, where all other parameters' values remain constant. In a two-parameter scenario, this corresponds to the column and row highlighted in Figure 1. To counter the effects of system noise, we repeat the measurements for each point five times. Second, we measure one additional point per model parameter outside these series (including five repetitions), choosing the cheapest available points. Once the data specified above has been acquired, Extra-P creates an initial performance model. If the compute budget is exhausted, the final model has been found. If not, we measure n_b more points, always choosing the next cheapest point (including their repetitions) as long as we can afford to continue. Remember that because of the monotonically increasing nature of PMNF functions; measurement points are cheaper the closer they are to the lower left corner of the matrix in Figure 1. We acquire new points in

batches of size n_b to reduce queue time. Otherwise, we would collect them one by one, which would lead to the best results. Using these additional points, we create a new performance model and compare how well the new model fits the measured data compared to the previous edition using the symmetric mean absolute percentage error (SMAPE). If we do not see an improvement for λ iterations of adding more points, we have found the final model. However, if we see an improvement, we keep collecting further points until we either run out of budget or decide to stop. Finally, a more thorough quantitative analysis suggested reducing the repetitions needed to counter system noise from five to four, as the fifth repetition provided only marginal benefits.

With the CPF strategy, in our synthetic evaluation considering two model parameters and $\pm 5\%$ of artificially induced noise, we found that using only 10% of the modeling budget results in a 9.6% loss in accuracy relative to the dense approach. Other budget levels (e.g., 20%) and experiments with different numbers of model parameters yielded similar trade-offs. For an in-depth description of this strategy and its results, the reader may refer to Ritter et al. [6].

III. Approach

The following Section introduces our novel GPR-based measurement point selection strategy. First, we outline the optimization problem, its decision variables, the search space, and the objective function. We then provide a detailed description of using Gaussian progress regression to identify an optimal set of points for a specific modeling task.

A. Optimizing measurement point selection

Identifying an optimal set of measurement points for a specific modeling task to maximize model accuracy while reducing cost and ensuring noise resilience is a classical optimization problem. The decision variables of the problem are the applications' configuration parameters, i.e., the model parameters x_i. Many commonly modeled application parameters, such as the number of processes or the input problem size, can only take specific values and are thus discrete. The problem's search space is an n-dimensional space of potential measurement points. The size of the search space and its dimensionality are defined by the number of model parameters m and their value ranges. Since the model parameters are discrete, the measurement points derived from them and the optimization problem are also discrete. The value range of a parameter x_1, e.g., the number of processes, is defined by its lower bound lb_1 and upper bound ub_1, which are determined by factors such as the used system, hardware, application, and the available modeling budget. For example, it might not make sense to analyze the performance of an application using less than 32 processes $lb_1 = 32$. Conversely, runs with more than 1,024 processes ($ub_1 = 1,024$) are not possible due to the evaluation system's size limitations.

Furthermore, application-specific constraint predicates $G_i(x_i)$ may narrow the choices for the values of a particular model parameter x_i. LULESH [8], a shock hydrodynamics code, for example, accepts only cubic integer values such as $x_1 = \{2^3, 3^3, 4^3, 5^3, \dots, p_i^3\}$ as input for the number of processes. In this case, we introduce a constraint $G_1(x_1) : x_1 = p_k^3$ where $p \in \mathbb{Z}$ and k is the index of the value within this parameter-value series. We then divide the range defined by the parameters bounds $lb_i \leq x_i \leq ub_i$ into intervals of size Δx_i^k. The step size Δx_i^k represents the interval between consecutive (discrete) values that x_i can assume and may vary depending on the defined constraints $G_i(x_i)$. For our LULESH example, we discretize the range $[32, 1\,024]$ into intervals of size $\Delta x_1^k = p_{k+1}^3 - p_k^3$, where p_{k+1} is the value following p_k. Consequently, $\mathbf{x}_1 = \{64, 125, 216, 343, 512, 729, 1\,000\}$ is the set of discrete parameter values x_1 can take. Of course, this process needs to be repeated for each model parameter.

Based on these definitions, we can define the search space of the optimization problem as shown in Equation 2. Considering only two model parameters, the number of processes x_1 and the problem size x_2, the search space for LULESH can be described as outlined in Equation 3. Using the parameter-value series $\mathbf{x}_1$ and $\mathbf{x}_2 = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$, we can determine the potential measurement points in $\mathbf{S}$ one can choose, e.g., $\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2) = \{P(64, 10), P(64, 20), P(125, 10), \dots\}$.

$$\mathbf{S} = \{(x_1, x_2, \dots, x_i) \mid lb_i \leq x_i \leq ub_i,\ G_i(x_i)\} \tag{2}$$

$$\begin{aligned} \mathbf{S} = \{(x_1, x_2) \mid\ & 32 \leq x_1 \leq 1\,024, \\ & 10 \leq x_2 \leq 100, \\ & G_1(x_1) : x_1 = p_i^3 \wedge p \in \mathbb{Z}, \\ & G_2(x_2) : x_2 \bmod 10 = 0\} \end{aligned} \tag{3}$$

$$C_w(\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_i)) = w_a \cdot A - w_c \cdot C \tag{4}$$

Next, we define the objective function of the optimization problem. Our objective function, which is outlined in Equation 4, is a reward function that weighs the achieved model accuracy A of an arbitrary set of measurement points $\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_i)$ against their cost C. We define model accuracy as a measure of how well the created model fits the training data set. Therefore, we use the SMAPE metric introduced in Section II, which considers the model's error and how many variables were used to achieve this error. This definition of model accuracy is different from the one that we introduced in Section II and the one that we use for the evaluation of the GPR approach in Sections IV and V. It is important to note the difference between these metrics, as one of them, the extrapolation accuracy, is used to quantify the quality of the created models, and the other is used as a criterion for the measurement point selection. This differentiation is necessary, as we can not use unseen points for the selection process. We define the cost of a measurement point $P(x_i)$ as the number of core hours required to perform this configuration's performance measurement. The number of core hours can be calculated by multiplying the number of used cores with the measured application runtime. We assume that applications use frameworks such as OpenMP for threading and MPI for message passing to utilize the available parallelism.

Following this principle, we can maximize model accuracy while minimizing cost. Since we optimize two objectives simultaneously, this leads to a trade-off known as Pareto efficiency [9]. Pareto efficiency is achieved if improving upon one objective can only be accomplished at the expense of another. We explore all Pareto-optimal solutions in the given search space to identify the most suitable trade-off between accuracy and cost. Furthermore, by changing the values of w_a and w_c, one can adjust the trade-off between accuracy and cost and thus decide which Pareto-optimal solution is preferred for a specific modeling problem and available budget [10].

B. Gaussian process regression as a guide

While the CPF strategy drastically reduces the average modeling cost, we have observed that its generic applicability comes at the expense of finding only suboptimal solutions for specific modeling problems. This is due to its disregard for previously measured application configurations and the quality of the resulting models. We employ GPR to select measurement points to solve this problem and identify an optimal set of experiment configurations for a specific modeling problem.

GPR is a non-parametric Bayesian approach to regression that is frequently applied in machine learning. It takes a set of hyperparameters and measurements as input and returns a Gaussian process (GP). The generated GP consists of a mean function $\overline{f}$ and a covariance function k, often called a kernel. Together, they define a normal distribution $N(\overline{f}, k)$ at every point t such that $\mathrm{GP}(t) = N(\overline{f}(t), k(t, t'))$ [11]. In our case, t is a measurement point $P(x_i)$ in the search space $\mathbf{S}$. The mean function can be interpreted as the most likely solution to the given problem, and the covariance function as the probabilistic range, where the solution most probably is, given the hyperparameter-defined assumptions.

Compared to other machine learning methods, GPR has a few significant benefits, which is why it is better suited for optimizing the process of measurement point selection. First, it works very well on small data sets. Other methods, such as deep neural networks (DNNs), require much larger training data sets to produce accurate predictions. Second, Gaussian processes work very well when dealing with noisy data, irregularly spaced observations, and situations where uncertainty estimation is essential. System noise will always influence the data obtained from empirical performance measurements. Thus, it is crucial to model this noise as a component of the created performance model. Even though the intervals for the measurement point selection are fixed because the search space is discrete, we have to deal with data gaps between successive observations and variable sampling frequencies, resulting in an uneven distribution of training data points.

Furthermore, a GP provides an interpretable model by providing a measure of uncertainty for its predictions, i.e., a probability distribution indicating how likely a particular prediction is close to the ground truth. The covariance function encodes the assumption that we can learn the underlying function by defining the similarity of two points, assuming that similar points should have similar target (metric) values. Therefore, the probability values of the covariance function can be used to investigate which measurement points will lead to improved model accuracy by analyzing the target values of already measured similar points. Finally, GPs require fewer computational resources than decision trees, random forests, or DNNs. Even though GPR could be used as a modeling technique by creating a performance model from the Gaussian processes mean function, our evaluation showed that it provides less accurate results than Extra-P's standard modeling approach. Therefore, it is not viable as a modeling approach for performance and scalability analysis.

The idea behind using GPR to guide measurement point selection is to estimate the expected gain in model accuracy by adding a not yet measured application configuration to the set of available modeling data. Using the set of available measurements, one can generate a Gaussian process and identify the maxima in its covariance function, thus identifying the best additional measurement points at the current point in time. Our approach uses a stationary Matérn $k_{\text{Matérn}}$ covariance function [11]. The Matérn kernel is a stationary covariance function and a generalization of the radial basis function (RBF) kernel. When choosing the kernel for our GPR approach, we considered several factors, such as smoothness, periodicity, trends, and noise levels. Some types of covariance functions, e.g., periodic or exponential ones, are unsuitable for our use case, as we do not expect any periodic or exponential performance behavior. After evaluating various covariance functions, we selected the Matérn kernel due to its superior performance and suitability for modeling smooth functions with gradual slope changes, as the parameter ν can control its degree of smoothness.

Equation 5 shows the Matérn kernel, where $d(\cdot, \cdot)$ is the Euclidean distance between two inputs, i.e., measurement points. ℓ is the characteristic length-scale of the process, expressing how close two points $P(x_i)$ and $P(x_i)'$ have to be to influence each other significantly and must be $\ell > 0$. k_ν is the modified Bessel function of order ν and $\Gamma(\nu)$ is the gamma function evaluated for ν. The parameter ν controls the smoothness of the resulting function. For $\nu \to \infty$, the Matérn kernel converges to the RBF kernel. For our analysis, we used $\nu = 1.5$, a scalar value for the length scale of $\ell = 1.0$, and the length scale bounds $\ell_{low} = 1\text{e}{-5}$, $\ell_{up} = 1\text{e}5$.

$$k_{\text{Matérn}}(P(x_i), P(x_i)') = \left(\frac{\sqrt{2\nu}}{\ell} d(P(x_i), P(x_i)')\right)^{\nu} \cdot \frac{1}{\Gamma(\nu) 2^{\nu-1}} \cdot k_\nu \left(\frac{\sqrt{2\nu}}{\ell} d(P(x_i), P(x_i)')\right) \quad (5)$$

$$k_{\text{White}}(P(x_i), P(x_i)') = \begin{cases} n & \text{if } P(x_i) = P(x_i)' \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

In addition, we use the white kernel in Equation 6 to explain the noise of the signal where n is the mean noise level found in the measurements. Tuning this parameter corresponds to estimating the system noise level. Finally, we sum the Matérn and the white kernel to obtain our covariance function $k(P(x_i), P(x_i)') = k_{\text{Matérn}} + k_{\text{White}}$.

However, before we can make any prediction, we have to train the GP. The more training data, the better its predictions

will be. The requirement for training data aligns well with Extra-P's sparse modeling approach, which mandates at least five distinct values per model parameter while keeping all other parameters fixed, as illustrated in Figure 1. Therefore, we measure the cheapest available application configurations that satisfy the sparse modelers' modeling requirements to obtain the necessary training data for the GP. We suggest repeating each of these configurations so that our approach can estimate the noise level on the measurements and model it using the white kernel. If these repetitions are not available, the initial points suggested will correspond to those repetitions.

In contrast to the CPF strategy, the GPR approach also considers the number of repetitions of a measurement point. Thus, it will suggest a specific measurement point, including the repetition number. This increases the search space even further but enables the GPR method to reason whether it is better to measure an unseen measurement point or repeat the measurement of an already measured point. More repetitions are not always beneficial and can sometimes even reduce model accuracy. Furthermore, providing the GPR strategy with this freedom enables better use of the available budget.

$$C_w(P(x_i)) = \frac{C(P(x_i))^2 \cdot w_c}{k_{\text{Matérn}}(P(x_i), P(x_i)')^2 \cdot w_a} \tag{7}$$

To identify the best additional measurement points, we use the function in Equation 7 to rate and rank all remaining points in the search space (including their repetitions) by calculating their weighted cost. Equation 7 is the practical adaptation of the objective function introduced in Section III-A, which weighs the achieved model accuracy of a set of measurement points $\mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$ against their cost. Using the weight factors w_c and w_a, we can adjust which Pareto-optimal solution we choose based on the modeling problem, available budget, and other requirements.

The function defines the weighted cost $C_w(P(x_i))$ of a measurement point $P(x_i)$, where $C(P(x_i))$ is the point's cost in core hours and w_c the weight factor. The lower the cost, the better the measurement point. Of course, we do not know in advance the exact cost of a measurement point that has yet to be executed. However, we can predict the expected cost using a performance model created by Extra-P that describes the runtime of the application as a function of the model parameters. After predicting the runtime for the measurement point, we multiply it by the number of processes used to calculate the core hours. We create this model using the same set of measurement points we used to train the GP. To predict the cost of a measurement point as accurately as possible, every time the GPR strategy selects a new point that does not exhaust the available modeling budget, we add this point to the training data set, retrain the GPR, and create a new performance model.

$k_{\text{Matérn}}(P(x_i), P(x_i)')$ is the covariance function of the Gaussian process for the points $P(x_i)$ and $P(x_i)'$, where $k(x) = k(x, x = y)$ and w_a is the weight factor for the model accuracy. By predicting the covariance of the predictive distribution, we obtain a probabilistic value that describes the covariability of a potential additional measurement point and those that have already been measured. This information

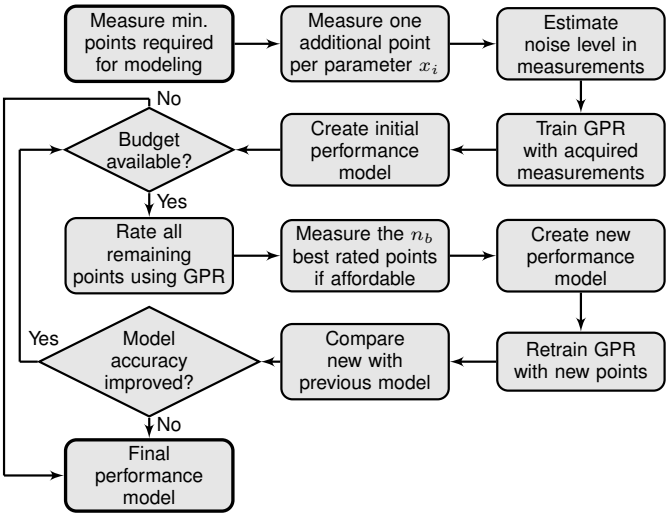

Fig. 3. The measurement point selection process of the GPR strategy.

enables us to quantify the possible improvement in model accuracy for each measurement point in the search space.

To reason about the trade-off between measuring new points and repeating the measurement for an already seen point, we use the weight function in Equation 8. First, we calculate a weight factor w_n for the mean percentage noise level found in the measurements n. Second, we calculate a weight factor w_r considering the number of repetitions r of a point $P(x_i)$ that have already been measured. We set the following bounds for their values $0 \leq n \leq 100$ and $1 \leq r \leq 10$. Finally, we sum both values to obtain the weight factor for the measurement point cost w_c. The weight function w_c is optimized to prioritize new points at low noise levels, as repeated measurements offer little value when variation is slight. At high noise levels, repetitions are preferred to mitigate the effects of noise.

$$\begin{aligned} w_n &= -\tanh\left(\tfrac{1}{4} \cdot n - \tfrac{5}{2}\right) \\ w_r &= 2^{\frac{1}{2} \cdot r - \frac{1}{2}} \\ w_c &= w_n + w_r \end{aligned} \tag{8}$$

Based on these definitions, we define the measurement selection process of the GPR strategy as outlined in the flowchart in Figure 3. First, the five cheapest available measurement points for each model parameter are measured, where all other parameters' values remain constant, as illustrated in Figure 1. Second, we measure one additional measurement point per model parameter that is not part of these series and uses a different parameter-value combination. For these additional points, we also choose the cheapest available measurement points. To enable an analysis of the system noise, we repeat each of these measurements twice. Third, we estimate the average noise level in our measurements by analyzing the divergence from the arithmetic mean metric value in percent. Next, we use the acquired measurements to train the GPR. We then let Extra-P create an initial performance model. Once the budget is exhausted, we will reach our final performance model. If there is still budget left, we use GPR to rate all remaining (not yet measured points) in the search space. We

then measure the n_b best-rated measurement points. While measuring several new points at once may reduce accumulated queue-waiting times for our experiments, the full potential of the GPR method is only achieved when adding points one by one. Subsequently, we use the new measurements to retrain the GPR and create a new performance model. Finally, we compare the accuracy of the new and the old models, checking how well each model fits the conducted performance measurements. For this purpose, we use the SMAPE metric introduced in Section II. If there is an improvement in model accuracy, we go back and identify additional points if there is still some budget left. However, if there has been no improvement in model accuracy for λ iterations of adding more points, we found our final model. The user can adjust the value of λ depending on the available budget and complexity of the modeling task. For modeling problems with two parameters, e.g., we use $\lambda = 3$ as a standard value.

IV. Synthetic Evaluation

To assess the effectiveness of our novel GPR-based measurement point selection strategy, we performed an extensive synthetic analysis, comparing its model accuracy, budget efficiency, and robustness to noise against six alternative strategies: using a complete matrix of measurement points, the currently used CPF strategy, a hybrid method, random search, grid search, and a Bayesian optimization approach.

A. Evaluation methodology

For our synthetic data evaluation, we generate artificial performance functions considering different numbers of model parameters $m = \{2, 3, 4\}$, amounts of artificially induced noise $n = \{\pm 1\%, \pm 2\%, \pm \%5, \pm 10\%\}$, and error intervals $[a, b]$, where $a = \{-5\%, -10\%, -15\%, -20\%\}$ and $b = \{5\%, 10\%, 15\%, 20\%\}$, for the extrapolation accuracy at the evaluation point P_+. For each value of m, we generated a set of 100,000 artificial performance functions to evaluate the measurement point selection strategies proposed in this paper. We then created a set of artificial measurements for each function using different sets of parameter value series to simulate various modeling problems and application parameters. Parameter value series that we used for measurement creation are, for example, $\mathbf{x}_1 = \{32, 64, 128, 256, 512\}$, $\mathbf{x}_2 = \{2, 4, 6, 8, 10\}$, or $\mathbf{x}_3 = \{1\,000, 2\,000, 3\,000, 4\,000, 5\,000\}$. Using these parameter-value series, we generate a search space of measurement points $\mathbf{S} = \mathbf{P}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_i)$ and create ten synthetic repetitions $r = 10$ for each point by inducing random amounts of artificial noise within a predefined range, e.g., $n = \pm 1\%$. Since one can not assume that the noise present in an HPC system is exclusively normally distributed, we use a mixed distribution, a combination of Gaussian, uniform, exponential, and Poisson noise components, to create a realistic noise level for our synthetic measurements.

To evaluate the accuracy of a generated performance model, we assess its extrapolation accuracy at an evaluation point P_+, a measurement point that has not been used for creating the performance model. To create the evaluation point P_+, we extended each of the above parameter-value series by adding one parameter value, indicated by the subscript plus sign, e.g., $x_{1_+} = 1\,024$, $x_{2_+} = 12$, and $x_{3_+} = 6\,000$, that is only used for the evaluation of the created performance models, e.g., resulting the in the evaluation point $P_+(x_{1_+}, x_{2_+}, x_{3_+}) = P_+(1\,024, 12, 6\,000)$. We calculate the extrapolation accuracy by comparing the predicted runtime value of the created model for the evaluation point with the generated ground truth value of the synthetic performance function. We then calculate the percentage error and check whether this error lies within a specific error interval $[a, b]$, e.g., $[-5, 5] = \pm 5\%$ at P_+. If this is the case, we count the model as accurate. The evaluation figures show the percentage of accurate models within the given error interval $[a, b]$ for P_+. Consider that this definition of model accuracy differs from the one used as a selection criterion for the measurement point selection by the CPF and GPR strategy (see Section III-A). Unseen points can be used to evaluate the extrapolation accuracy of the models, but not for selecting measurement points.

For the model cost, we use the definition previously introduced in Section III-A. Briefly, we define the cost of a measurement point as the number of core hours needed to conduct the runtime measurement for a given configuration. To compare the modeling cost of different measurement point selection strategies, we assess the model cost as the percentage of the total core hours required to measure all points in the search space—the full matrix of measurement points shown in Figure 1—with each point repeated five times.

To ensure the reproducibility of our results, we provide the software artifacts used for the evaluation, including Extra-P[1] and pyCubexR[2] in Zenodo. Additionally, we have made an extended evaluation available in a GitHub repository[3] including the results and figures from our experiments and detailed instructions for recreating them.

B. Evaluated measurement point selection strategies

Below, we briefly introduce the selection strategies used in our synthetic evaluation that have not yet been described. The **hybrid strategy** combines the CPF and GPR-based approaches to mitigate the GPR method's need for sufficient initial training data. It begins with CPF to collect a baseline set of points, then switches to GPR for improved selection. The switching point—dependent on factors like search space size—must be determined by an expert. Beyond this transition, both strategies are used as previously described. The **random strategy** performs a random search, selecting one point per iteration and taking a single measurement. If a point is selected again, repetitions are added sequentially until the maximum is reached. Unlike CPF, it allows individual repetition selection. The **grid strategy** performs a classical grid search by evaluating all possible measurement point combinations within the budget to find the most accurate model. Like the random strategy, it allows individual selection of point repetitions. The **Bayesian strategy** uses Bayesian optimization with a GP surrogate, identical to that of the GPR strategy,

[1]Extra-P: https://doi.org/10.5281/zenodo.10086772
[2]pyCubexR: https://doi.org/10.5281/zenodo.10092353
[3]Extended evaluation: https://github.com/extra-p/extrap-gpr

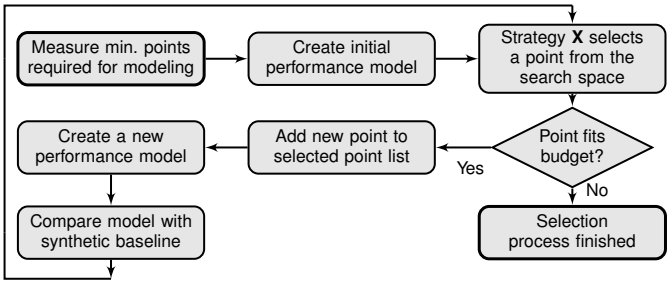

Fig. 4. The core measurement point selection procedure used by each strategy in our synthetic evaluation (excluding the full-matrix approach). Step three is a modular component that varies by strategy.

to model prediction accuracy across the search space. We apply *expected improvement* (EI) as the acquisition function to balance exploration and exploitation. Based on EI, the next sampling point is chosen and added if the budget allows it, and the GP model is updated accordingly.

All selection strategies, except the full-matrix approach, start from a shared baseline of measurement points: 9 for two-, 13 for three-, and 17 for four-parameter models, based on prior work [6] identifying these as cost-effective for initial modeling with Extra-P. From this baseline, each strategy iteratively selects additional points using a unified procedure (Figure 4). A model is first built from the baseline. Each strategy then proposes a new point based on current measurements and remaining candidates. If the point fits the budget, it is measured and added; otherwise, the evaluation ends. The model is updated after each new point, and this loop continues until the budget is depleted.

C. Synthetic evaluation results

Figure 5 compares the achieved model accuracy of the evaluated measurement point selection strategies. For two-parameter models (top row), the GPR strategy outperforms the CPF, random, and grid strategies at low and high noise levels, significantly improving model accuracy by up to **30.36%** compared to CPF, while using only 11% of the available modeling budget. We see similar results for $n = \pm 2\%$ with a maximum improvement in model accuracy of **30.17%** compared to CPF for $B = 11\%$. Thus, for low noise levels $n = \pm 1\%$ and $n = \pm 2\%$, the GPR strategy achieves almost the same accuracy as the full matrix of points, requiring only a tenth ($\approx 10\%$) of the whole modeling budget. The CPF strategy needs twice the budget ($\approx 20\%$) to achieve the same result. Furthermore, the results of the modeler using the whole matrix of points compared to the CPF and GPR strategy in Figure 5 indicate that for higher noise levels $n \geq \pm 5\%$ more points, i.e., a larger modeling budget, are less beneficial. Moreover, a full matrix of points is unnecessary and can sometimes lead to less accurate models. Where the modeler using the whole matrix of measurement points reaches only an accuracy of **66.01%** for $n = \pm 5\%$, the CPF and GPR strategy achieve **83.02%** ($B = 51\%$) and **84.09%** ($B = 41\%$), respectively. For higher noise levels, this effect is even more pronounced. Consequently, we conclude that as we collect more data, especially if these data points are noisy, i.e., not representative of the actual underlying performance behavior, the overall noise level of the training dataset will increase. Thus, it becomes harder for the model to learn meaningful patterns, degrading its extrapolation accuracy.

Another problem is overfitting on data subsets. The more data we obtain, particularly if this data includes subgroups or outliers, likely due to system noise, our model can become overly complex and capture patterns specific to those subgroups rather than generalizable trends. However, as the results in Figure 5 show, these problems are much less of an issue when using our new GPR strategy. Using only 10% of the available modeling budget, the GPR approach performs **30.58%** better than the CPF strategy for $n = \pm 5\%$ and **27.6%** better for $n = \pm 10\%$.

The grid sampling strategy consistently exhibits the weakest performance in our experiments, falling short of the CPF strategy across the entire budget range and for all noise levels when modeling two parameters. While the random strategy comes close to CPF for two parameters and lower noise levels, it is inferior for three parameters and higher noise levels and becomes erratic for four parameters. In contrast, the hybrid and Bayesian strategies performed comparably to the GPR approach under low-noise conditions but never surpassed their performance. Under higher noise levels—particularly at $n = \pm 10\%$—the Bayesian strategy briefly outperforms the GPR approach within narrow budget intervals ($B = 1$–2% and $B = 10$–11%). However, its performance later declines, sometimes falling below the random strategy's, whereas the GPR strategy remains more consistent and reliable across noise levels and budget constraints.

Note that the drop in the percentage of accurate models does not contradict the termination criterion *Model accuracy improved?* of the measurement-point selection process in Figures 2 and 3. While a suggested point might improve the model's accuracy on the training dataset, its influence on the model's extrapolation accuracy is unknown.

For three-parameter models (middle row), we see similar results. The GPR strategy again performs significantly better than the CPF strategy over the entire modeling budget range B, independent of the amount of artificially induced noise. For low noise levels, the GPR strategy needs only a twentieth of the whole modeling budget to achieve the same accuracy as the modeler using the full matrix of points. Even at high noise levels, the GPR strategy performs up to **26.3%** better for $n = \pm 5\%$ and **21.5%** better for $n = \pm 10\%$ than CPF.

While the Bayesian strategy sometimes performs better and sometimes worse than the CPF strategy, the hybrid approach achieves a predictive accuracy that closely approaches the GPR method. As observed throughout the evaluation, the grid search strategy performs significantly worse than all other methods. The results of the random strategy are more nuanced. It clearly underperforms compared to GPR at lower budgets, although it comes close to GPR and sometimes even beats it in higher budget ranges where all strategies naturally converge. Since this benefit only materializes at relatively large modeling budgets—often impractical in real-world applications—its overall value remains limited.

For four-parameter models (bottom row), we require an even

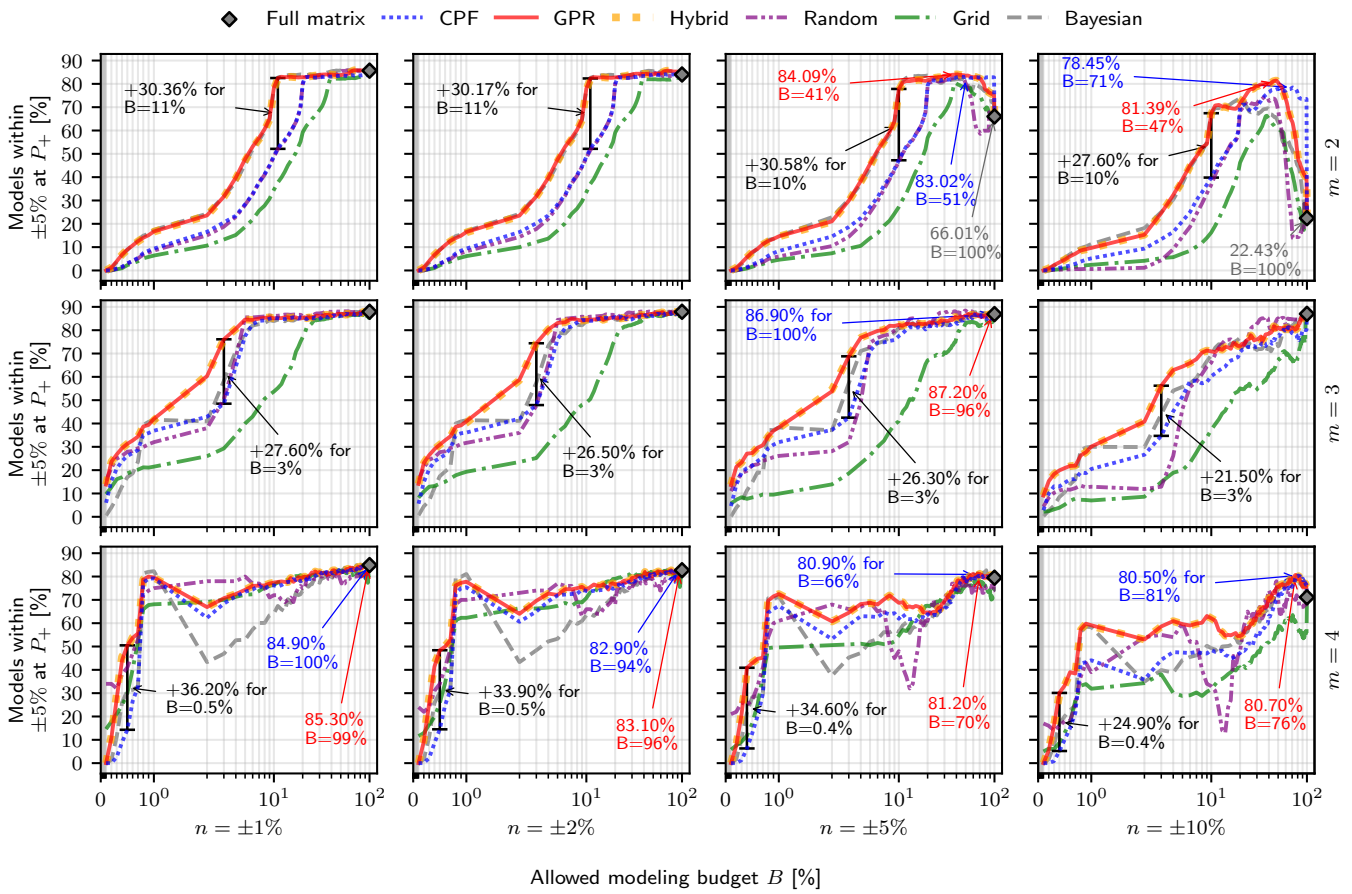

Fig. 5. Comparison of the achieved mean model accuracy on the synthetic evaluation dataset by the GPR, CPF, hybrid, random, grid, Bayesian, and full matrix measurement point selection strategies, considering different amounts of artificially induced noise $n = \{\pm 1, \pm 2, \pm 5, \pm 10\}\%$ and number of model parameters $m = \{2, 3, 4\}$. The x-axis represents the modeling budget B in percent, compared with the cost of the full matrix that may be used for modeling by each strategy. The figure uses a logarithmic scale to highlight better the differences in the lower budget range (0% to 10%), where they are particularly pronounced. The y-axis outlines the percentage of models whose prediction lies within $\pm 5\%$ of the actual measured runtime at the evaluation point P_+. The annotations outline the maximum improvement in model accuracy between the CPF and GPR techniques, including the highest model accuracy achieved by the selection strategies. In those subfigures that do not specify the highest model accuracy, all strategies achieved the same outcome. Each data point represents the deterministic output of a single Extra-P modeling experiment based on the conducted measurements; repeated runs with the same input yield identical results, so no variance bars are shown.

smaller percentage of the entire modeling budget to create accurate models. For $n = \pm 1\%$ and $n = \pm 2\%$, GPR performs exceptionally well for tiny modeling budgets. Using only 0.5% of the available modeling budget, the model accuracy of the GPR strategy is **36.2%** higher than that of the CPF strategy for $n = \pm 1\%$ and **33.9%** higher for $n = \pm 2\%$. Additionally, as outlined in Figure 5, the GPR strategy also achieves a higher maximum model accuracy, e.g., **85.3%** (GPR) compared with **84.9%** (CPF) for $n = \pm 1\%$. Furthermore, using only one-hundredth of the budget, 80% of the created models lie within $\pm 5\%$ at P_+ for up to $\pm 5\%$ of noise. For a noise level of $\pm 10\%$, we again observe that using a full matrix of measurement points leads to lower model accuracy. While **80.7%** of the created models lie within $\pm 5\%$ at P_+ using the GPR strategy, we achieve only **71.1%** using the full matrix of points. GPR generally shows greater improvements with four model parameters as noise levels increase.

The grid search strategy consistently performs worse than the GPR strategy across all noise levels. This pattern is also evident for the Bayesian approach, whose performance appears to decline as the dimensionality of the search space increases. The random strategy again exhibits some noteworthy behavior for four model parameters. It achieves the highest prediction accuracy among all strategies within the minimal budget range ($B = 0.1$–0.3%) and remains competitive at lower noise levels ($n = 1$–5%), but falls behind for higher ones. Moreover, it exhibits occasional spikes in prediction accuracy at large budgets, but its overall performance remains highly erratic. Finally, the hybrid strategy demonstrates robust performance in the four-parameter case, achieving accuracy comparable to the GPR strategy across nearly all conditions.

Figure 6 compares the model's extrapolation accuracy considering three model parameters for different error intervals $[a, b] = \{\pm 5\%, \pm 10\%, \pm 15\%, \pm 20\%\}$. As for Figure 5, we outlined the maximum improvement in overall model accuracy of the GPR strategy over the CPF technique. The results further highlight that the GPR strategy outperforms the CPF strategy independent of the prediction interval, e.g., by up to

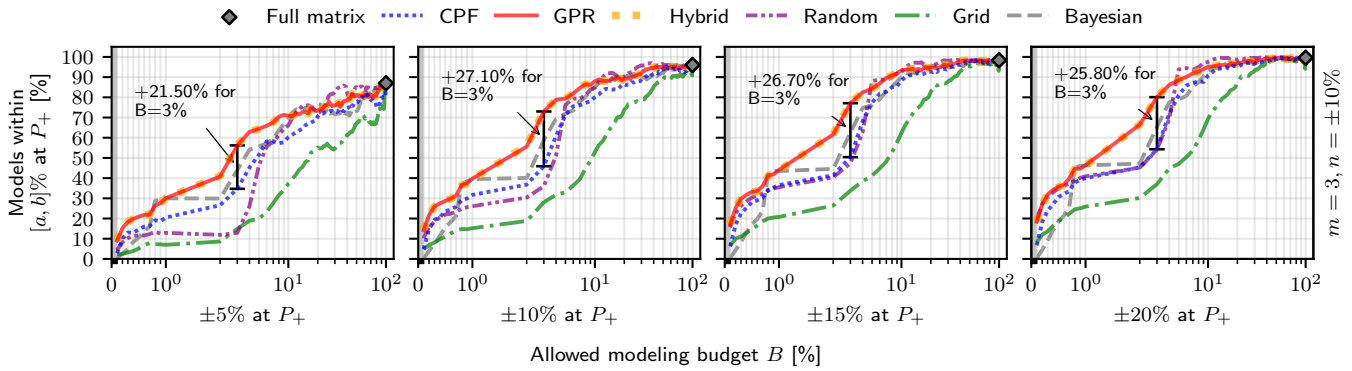

Fig. 6. Comparison of the achieved mean model accuracy on the synthetic evaluation dataset by the GPR, CPF, hybrid, random, grid, Bayesian, and full matrix selection strategies for $n = \pm 10\%$ and $m = 3$. The x-axis represents the modeling budget in percent, compared with the cost of the full matrix, that each strategy may use for modeling. The y-axis outlines the percentage of models whose prediction lies within the interval $[a, b]$, e.g., $[-5, 5] = \pm 5\%$ of the actual measured runtime at P_+. The annotations outline the maximum improvement in model accuracy between the CPF and GPR techniques. Each data point represents the deterministic output of a single Extra-P modeling experiment; repeated runs yield identical results.

21.5% for $\pm 5\%$ at P_+. Furthermore, using only a tenth of the modeling budget **95%** of the created models lie within $\pm 20\%$ at P_+. A performance prediction with a maximum error of $\pm 20\%$ for an application configuration far from the training ones is still very accurate, especially considering $\pm 10\%$ noise.

We observe trends consistent with those shown in Figure 5 for the remaining evaluated selection strategies. The grid search strategy consistently yields the lowest performance across all budget levels. The Bayesian optimization approach outperforms CPF but exhibits notable weaknesses in the low-budget regime. The random strategy again displays highly inconsistent behavior: at lower budget levels, its performance is generally inferior to that of the CPF strategy, while at higher budgets, it occasionally outperforms even the GPR approach through isolated spikes in accuracy. The hybrid strategy performs nearly identically to the GPR approach across the entire budget spectrum.

Generally, a percentage of 80% of the models lying within $\pm 5\%$ at the evaluation point is very accurate. For noise levels of $n = \pm 10\%$, even 60% within $\pm 5\%$ at P_+ are very high. One needs to consider that, in this case, the extrapolation error of the models is still smaller than the influence of system noise on the measurements. Furthermore, when considering several model parameters, not all influence the application's performance behavior equally. Therefore, one of the parameters' influences on performance is likely smaller than the influence of system noise, making it impossible to distinguish its impact on performance from noise. When deciding on the modeling budget, one should follow the Pareto principle, as explained at the end of Section III-A, and opt for a budget that enables the GPR strategy and the modeler to reach these 80%, as spending more delivers only significantly diminishing returns.

D. Discussion

In our evaluation, we observed that the GPR strategy achieves, overall, the highest model accuracy using the allowed modeling budget, independent of the number of model parameters, noise level, or the evaluation intervals. The improved extrapolation accuracy of the models compared with the CPF strategy confirms that our weighted cost function $C_w(P(x_i))$ for calculating measurement point cost and the selected weights were chosen appropriately to identify a near-Pareto-optimal solution that fits the modeling problem at hand very well. Other critical contributions to achieving this extrapolation accuracy are the weight function, enabling the GPR approach to reason between new measurement points and additional repetitions, and the modeling of system noise using a white kernel An interesting result regarding measurement point repetitions was that more than five repetitions did not significantly improve results at high noise levels.

The hybrid strategy performs comparably to the GPR strategy, although it never surpasses it. Its effectiveness is highly sensitive to the choice of the switching point at which the method transitions between the CPF and GPR strategies. This parameter requires domain expertise to configure, which may not always be available in practice. As a result, the hybrid strategy is less robust than the GPR approach. We conducted preliminary test runs for our evaluation to determine an optimal switching point for the hybrid strategy. Accordingly, the results presented in Figure 5 represent the upper-bound performance of the hybrid approach. In realistic scenarios without prior tuning or expert knowledge, its performance tends to be significantly lower than that of the GPR strategy.

As expected, the grid search strategy consistently yields the lowest performance across all experiments. Despite being a learning-based strategy and utilizing the same Gaussian process prior as our GPR strategy, the Bayesian optimization approach performs significantly worse overall than the GPR approach. However, in certain edge cases—such as the two-parameter experiments with high noise levels ($n = \pm 10\%$) and within the budget range of 1–2%—the Bayesian method slightly outperforms GPR. These isolated instances suggest that the trade-off between exploration and exploitation governed by the EI acquisition function employed in our Bayesian setup can be advantageous in specific scenarios.

The random strategy is inferior to GPR for two and three parameters, but comes at least close to CPF in some scenarios. At

four parameters, its performance is quite unreliable, sometimes reducing the accuracy as more points are added. The unexpectedly strong results in specific budget ranges—such as $B = 1\%$–5% for $n = \pm 1\%$—suggest that increased search space exploration can be beneficial, particularly in high-dimensional settings. This observation is further supported by the behavior of the Bayesian approach, where the EI acquisition function occasionally yields similar benefits. Despite these findings, the random strategy's inherently stochastic and unpredictable nature renders it unsuitable for practical measurement point selection. However, incorporating an initial phase of random sampling at very low budgets and a similar exploratory phase at high budget ranges, followed by a more structured strategy in intermediate stages, may offer a promising direction. Such a hybrid approach—adaptively switching selection paradigms based on the stage of the selection process—could effectively leverage the advantages observed in our experimental results. Nevertheless, it is important to note that in practical scenarios, particularly for use cases involving four model parameters, it is often not feasible to allocate more than 10% of the total measurement budget. This constraint limits the applicability of strategies that rely on performance gains at high budget levels and underscores the need for methods that deliver strong results within tight measurement constraints.

V. Application Case Studies

In this section, we present six application case studies demonstrating the overall cost reduction and improvement in model accuracy that the new GPR approach can achieve compared to the CPF strategy. We show that the new method eliminates previous limitations, for example, by including automatic noise detection, and extends the capabilities of Extra-P by suggesting whether to repeat a specific measurement configuration and an even lower minimum budget to create performance models. In our previous work [6], [7], we analyzed three application benchmarks (FASTEST, Kripke, and RELeARN). We showcased that we can accurately predict selected application kernels' runtime and other performance metrics. In contrast, in this paper, we provide a much more comprehensive analysis of all performance-relevant kernels of each application. We designate an application kernel as performance-relevant if its runtime exceeds at least one percent of the total application runtime at a specific evaluation point. Moreover, we are expanding our analysis to include three additional benchmarks, LULESH, MiniFE, and Quicksilver, to assess our approach across different applications, evaluation systems, noise levels, and model parameters.

A. Evaluation methodology & reproducibility

To further evaluate the model accuracy, modeling cost, and budget usage of our new GPR approach with the current CPF approach, we follow the same methodology we used for the synthetic evaluation outlined in Section IV-A. However, we use actual performance measurements from experiments with the benchmark codes this time. We use Score-P [12] to conduct the measurements for all case studies, which enables us to automatically instrument all kernels of an application and measure their runtime. As our synthetic analysis has shown that the GPR performed best from all the selection strategies, for the evaluation of the case studies, we limit ourselves to a more comprehensive analysis of the GPR strategy's performance compared to the CPF method.

For each case study, we create multi-parameter performance models for all instrumented application kernels and a range of modeling budgets using our novel GPR approach, the CPF approach, and the full matrix of measurement points. The range of the modeling budget depends on the particular case study. The minimum budget is determined by the cost of the minimal set of measurement points required by Extra-P to create initial models. The combined cost of all measurements conducted determines the maximum budget. This search space is sufficiently large to showcase the capabilities of the GPR strategy without spending a fortune on performance measurements. To evaluate the accuracy of the created models, we measure an additional application configuration, i.e., a measurement point not used for model creation. This evaluation point, which has a larger scale than the ones used for modeling, is then used to evaluate the performance model's predictive power. We do this by comparing the actual measured runtime with the model's prediction for the evaluation point. We then calculate the percentage error and repeat this process for every instrumented application kernel. Additionally, we filter the application kernels based on their percentage of the total application runtime at the evaluation point, considering only kernels with a contribution of $> 1\%$ in our analysis. This is because small and short-running kernels are over-proportionally affected by system noise and, thus, tend to have high runtime variation, making them difficult to model while not being relevant for overall application performance. Finally, we analyze how many created models lie within the distinct error intervals $\{\pm 5\%, \pm 10\%, \pm 15\%, \pm 20\%\}$.

Moreover, we repeat each performance experiment up to five times to study the impact of system noise on the measurements and the performance models. For this purpose, we provide a detailed analysis of the noise levels found in the measurements of each case study in Section V-E and discuss how these impact the GPR strategy's point selection.

To enable reproducibility of the presented results, we provide detailed instructions for recreating the results in our extended evaluation GitHub repository[3] along with the performance measurements of the benchmarks in Zenodo[4]. For a more in-depth analysis, please refer to the extended evaluation in the same repository.

B. Benchmarks & experimental setups

For our evaluation, we selected six HPC applications, some commonly used for benchmarking: FASTEST, Kripke, LULESH, MiniFE, Quicksilver, and RELeARN. In the following, we briefly introduce each of them and discuss the configurations used to conduct the performance measurements, which are shown in Table II. The configurations were selected based on our knowledge of the applications, evaluation systems, and modeling budgets that were available to us.

[4]Performance measurements: https://doi.org/10.5281/zenodo.10085298

TABLE II
APPLICATION CONFIGURATIONS USED TO CONDUCT THE CASE STUDIES' PERFORMANCE EXPERIMENTS, INCLUDING THE NUMBER OF REPETITIONS PER CONFIGURATION. A SUBSCRIPT PLUS SIGN, E.G., p_+, INDICATES PARAMETER VALUES USED FOR THE EVALUATION POINTS P_+.

App/System	Experiment configuration	Rep.
FASTEST, SuperMUC	$p = \{16, 32, 64, 128, 256, 512, 1\,024, 2\,048\}$, $s = \{131\,072, 65\,536, 32\,768, 16\,384, 8\,192\}$, $p_+ = \{4\,096\}$, $s_+ = \{8\,192\}$	r=5
Kripke, Vulcan	$p = \{8, 64, 512, 4\,096, 32\,768\}$, $d = \{2, 4, 6, 8, 10\}$, $g = \{32, 64, 96, 128, 160\}$, $p_+ = \{32\,768\}$, $d_+ = \{12\}$, $g_+ = \{160\}$	r=5
LULESH, Lichtenberg	$p = \{125, 216, 343, 512, 729\}$, $s = \{10, 15, 20, 25, 30\}$, $p_+ = \{1\,000\}$, $s_+ = \{35\}$	r=5
MiniFE, Lichtenberg	$p = \{64, 128, 256, 512, 1\,024\}$, $s = \{75, 80, 85, 90, 95\}$, $p_+ = \{2\,048\}$, $s_+ = \{100\}$	r=5
Quicksilver, Lichtenberg	$p = \{16, 32, 64, 128, 356\}$, $m = \{2, 4, 8, 12, 16\}$, $s = \{10, 20, 30, 40, 50\}$, $p_+ = \{512\}$, $m_+ = \{20\}$, $s_+ = \{60\}$	r=5
RELeARN, Lichtenberg	$p = \{32, 64, 128, 256, 512\}$, $s = \{5\,000, 6\,000, 7\,000, 8\,000, 9\,000\}$, $p_+ = \{1\,024\}$, $s_+ = \{10\,000\}$	r=2

The software package FASTEST [13] is a tool for simulating flows, e.g., turbulent flows around a cylinder or mixing simulations in complex three-dimensional configurations. For our analysis, we apply strong scaling and consider two of its configuration parameters: the number of processes p and the problem size per process s. We measured 25 application configurations for modeling using the parameter-value combinations for p, s shown in Table II and repeated each measurement five times. For example, with the parameter values $p = 2\,048$ and $s = 8\,192$ we get the measurement point $P(p, s) = P(2\,048, 8\,192)$. Furthermore, we measured one additional configuration $P_+(p_+, s_+) = P_+(4\,096, 8\,192)$, which is only used for evaluation. The unique properties of FASTEST limit how parameter values can be combined. For instance, when $p = 16$, the problem size per process must be $s = 131\,072$. However, for $p = 32$, we can select s from the set $\{131\,072, 65\,536\}$. Finally, for $p = 2\,048$, only $s = 8\,192$ is valid. As a result, the measurement point matrix takes the shape of a parallelogram.

Kripke [14] is an open-source code and research tool that was created to explore how different data layouts, programming paradigms, and architectures affect the implementation and performance of Sn transport [14]. In computational nuclear physics and radiation transport, the Sn transport method is a numerical technique used to solve the Boltzmann transport equation, which describes the behavior of particles, e.g., neutrons, as they travel through and interact with a medium. For our analysis of Kripke, we consider three configuration parameters: the number of processes p, the number of direction sets d, and the number of energy groups g. Using the parameter value combinations shown in Table II, we measured 150 application configurations with five repetitions each (a total of 750 performance experiments). To evaluate the accuracy of the created models, we use the measurement point $P_+(p_+, d_+, g_+) = P_+(32\,768, 12, 160)$.

LULESH (Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics) is a proxy app created by Lawrence Livermore National Laboratory to solve a simple Sedov blast problem with analytic answers—but represents the numerical algorithms, data motion, and programming style typical in scientific C or C++ based applications [8]. Simulations of a wide variety of science and engineering problems require modeling hydrodynamics. Thus, it is a frequently used benchmark in the field of HPC. We use LULESH 2.0 with MPI support for our analysis, considering two parameters: the number of processes p and the problem size s. Using the parameter values outlined in Table II, we measured 25 configurations for modeling and one point, $P_+(p_+, s_+) = P_+(1\,000, 35)$, for evaluation. We repeat the measurements for each configuration five times.

MiniFE, also known as HPCCG, is a mini-app that mimics the finite element generation, assembly, and solution for an unstructured grid problem, which is required by many engineering applications [15], [16]. It is not intended to be an actual physics problem but sufficiently realistic for performance analysis, for example, for studying the scalability of competing systems. MiniFE contains approximately 1 500 lines of C++ code and supports a variety of APIs such as OpenMP, MPI, and CUDA. We use its MPI version for our analysis, considering two parameters: the number of processes p and the problem size s. As before, we use the parameter-value combinations in Table II to measure 25 different configurations for modeling and one point, $P_+(p_+, s_+) = P_+(2\,048, 100)$, for evaluation. Finally, we repeat each of the measurements five times.

Quicksilver is an open-source proxy app that represents the key elements of the Mercury [17] workload by solving a simplified dynamic Monte Carlo particle transport problem [18]. Simply put, Monte Carlo transport simulations sample the probabilities of the various interactions that a particle can have with its surroundings. Quicksilver attempts to replicate the memory access patterns, communication patterns, and the branching or divergence of Mercury for problems using multigroup cross-sections. We use its MPI version for our analysis, considering three configuration parameters: the number of processes or MPI ranks p, the number of mesh elements m, and the number of particles s. We measured 125 different configurations for modeling and one additional for evaluation, repeating each five times. We use the point $P_+(p_+, m_+, s_+) = P_+(512, 20, 60)$ to evaluate the accuracy of the created performance models.

RELeARN simulates the rewiring of connections between neurons in the brain based on the Model of Structural Plasticity (MSP) by Butz and van Ooyen [19] and employs a scalable approximation algorithm [20] with a runtime complexity of $\mathcal{O}(s/p \cdot \log(s) + p)$ [21]. RELeARN, which is parallelized using MPI, has three configuration parameters: the number of processes p, the problem size s, and Θ, which determines the degree of approximation used by the underlying Barnes-Hut algorithm [22]. For our analysis, we consider only two parameters, p, s, and set $\Theta = 0.1$. Modeling Θ is impossible because it influences the computational complexity. We measured its performance for 25 different configurations for modeling and one additional configuration for evaluation with, lacking sufficient computing time, only two repetitions each. We varied p and s to obtain these configurations, as shown

TABLE III
EVALUATION SYSTEMS AND THEIR HARDWARE CONFIGURATIONS.

Name	System hardware
Lichtenberg (MPI section)	630 nodes, 2x Intel Xeon Platinum (Cascade Lake) 9242 CPU (48 cores @2.3 GHz, no hyperthreading), 384 GB RAM (DDR4-2933), InfiniBand HDR100 (100 GBit/s)
Vulcan	24 576 nodes, 1x IBM PowerPC A2 CPU (16 cores @1.6 GHz), 16 GByte DDR3 RAM, 5D Torus interconnect
SuperMUC	3 072 nodes, 2x Intel Xeon Processor E5-2697 v3 (Haswell) CPUs (14 cores @2.6 GHz, 28 threads), 64 GByte DDR3 RAM, Infiniband FDR14

in Table II. We use the measurement point configuration $P_+(p_+, s_+) = P_+(1\,024, 10\,000)$ for evaluation.

C. Evaluation systems

To study the impact of different system architectures and noise levels on the accuracy of our performance models, we ran performance experiments on three different HPC systems: the Lichtenberg cluster at TU Darmstadt, the Vulcan supercomputer at Lawrence Livermore National Laboratory, and SuperMUC at Leibniz Supercomputing Centre. Each of these systems features different hardware and software configurations. Vulcan and SuperMUC are both IBM systems with different CPUs (IBM PowerPC vs. Intel), operating systems, and network architectures. On the other hand, Lichtenberg is a Linux-based system with Intel CPUs, a widespread combination. Table III outlines the hardware configuration of each system in detail. As shown in Table II, we ran FASTEST on SuperMUC, Kripke on Vulcan, and the remaining benchmarks, LULESH, MiniFE, Quicksilver, and RELeARN, on Lichtenberg. We conducted the performance experiments under the systems' normal operating conditions, ensuring that the recorded application behavior and system noise reflect realistic values under standard load.

D. Evaluation results

Figure 7 presents the results of our case studies. For each case study, it shows the achieved extrapolation accuracy, i.e., the percentage of models whose prediction lies within $\pm 10\%$ at P_+ and the budget usage of the CPF and GPR strategy. The x-axes of the plots represent the modeling budget B in percent, compared to the cost of the full matrix of measurement points that may be used for modeling by each strategy. Additionally, we display the mean minimal modeling budget $\bar{B}_{min}$, i.e., the cost of the measurement points required by Extra-P to create initial models, the number of kernels that were modeled (i.e., those that consume more than one percent of the total application runtime), the mean noise level in percent, and the number of measurement repetitions for each case study.

1) FASTEST: We modeled a total of 18 application kernels. The minimum required modeling budget for the most expensive application kernel is $\bar{B}_{min} = 19\%$. Therefore, the x-axes for the FASTEST plots in Figure 7 range from 19% to 100%, where 100% equals the cost of the full matrix of measurement points (all configurations and their five repetitions). The results of the left plot in Figure 7 show that the GPR strategy significantly outperforms the CPF strategy regarding the achieved model accuracy. Starting from $\bar{B}_{min} = 19\%$, the CPF strategy is not able to produce any accurate models, as none of the predictions lies within $\pm 10\%$ of the actual measured runtime at P_+. Using the GPR strategy, 11.1% of the created models lie within the $\pm 10\%$ accuracy interval using only the minimum required modeling budget. In contrast, the CPF strategy needs at least 40% of the total modeling budget to achieve these results. Furthermore, using only **65%** of the modeling budget, the GPR strategy achieves its highest model accuracy of **50.0%**. The CPF strategy requires a modeling budget of 85% to reach this accuracy level. The critical observation of this experiment, however, is that from this point on, a higher modeling budget does not automatically yield higher model accuracy. On the contrary, more measurement points can lead to an overall lower model accuracy, as shown by the decrease in accuracy of all strategies past $B = 87\%$. Consequently, the achieved model accuracy using all measured points, which is 44.4%, outlined by the gray diamond, is smaller than the one reached by the GPR strategy with only 65% of the budget.

For a minimal budget fraction of $\bar{B}_{\text{min}} = 19\%$, the GPR approach utilizes, on average, **33.67** fewer points than the CPF strategy. This advantage persists up to a budget threshold of $B = 30\%$. Between $B = 30\%$ and $B = 40\%$, both strategies converge regarding the number of points selected. Beyond $B > 40\%$, the trend reverses: the GPR strategy begins selecting up to **13.06** more points, which are comparatively less expensive, while the CPF strategy opts for fewer, higher-cost points. When coupled with the improved predictive accuracy of the resulting models, these findings indicate that the GPR strategy is more effective at identifying informative and cost-efficient application configurations for measurement.

2) Kripke: We analyzed three application parameters, resulting in a three-dimensional matrix of measurement points and a much larger number of potential points that can be selected. Consequently, choosing the right points to improve model accuracy while minimizing model cost becomes more challenging. Using only a minimum budget of $\bar{B}_{min} = 2\%$ and any point selection strategy, 50% of the created performance models lie within $\pm 10\%$ at P_+. Starting from $B = 10\%$, we see that the GPR strategy achieves a model accuracy that is about 5% to 10% higher than that of the CPF strategy. With an increasing modeling budget, the difference between the CPF and GPR strategy becomes more pronounced, reaching a maximum difference of **14.29%** for $B = 86\%$. As for FASTEST, the GPR strategy reaches the highest achieved model accuracy of 85.71% with only a fraction of the total modeling budget, namely $B = 38\%$. Using more points for modeling does not lead to any further improvement in model accuracy. The maximum achieved accuracy is not separately outlined as all strategies reach the same maximum of 85.71% at $B = 100\%$. For budget values between $B = 2\%$ and $B = 30\%$, the GPR method uses consistently fewer points, up to **32.93** points less for $B = 19\%$, for modeling. For larger budgets, however, we cannot observe any differences.

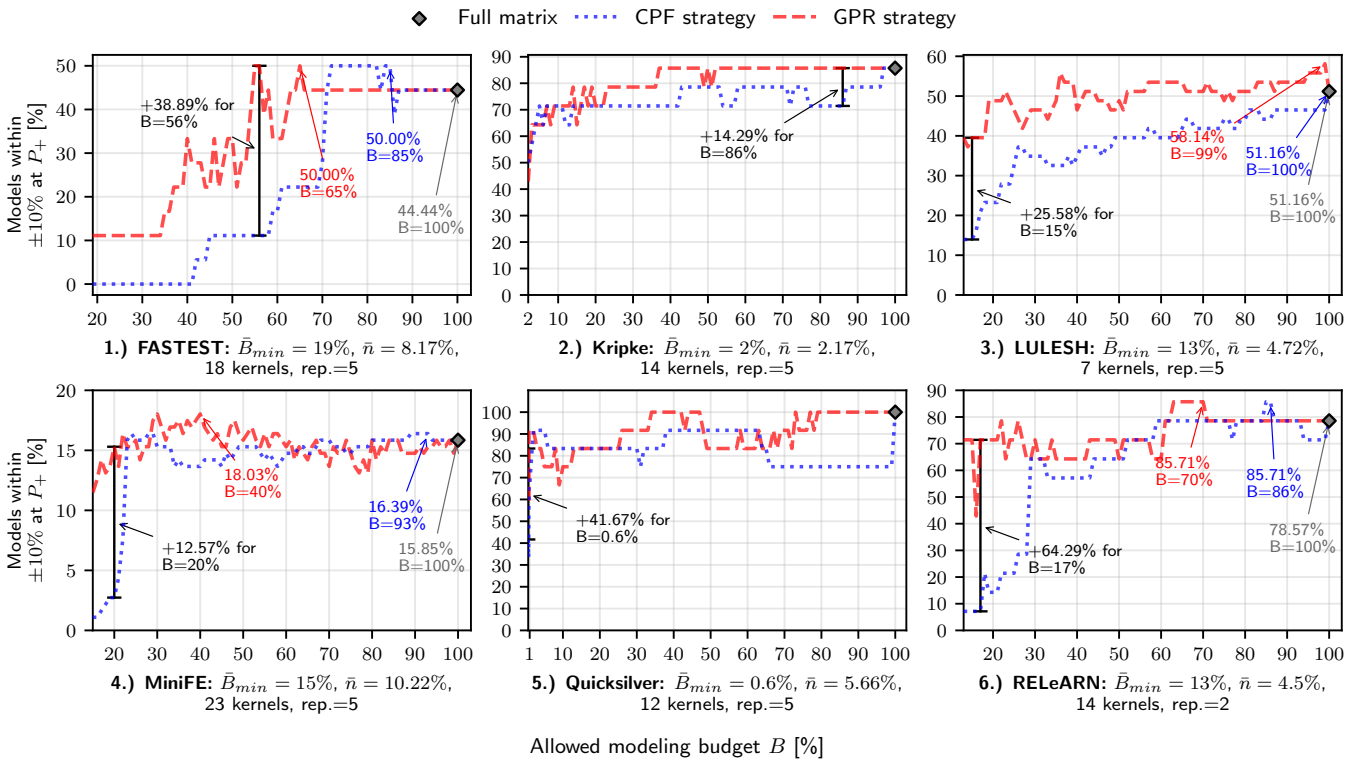

Fig. 7. Achieved model accuracy and budget usage used for modeling using the full matrix of measurement points, the CPF, and GPR strategies for each case study. The x-axis represents the modeling budget in percent B, compared to the cost of the full matrix, that may be used for modeling by each strategy. The legend outlines the mean minimum budget $\bar{B}_{min}$ in percent required to create performance models, the measurements mean noise level $\bar{n}$, the number of modeled kernels with $> 1\%$ runtime of the total application runtime, and the number of repetitions. The annotations outline the maximum improvement in model accuracy between the CPF and GPR techniques and their highest achieved model accuracy. Each data point represents the deterministic output of a single Extra-P modeling experiment; repeated runs yield identical results.

3) LULESH: We observe a significant improvement in overall model accuracy when using the GPR strategy for measurement point selection. Starting from the minimum required budget $\bar{B}_{min} = 13\%$, the GPR strategy significantly outperforms our old approach over the entire budget range. For $\bar{B} = 15\%$, 39.48% of the models created using the GPR strategy are within $\pm 10\%$ at P_+ compared to only 13.9% when using the CPF strategy, a difference of **25.58%**. Like in the other case studies, the GPR strategy achieves the highest model accuracy. It reaches 58.14% for $B = 99\%$, though it reaches almost the same accuracy using only 36% of the total modeling budget. However, the CPF strategy does not reach the same level of predictive accuracy, attaining a maximum of only 51.16% even when utilizing the entire budget ($B = 100\%$). A similar pattern is observed concerning the number of measurement points utilized for model construction: the GPR strategy consistently requires fewer points than the CPF strategy, using fewer measurement points on average **15.56**.

4) MiniFE: We generally observe a much lower model accuracy than for all other case studies. There are two reasons for this. First, when analyzing the measurements, we found the highest mean noise level $\bar{n} = 10.22\%$ of all case studies, which makes it more challenging to create accurate models. Second, when the problem size is increased, the application runtime does not increase monotonically, which is a base assumption of Extra-P's modeling methodology. Despite the lower model accuracy, we can observe the same trends as in the other case studies. Using only a minimal budget, the GPR strategy performs significantly better than the CPF strategy for all 23 modeled application kernels. Furthermore, it achieves the highest model accuracy of 18.03% using only 40% of the budget. In contrast, the CPF strategy reaches a maximum model accuracy of 16.39%, using $B = 93\%$. Using all available measurements, we even achieved an accuracy of only 15.85%. Especially for low modeling budgets, the GPR strategy significantly outperforms the CPF technique, performing up to **12.57%** better for $B = 20\%$. Finally, the GPR strategy uses significantly fewer points for modeling at low budget levels ($B \leq 35\%$), averaging 19.08 fewer points.

5) Quicksilver: As for Kripke, we considered three application parameters for the analysis, making the search space for the measurement point selection much larger. Despite being a challenging optimization problem, we observe a generally high model accuracy for the 12 measured application kernels. Using only a minimum budget of $\bar{B}_{min} = 0.6\%$, the GPR strategy reaches a mean model accuracy of 83.3% compared to only 41.6% using the CPF strategy, a difference of **41.7%**. Subsequently, the CPF strategy performs slightly better until a budget of $B = 13\%$ is reached. Past this point, the GPR strategy outperforms it by up to 22%. As for the other case

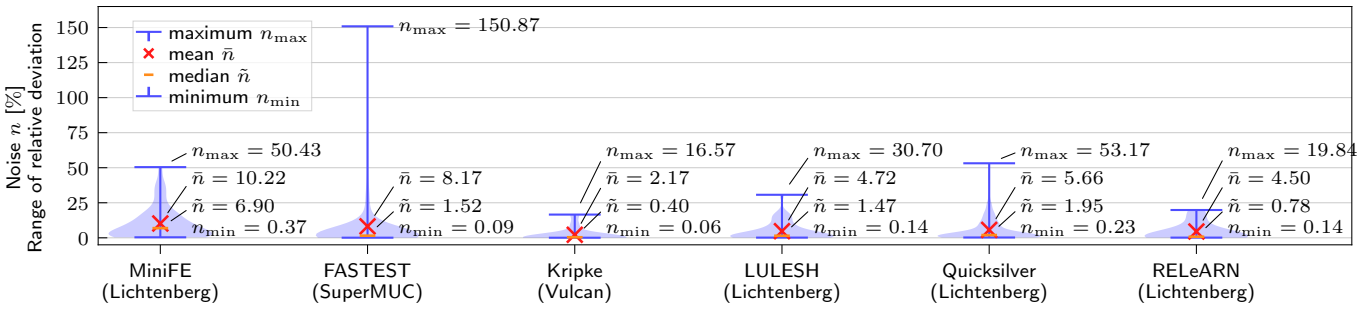

Fig. 8. Noise level n (range of relative deviation from the mean metric value of an application kernel in percent) found in the performance measurements of the application case studies. For each case study, the violin plots mark the mean $\bar{n}$, median $\tilde{n}$, minimum n_{min}, and maximum n_{max} noise found in the measurements and show the distribution of the deviations. The x-axis shows the different case studies and the systems they were conducted on.

studies, the GPR strategy reaches the maximum achieved model accuracy using only a fraction ($B = 34\%$) of the modeling budget. Conversely, the CPF strategy needs the full budget to achieve the same mean model accuracy. With all measurement points, all strategies reach 100% accuracy.

Regarding the number of points used for modeling, we observe only slight differences between both strategies. However, the GPR strategy consistently uses fewer points. For $B = 19\%$, we observed that the GPR strategy used on average **25.08** points less for modeling. Overall, we see less pronounced differences than we did for Kripke. Therefore, we conclude that the larger the number of available points, the less important the number of points used for modeling. Instead, choosing the right points becomes critical.

6) RELeARN: We measured only two repetitions per measurement point for the RELeARN analysis, which makes this case study unique. The trade-off between fewer repetitions and new measurement points holds a different optimization challenge for the selection strategies. Generally, we see a significant improvement in model accuracy for small modeling budgets starting from $\bar{B}_{min} = 13\%$ to $B = 30\%$. While the CPF strategy achieves a model accuracy of only 7.14% using $\bar{B}_{min}$, the GPR strategy achieves a model accuracy of 42.85%, a difference of 35.71%. For $B = 17\%$, the GPR strategy performed **64.29%** better than the CPF strategy. For $B = 70\%$ the GPR strategy reaches a maximum model accuracy of 85.71%. In contrast, the CPF strategy needs 86% of the modeling budget to achieve this result. Using all available measurement points instead results in a lower overall model accuracy of just 78.57%.

Initially, the CPF strategy uses many points for modeling, leading to lower accuracy. At $B = 13\%$, the GPR strategy uses, on average, **19.5** fewer points. At $B = 22\%$, the CPF strategy uses only 3.24 fewer points but yields less accurate models. Therefore, prioritizing unseen measurement points over repetitions improves model accuracy at low noise levels, such as those observed in the RELeARN measurements ($\bar{n} = 4.5\%$). The GPR strategy utilizes the available modeling budget more efficiently by adjusting the number of points to optimize model accuracy for the given problem.

E. The influence of noise on model accuracy

Figure 8 shows the noise level n in percent present in the measurements of all six case studies and the systems they were run on. In addition, it marks the minimum n_{min}, mean $\bar{n}$, median $\tilde{n}$, and maximum n_{max} noise level found for each of them. To obtain the noise level for an application kernel, we calculate the range of relative deviation from the mean application runtimes for each application configuration and its repetitions individually in percent. We do this for each kernel with a runtime larger than 1% of the total application runtime and obtain one data point for each kernel and measurement point combination. Furthermore, the violin plots show the distribution of the noise levels, i.e., which percentage of kernels are affected by a certain noise level.

Our noise analysis shows that Kripke's measurements, which were conducted on Vulcan, are the least noisy of all case studies. As visible in the distribution in Figure 8, most of the Kripke measurements have a noise level ranging between 0.06% and 2.17%, with a maximum of $n_{max} = 16.57\%$. The negligible run-to-run variation explains why all measurement point selection strategies achieve high model accuracies with a minimal modeling budget. Additionally, the low noise level makes repeating already measured configurations less beneficial. Thus, the GPR strategy prioritizes unseen measurement configurations over repetitions, resulting in a high model accuracy for all application kernels, as shown in Figure 7.

We observed similar noise levels for the case studies conducted on Lichtenberg, with MiniFE being an outlier due to the previously described scalability bug we identified with our analysis. For LULESH, Quicksilver, and RELeARN, the mean noise levels range between 0.78% and 1.95%. Similarly, their median noise levels range between 4.50% and 5.66%, indicating only minimal run-to-run variations, similar to Kripke on Vulcan. Furthermore, for RELeARN and LULESH, the distribution of the noise levels in the violin plots shows that more measurements are affected by noise. For RELeARN, for example, the distribution of kernels with high noise levels, up to $n_{max} = 19.84\%$, is much higher than for the other case studies. Consequently, more measurement repetitions are required to counter the effects of system noise, which explains why all measurement point selection strategies need more budget to create accurate performance models. In contrast, for Quicksilver, the distribution of the noise levels suggests

TABLE IV
COMPARISON OF RELATED APPROACHES FOR PERFORMANCE MODELING AND PREDICTION.

	Objective	Examples	Approach	Model type	Interpretability	Scalability focus	Key limitations
Semi-automated	Simplify analytical performance modeling, model portability	PALM [23] ASPEN [24]	Annotation-based modeling using source/DSL annotations	Symbolic	High	Partial	Requires manual annotations; limited automation
	Statistical performance tuning and execution-based prediction	Vuduc et al. [25] Jayakumar et al. [26]	Modeling based on empirical data, kernel databases using manual regression/pattern matching for learning	Symbolic, pattern-based	Medium	No	Requires user hypothesis selection; limited to known kernels
	Power management for performance/energy optimization on heterogeneous systems	OPEN [27]	Statistical surrogate models using runtime profiling and collaborative filtering and regression for learning	Statistical (online) and regression (offline)	High	Yes	Limited to CPU/GPU heterogeneous systems
Automated	Predict how different configuration options affect system performance	Siegmund et al. [28]	Sampling-based linear regression to learn models from configuration options and performance measurements	Statistical	Low	No	Coarse-grained, monolithic models; cannot capture fine-grained interactions
	Predict runtime using a combination of system measurements and simplified application models	Carrington et al. [29]	Automated empirical performance prediction using symbolic regression based on profiling and system runtime data	Symbolic	High	Yes	Challenges in modeling all performance aspects
	Improve automatic performance modeling by leveraging compiler analysis techniques	Hoefler et al. [30]	Compiler-extracted program features combined with limited runtime measurements to build regression-based performance models	Symbolic	Medium	Yes	Restricted search space; limited model diversity and accuracy
Automated & ML-enhanced	Predict application or processor performance across parallel, multicore, and virtualized systems	Ipek et al. [31] Lee et al. [32] Kundu et al. [33] Li et al. [34]	Black-box machine learning for empirical performance modeling based on benchmark data, design-space samples; using ANNs and regression trees for learning	Black-box ML	Low	No	Opaque models and limited interpretability
	Predict and optimize HPC storage and I/O performance, including variability	VAE-ABO [35] Madireddy et al. [36] Xu et al. [37]	I/O and storage measurements are used to build predictive models through VAE-guided Bayesian optimization and GP-based modeling	Probabilistic	Medium	Yes (I/O)	Limited to I/O performance; requires domain-specific instrumentation
	Predict iteration times of HPC applications	LSTM-Feat [38]	Surrogate models are built using network features and application characteristics as inputs; models are trained on sampled execution data	Statistical	Low	Yes (network)	Requires network-level instrumentation; limited generalization
	Efficiently model application performance by minimizing the number of costly measurements	Duplyakin et al. [39] HiPerBOt [40] Balaprakash et al. [41] Dieguez et al. [42]	Use active learning, Bayesian optimization, or cost-aware surrogate modeling to iteratively select the most informative configurations for tuning and performance prediction	Statistical / probabilistic	Medium	Partial	Iterative sampling overhead; limited generalization beyond observed regions
	Analyze application performance across high-dimensional configuration spaces while minimizing measurement cost	Neumann et al. [43] Schmid et al. [44] SBM [45]	Use sparse grid regression or surrogate modeling trained on selected runtime measurements to build cost-effective, predictive performance models	Statistical	Medium	Partial	High data requirements; limited to observed parameter spaces
	Predict the performance of parallel applications on different hardware platforms	Yang et al. [46]	Partial executions on a reference platform are used to build analytical/statistical models for cross-platform performance prediction	Statistical	Medium	Yes (cross-system)	Assumes consistent scaling behavior; limited accuracy for nonlinear performance trends
	Predict and optimize application performance across tasks, configurations, and architectures	Jamshidi et al. [47] Nichols et al. [48] Luszczek et al. [49] GPTuneBand [50]	Use transfer learning, multitask modeling, and multi-fidelity Bayesian optimization to build predictive models and guide autotuning with minimal measurements	Statistical / probabilistic	Medium	Partial	Requires related systems or auxiliary models; computationally expensive
	Detect scalability bottlenecks, identify code regions that limit performance at large scale	**Extra-P** (this work) [3]	Fully automatic model generation based on small-scale experiments; linear regression and GPR for budget/noise-aware learning	Symbolic	High	Yes	Assumes stable region behavior across scale

that there are fewer kernels influenced by noise. Even though we observed a maximum noise level of $n_{max} = 53.17\%$, the Quicksilver measurements are overall less affected by noise. The modeling results confirm this observation, as we can reach very high model accuracies using only a minimal budget.

For MiniFE, the mean $\bar{n} = 10.22\%$ and median $\tilde{n} = 6.9\%$ noise levels are twice as high as for the other case studies measured on Lichtenberg. The higher noise level in the measurements can be attributed to the application configurations used for conducting the performance measurements. We ran MiniFE at a significantly larger scale than the other case studies, using up to 2 048 MPI ranks, compared to a maximum of 1 000 ranks for LULESH and 512 ranks for Quicksilver. For the exact application configurations used for the analysis, please see Table II. Running the application at a larger scale increases the chance of system noise affecting the performance measurements, e.g., through OS noise, concurrently running jobs, or network congestion. Finally, the identified scalability bug further contributes to the higher run-to-run variations.

For FASTEST, the noise level we observe is generally comparable to the other case studies with a mean noise level of $\bar{n} = 8.17\%$ and a median noise level of $\tilde{n} = 1.52\%$. Even though the maximum noise level of $n_{max} = 150.87\%$ found in the measurements is very high, the violin plot's distribution shows that almost no measurements are affected by this noise level. Due to FASTEST's unique measurement point matrix and configuration requirements, as described in Section V-B, more measurement points are required to create accurate performance models. Extrapolating its performance using fewer points or more repetitions is not possible despite the overall low impact of system noise on the measurements. Nevertheless, this spike still suggests that on SuperMUC, considerable run-to-run variations may be rare but not impossible.

F. Discussion of the results

We observed that more measurement points do not automatically lead to better model accuracy. Too many measurement points or repetitions often lead to overfitting of the training data, leading to poor prediction accuracy for unseen application configurations. Having a complete matrix of measurement points is practically never required. As shown by the results in Figure 7, the highest model accuracy can always be achieved (and sometimes only) using only a fraction of the budget.

Our noise analysis showed that run-to-run variations significantly impact our strategies' measurement point selection requirements. Depending on the noise level it encounters, the GPR strategy makes smarter decisions when choosing between more repetitions or additional unseen measurement configurations, improving model accuracy compared to the CPF approach. As for the synthetic evaluation, we could not derive any generic selection strategy from the selections of the GPR-based approach, which further highlights that there is no trivial way to derive a generic optimal selection strategy. Therefore, as suggested in Section III, we conclude that the best point selection strategy is to measure a set of the cheapest available points, including two repetitions for each configuration, that satisfies the minimum modeling requirements of Extra-P (see Section II). Subsequently, one can use the GPR strategy to identify further measurement points to improve the models predictive power if more modeling budget is available.

VI. Related Work

Table IV provides a comparative overview of representative performance modeling approaches, categorized by their automation level and degree of learning integration. The table contrasts each method's objective, modeling approach, model form, interpretability, scalability focus, and limitations. Furthermore, it highlights how Extra-P uniquely combines fully automatic model construction, active learning, and symbolic scaling models while maintaining high interpretability and scalability across application scales.

VII. Conclusion

The GPR-based strategy achieves a much better balance between model accuracy and measurement cost by selecting points individually for each modeling task, reducing the need for expert knowledge or manual analysis. In our synthetic evaluation, with two model parameters and 5% induced noise, the naive approach achieved 66% accuracy, which included trying all possible measurement points and five repetitions. In contrast, at only 10% of the naïve approach's cost, the generic approach achieved 47.3% accuracy, while the GPR-based approach achieved even 77.8% accuracy. Similar improvements were observed across experiments with varying numbers of model parameters and noise levels. Case studies of six real-world applications confirmed that the GPR strategy can improve model accuracy by up to 64.3% with the same measurement budget as CPF. By making performance modeling more accurate, affordable, and intuitive, this strategy significantly enhances the usability and applicability of Extra-P. An implementation is publicly available in Extra-P version 4.2.0. Future work will explore hybrid approaches integrating acquisition functions, such as EI or UCB, with GPR to improve the exploration–exploitation trade-off.

Acknowledgment

The idea for this paper was first explored in the master thesis of Benedikt Nauman [51], a co-author. The work on this paper was funded by the German Research Foundation (DFG) – Project No. 449683531 (ExtaNoise). Moreover, it obtained funding from the European Commission and the German Federal Ministry of Research, Technology and Space (BMFTR) under the EuroHPC Programmes DEEP-SEA (GA No. 955606, BMFTR funding No. 16HPC015) and ADMIRE (GA No. 956748, BMFTR funding No. 16HPC006K), which received support from the European Union's Horizon 2020 programme and DE, FR, ES, GR, BE, SE, GB, CH (DEEP-SEA) and DE, FR, ES, IT, PL, SE (ADMIRE), respectively. Furthermore, the authors gratefully acknowledge BMFTR and the Hessian Ministry for Science and the Arts (HMWK) for supporting this work as part of the NHR funding. The authors also sincerely appreciate the computing time provided to them on the high-performance computer Lichtenberg at Technical University of Darmstadt, which is funded by BMFTR and the State of Hesse. Finally, the authors feel indebted for further computing time on the supercomputers Vulcan at Lawrence Livermore National Laboratory and SuperMUC at Leibniz Supercomputing Centre.

References

[1] M. M. Mathis, N. M. Amato, and M. L. Adams, "A general performance model for parallel sweeps on orthogonal grids for particle transport calculations," College Station, TX, USA, Tech. Rep., 2000.

[2] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, ser. SC'01. ACM, 2001, pp. 37–49.

[3] A. Calotoiu, T. Hoefler, M. Poke, and F. Wolf, "Using automated performance modeling to find scalability bugs in complex codes," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 1–12. [Online]. Available: https://doi.org/10.1145/2503210.2503277

[4] S. Shudler, A. Calotoiu, T. Hoefler, A. Strube, and F. Wolf, "Exascaling your library: Will your implementation meet your expectations?" in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15, ACM. New York, NY, USA: Association for Computing Machinery, 2015, p. 165–175. [Online]. Available: https://doi.org/10.1145/2751205.2751216

[5] "Extra-P, Automated performance modeling for HPC applications," 2024, ver. 4.2.1. [Online]. Available: https://github.com/extra-p/extrap

[6] M. Ritter, A. Calotoiu, S. Rinke, T. Reimann, T. Hoefler, and F. Wolf, "Learning cost-effective sampling strategies for empirical performance modeling," in *Proceedings of the 34th International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 884–895.

[7] A. Calotoiu, D. Beckinsale, C. W. Earl, T. Hoefler, I. Karlin, M. Schulz, and F. Wolf, "Fast multi-parameter performance modeling," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2016, pp. 172–181.

[8] I. Karlin, J. Keasler, and R. Neely, "LULESH 2.0 updates and changes," Tech. Rep. LLNL-TR-641973, August 2013.

[9] D. A. Iancu and N. Trichakis, "Pareto efficiency in robust optimization," *Management Science*, vol. 60, no. 1, pp. 130–147, 2014.

[10] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics and Optimization*, vol. 4, no. 1, pp. 41–59, 1977.

[11] C. E. Rasmussen, C. K. Williams *et al.*, *Gaussian processes for machine learning*. Springer, 2006, vol. 1.

[12] D. an Mey, S. Biersdorf, C. Bischof, K. Diethelm, D. Eschweiler, M. Gerndt, A. Knüpfer, D. Lorenz, A. Malony, W. E. Nagel *et al.*, "Score-P: A unified performance measurement system for petascale applications," in *Competence in High Performance Computing 2010*. Springer, 2011, pp. 85–97.

[13] M. Kornhaas, M. Schäfer, and D. Sternel, "Efficient numerical simulation of aeroacoustics for low mach number flows interacting with structures," *Comput Mech*, vol. 55, pp. 1143–1154, 2015.

[14] A. J. Kunen, T. S. Bailey, and P. N. Brown, "Kripke-a massively parallel transport mini-app," Lawrence Livermore National Lab (LLNL), Livermore, CA (United States), Tech. Rep., 2015.

[15] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[16] P. T. Lin, M. A. Heroux, R. F. Barrett, and A. B. Williams, "Assessing a mini-application as a performance proxy for a finite element method engineering application," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5374–5389, 2015.

[17] P. Brantley, S. Dawson, S. McKinley, M. O'Brien, D. Peters, M. Pozulp, G. Becker, K. Mohror, and A. Moody, "Llnl mercury project trinity open science final report," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-689799, 2016.

[18] D. F. Richards, R. C. Bleile, P. S. Brantley, S. A. Dawson, M. S. McKinley, and M. J. O'Brien, "Quicksilver: a proxy app for the monte carlo transport code mercury," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 866–873.

[19] M. Butz and A. van Ooyen, "A simple rule for dendritic spine and axonal bouton formation can account for cortical reorganization after focal retinal lesions," *PLoS computational biology*, vol. 9, no. 10, p. e1003259, 2013.

[20] S. Rinke, M. Butz-Ostendorf, M.-A. Hermanns, M. Naveau, and F. Wolf, "A scalable algorithm for simulating the structural plasticity of the brain," *Journal of Parallel and Distributed Computing*, vol. 120, pp. 251–266, 2018.

[21] F. Czappa, A. Geiß, and F. Wolf, "Simulating structural plasticity of the brain more scalable than expected," *Journal of Parallel and Distributed Computing*, vol. 171, pp. 24–27, Jan. 2023. [Online]. Available: https://arxiv.org/abs/2210.05267

[22] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," *Nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[23] N. R. Tallent and A. Hoisie, "Palm: Easing the burden of analytical performance modeling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC'14. New York, NY, USA: ACM, 2014, pp. 221–230. [Online]. Available: http://doi.acm.org/10.1145/2597652.2597683

[24] K. L. Spafford and J. S. Vetter, "Aspen: A domain specific language for performance modeling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC'12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 84:1–84:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389110

[25] R. Vuduc, J. W. Demmel, and J. A. Bilmes, "Statistical models for empirical search-based performance tuning," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 65–94, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1177/1094342004041293

[26] A. Jayakumar, P. Murali, and S. Vadhiyar, "Matching application signatures for performance predictions using a single execution," in *Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2015)*, May 2015, pp. 1161–1170.

[27] Z. Zheng, Z. Lan, X. Wu, V. E. Taylor, and M. E. Papka, "Coordinated power management on heterogeneous systems," *arXiv preprint arXiv:2508.07605*, 2025.

[28] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 284–294. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786845

[29] L. Carrington, A. Snavely, and N. Wolter, "A performance prediction framework for scientific applications," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 336–346, February 2006. [Online]. Available: http://dx.doi.org/10.1016/j.future.2004.11.019

[30] A. Bhattacharyya, G. Kwasniewski, and T. Hoefler, "Using compiler techniques to improve automatic performance modeling," in *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT'15)*, San Francisco, CA, USA, 2015, pp. 1–12.

[31] E. Ipek, B. R. De Supinski, M. Schulz, and S. A. McKee, "An approach to performance prediction for parallel applications," in *European Conference on Parallel Processing*. Springer, 2005, pp. 196–205.

[32] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee, "Methods of inference and learning for performance modeling of parallel applications," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 2007, pp. 249–258.

[33] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *ACM Sigplan Notices*, vol. 47, no. 7. ACM, 2012, pp. 3–14.

[34] B. Li, L. Peng, and B. Ramadass, "Accurate and efficient processor performance prediction via regression tree based modeling," *Journal of Systems Architecture*, vol. 55, no. 10-12, pp. 457–467, 2009.

[35] M. Dorier, R. Egele, P. Balaprakash, J. Koo, S. Madireddy, S. Ramesh, A. D. Malony, and R. Ross, "Hpc storage service autotuning using variational-autoencoder-guided asynchronous bayesian optimization," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2022, pp. 381–393.

[36] S. Madireddy, P. Balaprakash, P. Carns, R. Latham, R. Ross, S. Snyder, and S. M. Wild, "Machine learning based parallel i/o predictive modeling: A case study on lustre file systems," in *International Conference on High Performance Computing*. Springer, 2018, pp. 184–204.

[37] L. Xu, Y. Hong, M. D. Morris, and K. W. Cameron, "Prediction for distributional outcomes in high-performance computing input/output variability," *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 73, no. 3, pp. 561–580, 2024.

[38] X. Xu, K. A. Brown, T. Mallick, X. Wang, E. Cruz-Camacho, R. B. Ross, C. D. Carothers, Z. Lan, and K. Shu, "Surrogate modeling for hpc application iteration times forecasting with network features," in *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2024, pp. 93–97.

[39] D. Duplyakin, J. Brown, and R. Ricci, "Active learning in performance analysis," in *2016 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2016, pp. 182–191.

[40] H. Menon, A. Bhatele, and T. Gamblin, "Auto-tuning parameter choices in hpc applications using bayesian optimization," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 831–840.

[41] P. Balaprakash, R. B. Gramacy, and S. M. Wild, "Active-learning-based surrogate models for empirical performance tuning," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–8.

[42] A. P. Dieguez, M. Choi, M. Okyay, M. Del Ben, B. M. Wong, and K. Z. Ibrahim, "Cost-effective methodology for complex tuning searches in hpc: Navigating interdependencies and dimensionality," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2024, pp. 792–801.

[43] P. Neumann, "Sparse grid regression for performance prediction using high-dimensional run time data," in *Euro-Par 2019: Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, August 26–30, 2019, Revised Selected Papers 25*. Springer, 2020, pp. 601–612.

[44] L. Schmid, T. Sağlam, M. Selzer, and A. Koziolek, "Cost-efficient construction of performance models," in *Proceedings of the 4th Workshop on Performance EngineeRing, Modelling, Analysis, and VisualizatiOn STrategy*, 2024, pp. 1–7.

[45] B. Bogale, I. Lumsden, D. Sukkari, D. Yokelson, S. Brink, O. Pearce, and M. Taufer, "Surrogate models for analyzing performance behavior of hpc applications using the raja performance suite," in *International Conference on Computational Science*. Springer, 2025, pp. 327–335.

[46] L. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 40–40.

[47] P. Jamshidi, N. Siegmund, M. Velez, C. Kästner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 2017, pp. 497–508.

[48] D. Nichols, A. Movsesyan, J.-S. Yeom, A. Sarkar, D. Milroy, T. Patki, and A. Bhatele, "Predicting cross-architecture performance of parallel programs," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2024, pp. 570–581.

[49] P. Luszczek, W. M. Sid-Lakhdar, and J. Dongarra, "Combining multitask and transfer learning with deep gaussian processes for autotuning-based performance engineering," *The International Journal of High Performance Computing Applications*, vol. 37, no. 3-4, pp. 229–244, 2023.

[50] X. Zhu, Y. Liu, P. Ghysels, D. Bindel, and X. S. Li, "Gptuneband: Multi-task and multi-fidelity autotuning for large-scale high performance computing applications," in *Proceedings of the 2022 SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, 2022, pp. 1–13.

[51] B. C. Naumann, "Automated performance modelling using Gaussian process regression," Master's thesis, Technical University of Darmstadt, Darmstadt, Hesse, Germany, 2020, (unpublished).