Received: 18 October 2022 | Revised: 15 February 2023 | Accepted: 14 April 2023

DOI: 10.1002/sta4.576

ORIGINAL ARTICLE

WILEY

A localized ensemble of approximate Gaussian processes for fast sequential emulation

Kellin N. Rumsey[1] | Gabriel Huerta[2] | J. Derek Tucker[2]

[1]Statistical Sciences, Los Alamos National Laboratory, Los Alamos, New Mexico, USA

[2]Statistical Sciences, Sandia National Laboratories, Albuquerque, New Mexico, USA

Correspondence
Kellin Rumsey, Statistical Sciences, Los Alamos National Laboratory, P.O. Box 1663 Los Alamos, 87545 NM, USA.
Email: knrumsey@lanl.gov

Funding information
Laboratory Directed Research and Development

More attention has been given to the computational cost associated with the fitting of an emulator. Substantially less attention is given to the computational cost of using that emulator for prediction. This is primarily because the cost of fitting an emulator is usually far greater than that of obtaining a single prediction, and predictions can often be obtained in parallel. In many settings, especially those requiring Markov Chain Monte Carlo, predictions may arrive sequentially and parallelization is not possible. In this case, using an emulator procedure which can produce accurate predictions efficiently can lead to substantial time savings in practice. In this paper, we propose a global model approximate Gaussian process framework via extension of a popular local approximate Gaussian process (laGP) framework. Our proposed emulator can be viewed as a treed Gaussian process where the leaf nodes are laGP models, and the tree structure is learned greedily as a function of the prediction stream. The suggested method (called leapGP) has interpretable tuning parameters which control the time-memory trade-off. One reasonable choice of settings leads to an emulator with a $\mathcal{O}(N^2)$ training cost and makes predictions rapidly with an asymptotic amortized cost of $\mathcal{O}(\sqrt{N})$.

KEYWORDS
computer models, emulation, Gaussian process, local approximate GPs, MCMC

1 | INTRODUCTION

Computer models are an integral feature of modern science, playing a central in role the field of uncertainty quantification. Black-box computer models which simulate complex physical systems are often expensive to run or are otherwise proprietary. Instead of having unlimited access to a computer model, it is common to provide a practitioner with a set of *N model runs*. Conditional on these model runs, an *emulator* (also called surrogate) is built as a cheap-to-evaluate approximation to the computer model output for all input values of interest, including those which were not provided explicitly in the model runs.

The Gaussian process (GP; reviewed in Section 2.1) is the gold-standard of emulation but requires $\mathcal{O}(N^3)$ time to train, rendering it infeasible for computer models with high-dimensional inputs or a non-stationary and non-smooth output, which can require a large set of computer model runs to effectively summarize the input–output relationship. In this setting, we are forced to consider one of the many alternatives such as methods for approximate GP regression or non-GP emulation.

While much attention has been given to alleviating the cost of training an emulator, little attention is typically given to the time and resources required for making efficient predictions. This is primarily because it is usually more expensive to fit an emulator than it is to make a single

prediction, and predictions can commonly be made in parallel for a large number of input locations at once. There are however many important applications in which the input locations for prediction arrive sequentially and parallelization is not possible. For instance, in Bayesian model calibration (Higdon et al., 2004; Kennedy & O'Hagan, 2001) the posterior is explored using Markov Chain Monte Carlo (MCMC), in which the input location needed for prediction at time t is not even generated until after processing the emulator prediction at time $t-1$. Although running multiple MCMC chains simultaneously or using specialized MCMC algorithms such as Hamiltonian Monte Carlo (Hoffman et al., 2014) may allow for some limited parallelization, the prediction scheme is fundamentally sequential. Work on sequential contour estimation (Ranjan et al., 2008) and stochastic optimization with simulated annealing (Bertsimas & Tsitsiklis, 1993) or genetic algorithms (Holland, 1992) are additional distinct applications in which sequential emulation may be desirable. When parallelization of the emulator predictions is not possible and a large number of predictions are required, the need for emulators which can predict efficiently becomes increasingly important.

There are many options for emulation when a standard GP is infeasible. Some popular non-GP emulators include multivariate adaptive regression splines (MARS and BMARS Francom & Sansó, 2020; Friedman, 1991), additive regression trees (random forests and BART Chipman et al., 2010; Biau & Scornet, 2016) and projection pursuit regression (PPR and BPPR; Friedman and Stuetzle (1981); (Collins et al., 2022), each having training algorithms which scale favourably with N. In most cases, however, these thrifty alternatives are outperformed by the GP in terms of predictive performance. Moreover, the computational efficiency of online predictions for these methods is less than desirable, especially in the Bayesian implementations which have ensemble interpretations. Sparse GPs (Hensman et al., 2013; Snelson & Ghahramani, 2005; Titsias, 2009) represent a notable class of approximate GPs. They address the issue by finding a set of $M \ll N$ inducing points for which the resulting GP is a good approximation of the full GP. The training cost of such an approach is often $\mathcal{O}(NM^2)$. Others have approached this problem by looking at low-rank approximations to the GP covariance matrix (Moran & Wheeler, 2020; Solin & Särkkä, 2020). The sparse and low-rank approximations are excellent choices for many problems, but they do have disadvantages such as implicit assumptions of stationarity and a tendency to smooth over small-scale structures in the computer model. Highly relevant to the ideas proposed in this work is the class of local approximations which include nearest neighbour kriging (Cressie, 1991) and the more sophisticated local approximate Gaussian process (laGP) framework of Gramacy and Apley (2015). These local methods find "neighbours" of each new prediction location according to some criterion and predict using a small-scale GP fitted from just these points. The advantages of this approach include (i) interpolative prediction at locations which are in the model run set, (ii) ability to capture small-scale structure in the computer model and (iii) automatically handles non-stationarity of the response surface. In the context of the present discussion, the primary downside of this approach is that, because there is formally no "training-step", the local model must find the optimal neighbourhood and fit a local model for each new prediction location. If the size of the neighbourhood is $n = \mathcal{O}(N^a)$, then the cost of each prediction is $\mathcal{O}(N^{3a})$, which can be tedious and inefficient when a large number of sequential predictions are required. We briefly note that a similar class of methods, based on Vecchia approximations (Vecchia, 1988), have recently become popular (Katzfuss et al., 2020; Katzfuss & Guinness, 2021; Sauer et al., 2022). These methods generally aim to construct a sparse approximation to the Cholesky factor of the covariance matrix in $\mathcal{O}(nm^3)$ time. Since the Cholesky factor is usually constructed separately for each new prediction location, these methods will similarly suffer when a large number of sequential predictions are required. If a global Cholesky factor is constructed for all prediction locations, then the Vecchia-based approximations will have similar drawbacks to the low-rank approximations.

In this paper, we propose a global model extension to the laGP framework of Gramacy and Apley (2015) (see Section 2.2 for review) referred to as a localized ensemble of approximate GPs (leapGP). The leapGP algorithm can be viewed as a treed GP (Gramacy & Lee, 2008) in which the leaves are laGP models and the tree structure is learned greedily and is dependent on the stream of predictions. Our proposed method allows us to reuse previous work, trading time for memory, in order to obtain predictions rapidly. Although leapGP is meant to approximate the behaviour of standard laGP, the method induces a form of local regularization which can lead to better emulation in certain settings. Our proposal has two tuning parameters (M_0, ρ) which influence both the accuracy and computational efficiency of leapGP, but we propose a default setting which leads to a $\mathcal{O}(N^2)$ training algorithm with an amortized cost of $\mathcal{O}(\sqrt{N})$ per prediction ($\mathcal{O}(N)$, if uncertainty is also requested). Code to reproduce all of the examples in this paper can be found online (https://github.com/knrumsey/leapGP-analysis-comparison). Standard GP regression and laGP regression are reviewed in Section 2, and the leapGP is described in Section 3. We demonstrate the effectiveness of the proposed method in Section 4, comparing our approach to standard laGP and several other alternatives. Conclusions are given in Section 5.

2 | BACKGROUND

2.1 | GPs

In GP emulation, the goal is to learn the behaviour of an unknown function (computer model) f which is observed only at a finite set of input locations

$$\begin{aligned} \boldsymbol{y} &= f(\boldsymbol{x}) + \epsilon, \epsilon_1, ..., \epsilon_n \overset{\text{iid}}{\sim} N(0, \tau), \\ \boldsymbol{x} &= [x_1, ..., x_N], x_i \in \mathcal{X} \subset \mathbb{R}^d. \end{aligned} \tag{1}$$

In the typical case where f is deterministic, the variance term τ is called a *nugget* and is usually set to a very small fixed constant for improved numerical stability. The GP specifies a prior distribution over the infinite dimensional function space as

$$f(x) \sim \mathrm{GP}(\mu(x), K(x,x')), \tag{2}$$

which is fully defined by the mean function $\mu(x)$ and the covariance function $K(x,x')$, the latter of which depends on a set of parameters denoted $\boldsymbol{\kappa}$. Although there are many reasonable alternatives, we will restrict our attention to the isotropic Gaussian covariance function

$$K(x,x';\boldsymbol{\kappa}) = \kappa_1 \exp\left[-(x-x')^2/\kappa_2\right].$$

We also note that taking $\mu(x) = 0$ for all x is a standard simplifying assumption in the computer model literature. For a new input location x^*, the vector $\left[f(x^*)f(\boldsymbol{x})^\top\right]^\top$ has a multivariate normal distribution. Through Gaussian conditioning, the predictive distribution for f at x^* is normal with

$$\begin{aligned} \text{mean } \mu(x^*|\boldsymbol{x},\boldsymbol{y},\mu(\cdot),K(\cdot,\cdot)) &= k^\top(x^*)K^{-1}\boldsymbol{y}, \\ \text{and variance } \nu(x^*|\boldsymbol{x},\boldsymbol{y},\mu(\cdot),K(\cdot,\cdot)) &= \frac{\psi}{N}\left[K(x^*,x^*) - k^\top(x^*)K^{-1}k(x^*)\right], \end{aligned} \tag{3}$$

where K is an $N \times N$ matrix with $(ij)^{\text{th}}$ entry $K(x_i,x_j)$, $k(x^*)$ is an N-vector with i^{th} entry equal to $K(x^*,x_i)$ and $\psi = \boldsymbol{y}^\top K^{-1}\boldsymbol{y}$. Estimation of the parameters $\boldsymbol{\kappa}$ and τ can be done using maximum likelihood or via empirical Bayes, and we refer the reader to Rasmussen (2003) and Gramacy (2020) for a more detailed discussion.

GP emulation scales poorly because Equation (3) requires the inverse of the potentially large matrix K. Since K varies with the unknown parameters $\boldsymbol{\kappa}$, many cubic-time inversions may be needed. By fixing κ_2 at an empirically reasonable value, substantial speedup can be obtained in applications such as Bayesian model calibration(Rumsey & Huerta, 2021), but it remains a computational bottleneck and limits deployment of the full scale GP to problems with only a moderate number of model runs.

2.2 | laGP

Since the primary computational drawback of the GP is related to inverting an $N \times N$ matrix, the laGP approach attacks this directly by requiring just the inverse of an $n \times n$ matrix for $n \ll N$. Subset of data (SoD) approaches (reviewed in Liu et al., 2020) are a simple and intuitive approach to approximate GP emulation, in which a subset of the full data is retained and used for the calculations in Equation (3). The laGP approach of Gramacy and Apley (2015) takes a localized version of this idea, allowing for a unique subset of the data to be selected for each input location x.

The nearest neighbour *sub-design*, which consists of the points $\boldsymbol{x}_n(x) \subset \boldsymbol{x}$ which are nearest to x, has been used for decades and has been extensively studied (Cressie, 1991). This design is known to be suboptimal (Vecchia, 1988) and is uniformly sub-optimal under certain conditions (Stein et al., 2004). Finding truly optimal sub-designs remains out of reach, except for in trivial cases, as it involves complicated, high-dimensional, non-convex optimization which varies explicitly with $\boldsymbol{\kappa}$. Instead, the laGP approach relies on a greedy construction of the sub-design using active learning (e.g. Cohn et al., 1996) which is demonstrably more effective than nearest neighbours and can be constructed quickly without increasing the asymptotic complexity of the procedure.

To make a prediction (with uncertainty) at a new input location x^*, laGP proceeds as follows. First, a small nearest neighbour sub-design (of size $n_0 < n$) is obtained. For $j = n_0 + 1, \ldots n$, the j^{th} observation is added to the sub-design one at a time by evaluating a criterion and greedily selecting the best point. At step j, if the criterion can be evaluated for a set of candidate points not currently in the design in $\mathcal{O}(j^2)$ time, then the complexity of the entire procedure will remain $\mathcal{O}(n^3)$. If $(\boldsymbol{x}_j(x), \boldsymbol{y}_j(x))$ is the local sub-design at step j, then next point is selected according to the rule

$$x_{j+1} = \underset{x \notin \boldsymbol{x}_j(x)}{\operatorname{argmin}} \left\{ \mathbb{E}\left(\left[f(x) - \mu_{j+1}(x; \hat{\boldsymbol{\kappa}}_{j+1})\right]^2 | \boldsymbol{x}_j(x), \boldsymbol{y}_j(x) \right) \right\}. \tag{4}$$

In the work by Gramacy and Apley (2015), it is shown that this criterion can be viewed as an empirical Bayes mean square prediction error and can be well-approximated in the allotted time. For additional details, we refer the reader to the original work and its implementation-focused companion, Gramacy (2016). Many methods for approximating Equation (4) are proposed, special cases are discussed and mathematical details are given (including the use of the partition inverse equations for fast updating). The extension that we propose is invariant to the particular method used for evaluating equation (4). In all of our examples, we use the *active learning Cohn* approximation (the default in the `laGP` R package) for both laGP and our proposed extensions. After the final sub-design of size n has been constructed, the parameters $\boldsymbol{\kappa}$ are estimated using MLE.

3 | A LOCALIZED ENSEMBLE OF APPROXIMATE GPs

As discussed extensively above, the laGP algorithm is an approximation to the full GP with several desirable properties. Emulation with laGP is treated inherently local, and the lack of a global model has both advantages and disadvantages. The primary disadvantage comes at the cost of slower predictions, since a local sub-design at input location x^* must be fully reconstructed, even if it differs very little from the design that was built for a previously seen input location $x^* + \epsilon$.

We propose a simple extension to this algorithm which builds a global model consisting of laGP *prediction hubs*. By trading memory for time, we can approximate the behaviour of standard laGP while reusing our previous work as often as possible. We make decisions based on computational efficiency wherever possible and will demonstrate that the accuracy of these approximations is comparable and sometimes superior to the standard laGP prediction. The proposed extension, which we refer to as a leapGP, is based on maintaining a data structure of previously trained laGP models (called prediction hubs). Whenever possible, we predict the value of f at a new input location x^* using a nearby prediction hub. If no prediction hub exists which is sufficiently reliable for prediction, then we construct a new hub at location x^* and add it to the data structure for future use. In a sense, this process can also be viewed as a modification of the treed GP method of Gramacy and Lee (2008) where the leaves of the tree correspond to laGP models and the tree structure is learned greedily and is determined by the stream of new prediction locations. We note that a similar two-stage approach was proposed by (Nguyen-Tuong et al., 2008), but our method improves on this earlier work by incorporating the enhanced neighbourhood design of the laGP and saves additional time by building the local models sequentially and only as needed.

3.1 | The prediction hub

A prediction hub, corresponding to the laGP model built for a particular input x, is a mathematical tuple $\mathcal{H}$ containing the minimal information needed to evaluate the GP Equation (3) for any new value x^*. We define

$$\mathcal{H} = \{x, \mathcal{I}, \hat{\boldsymbol{\kappa}}, L, \boldsymbol{\alpha}, \psi\}, \tag{5}$$

where x is the coordinates of the hub (the input location for which it was originally designed), $\mathcal{I} \subset \{1,\ldots,N\}$ is the set of indices corresponding to the sub-design $\boldsymbol{x}_n(x)$, $\hat{\boldsymbol{\kappa}}$ is the estimated hyperparameters to the covariance function and L is the Cholesky factor of $K^{-1} = LL^\top$. We also store the vector $\boldsymbol{\alpha} = K^{-1}\boldsymbol{y}$ and the scalar value $\psi = \boldsymbol{y}^\top \boldsymbol{\alpha} = (\boldsymbol{y}^\top L)(\boldsymbol{y}^\top L)^\top$ to facilitate faster evaluation of Equation (3), and we pay a negligible memory cost in doing so. We also note that, if only the mean of the emulator is required, the prediction hub does need to store L which markedly reduces the memory cost.

3.2 | Training phase

It is often worthwhile to spend some time training an initial model in advance, in order to save time later. We can achieve this with an optional *training* phase, in which M_0 prediction hubs are built in advance at a series of locations across the input domain. By setting $M_0 = \mathcal{O}(N^a)$ and $n = \mathcal{O}(N^b)$, we can guarantee that the training algorithm will scale as $\mathcal{O}(N^{a+3b})$. We take $a = b = 1/2$ as a default, so that the training scales as $\mathcal{O}(N^2)$ which is a drastic improvement over the cubic scaling of a full GP, but smaller values can be chosen when N is exceptionally large. Indeed, one can opt to skip this training phase altogether ($M_0 = 0$) although the computational advantages compared with standard laGP may be slight unless a huge number of predictions are required. We also note that this training phase is trivially parallelizable and can usually be accomplished with $\mathcal{O}(N^{3b})$ complexity in practice.

There are many reasonable schemes for choosing the location of these first M_0 hubs, but we have found that the stochastic partitioning around medoids (PAM) algorithm (Schubert & Rousseeuw, 2019) is a fast and reliable choice. The PAM algorithm is a clustering algorithm which leads to good hub placement when there is structure in the data $\boldsymbol{x}$ and is generally space filling even when there is no such structure. Moreover, the PAM algorithm is very fast, $\mathcal{O}(N^2)$ with small overhead, and can easily be bounded above by running it on a random subset of the original design, still leading to good results. Maximin Latin hypercube designs represent a simple yet reasonable alternative (Park, 1994).

3.3 | Prediction phase

After the optional training phase has completed, we assume that prediction locations $x_1^*, x_2^*, \ldots$ come as a stream, one after another, so that parallelization is not possible. At any given time, we must maintain a list of the current M predictions hubs $\mathcal{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_M)$. We also maintain a balanced k-d tree, as described by Procopiuc et al. (2003), which requires $\mathcal{O}(M)$ memory. After training, the initial k-d tree can be constructed in

$\mathcal{O}(M_0 \log^2 M_0)$ time, and when a new prediction hub is constructed during the prediction phase, we insert its coordinates into the k-d tree in $\mathcal{O}(\log M)$ time. The advantage of the k-d tree is that, for any input location x, we can find the k-nearest hubs in (expected) $\mathcal{O}(k \log M)$ time.

When predicting the value of $f(x^*)$, we must find a suitable $\mathcal{H}_m$. Rather than using the hub which is nearest to x^*, we prefer to use the hub which maximizes the correlation between x^* and the hub coordinate. For hub $\mathcal{H}_m$ ($m = 1,\ldots,M$), we compute the correlation (denoted r_m) between x^* and the coordinates of $\mathcal{H}_m$ using the correlation parameters of the corresponding hub. For the Gaussian covariance function, we calculate

$$r_m = \frac{K(x^*, \mathcal{H}_m.x; \mathcal{H}_m.\widehat{\boldsymbol{\kappa}})}{\mathcal{H}_m.\widehat{\kappa_1}}, \tag{6}$$

where the notation $\mathcal{H}.z$ denotes the field z belonging to tuple $\mathcal{H}$. The candidate hub is then given by $\mathcal{H}_{m_\star}$ where $m_\star = \text{argmax}_{m=1,\ldots,M}\{r_m\}$. The ideal approach to finding $m_\star$ would require a non-trivial modification of the balanced k-d tree which is a potential area for future work. As a sensible alternative, we use the k-d tree to find the k nearest hubs (e.g. $k = 5$) and find the largest r_m among this subset. For large M, using approximate nearest neighbour search is another avenue for speedup (Hajebi et al., 2011).

Given a user defined parameter $\rho \in [0,1]$, we accept the candidate hub if $r_{m_\star} \geq \rho$ and we compute the mean and variance of $f(x^*)$ efficiently using Equation (3) and the values contained in $\mathcal{H}_{m_\star}$. If we reject the candidate hub, then we emulate $f(x^*)$ using standard laGP and we add this newly constructed hub to our data structure and insert a new node into the k-d tree. We now have one additional hub at our disposal, with which we can make new predictions in the future. The effect of ρ on the performance of the algorithm will be discussed further in the following subsection.

It is worth noting that an alternative is to create an ensemble of predictions, perhaps using all hubs for which r_m exceeds ρ. One must exercise caution when choosing weights for the averaging scheme, and optimal weights can be costly to obtain. Moreover, we have found that this approach, which is explored by Edwards and Gramacy (2021), does not usually lead to significantly better predictions despite the significant computational burden.

3.4 | Complexity analysis

We first consider the memory cost associated with storing a single prediction hub, which is $\mathcal{O}(d+n)$ if only the emulator mean is required and $\mathcal{O}(d+n^2)$ in general. If $n \leq \sqrt{N}$, then storing a hub is smaller than the $\mathcal{O}(dN)$ cost of keeping the full data. When N is very large, we can often achieve accurate predictions with a much smaller $n \ll \sqrt{N}$. To provide an upper bound on the memory requirements, it is sensible to place a hard cap on the allowable number of hubs which can be stored. It is easy to conceive of an adaptive approach in which the tree structure of hubs is pruned and the locations optimized, but we leave this for future work.

The leapGP algorithm as described depends on just two parameters M_0 and ρ (in addition to the laGP parameters, such as neighbourhood size n). One special case which we have already discussed briefly is the case where $M_0 = 0$. In this case, no training is required and the resulting method will closely resemble laGP for large ρ, except for that time; it will be saved through the reuse of work when future prediction locations are sufficiently close to previous ones (as is common in MCMC applications). A second special case is the one where $\rho = 0$, in which prediction hubs are *always* used for emulation. This approach leads to extremely fast predictions, but its accuracy depends on choosing an appropriate value for M_0 and can require a considerable amount of time budgeted for training. Finally, we note that the case where $M_0 = 0$ and $\rho = 1$ is equivalent to using standard laGP for prediction. In summary, the parameter M_0 dictates how much effort should be front-loaded into a training phase, so that predictions can be made more efficiently later on. The parameter ρ controls the accuracy of the approximation, since values close to 1 should force behaviour which is similar to the standard laGP. Smaller values of ρ lead to a greater probability of reusing work and thus faster predictions.

Setting $n = N^a$ for some $a \in (0,1)$, the complexity of laGP is $\mathcal{O}(N^{3a})$ which is the cost of building a sub-design as well as the cost of computing the necessary terms for evaluating the emulator (i.e. K^{-1}). When using a pre-existing hub for prediction, leapGP requires just $\mathcal{O}(\log M + N^a)$ time to find the nearest hub and compute the mean of the emulator. If the variance of the emulator is also desired, the cost increases to $\mathcal{O}(\log M + N^{2a})$. The computational cost of the leapGP training phase is $\mathcal{O}(M_0 N^{3a} + M_0 \log^2 M_0)$, where the first term is due to construction of the laGP models (which can be parallelized) and the second term is due to the construction of the k-d tree. Setting $M_0 = N^b$ for $b \in (0,1)$, we can write the computational cost of leapGP as

$$\begin{aligned} &\mathcal{O}\left(\frac{N^{3a+b}}{p} + b^2 N^b \log^2 N\right) \qquad \text{for training and} \\ &\mathcal{O}\left(N^{3a}\omega_T + (\log M_T + N^{ca})(1-\omega_T)\right) \quad \text{per prediction, when making } T \text{ predictions,} \end{aligned} \tag{7}$$

where $p \leq N^b$ is the number of processors used for parallel training and $c = 2$ if the variance of the emulator is desired and $c = 1$ otherwise. M_T indicates the number of prediction hubs after T predictions and $\omega_T = (M_T - M_0)/T$ is the proportion of the first T predictions which required a new prediction hub. For a continuous computer model f over a compact domain, it is clear that a finite number of prediction hubs will suffice to effectively cover the space when $\rho < 1$. This implies that $M_T \to M < \infty$ and $\omega_T \to 0$ as $T \to \infty$, and thus, the amortized cost of prediction for leapGP is $\mathcal{O}(N^{ca})$ for sufficiently large T. The degree of separation between this ideal case and reality depends heavily on (i) the dimensionality of $\boldsymbol{x}$ and (ii) the relative smoothness of the computer model output as a function of $\boldsymbol{x}$. In general, surfaces which are harder to emulate will require more predictions (larger T) before the computational advantages of leapGP will make themselves felt. Regardless, leapGP has a relatively small overhead and will at worst be computational comparable with laGP. For the recommended default values, $a = b = 1/2$, leapGP guarantees $\mathcal{O}(N^2 + \sqrt{N}\log^2 N)$ training cost and an amortized $\mathcal{O}(\sqrt{N})$ prediction cost (or $\mathcal{O}(N)$ with uncertainty) where the overhead depends on the regularity of f and the dimension of the input space.

4 | EXAMPLES

4.1 | Illustrative example: Twin Galaxies function

We begin with the analysis of a simple bivariate function, studied by Rumsey et al. (2022) in the context of large-scale climate models. The Twin Galaxies function is highly non-isotropic and will be difficult to emulate adequately with a stationary process. The function is given by $f(\boldsymbol{x}) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})$ over $\boldsymbol{x} \in [0,1]^2$ where

$$\begin{aligned} f_1(x) &= \frac{11}{40}\left(18 + 5x_1 - 35x_2 + 5x_1x_2 + 38x_2^2 - 15x_1^3 - 5x_1x_2^2 - 11x_2^4 + x_1^3x_2^2\right) \quad \text{(Lim et al., 2002)} \\ f_2(x) &= 5\exp\left(-((8x_1-2)^2 + (8x_2-2)^2)\right)(8x_1 - 2) \qquad\qquad \text{(Gramacy \& Lee, 2008)} \end{aligned} \tag{8}$$

and is shown in Figure 1a. We begin our analysis by generating a design of $N = 1000$ observations $\boldsymbol{x}$ from the unit square $(0,1)^2$ using a Latin hypercube design. For each $\boldsymbol{x}_i$, we evaluate $y_i = f(\boldsymbol{x}_i)$ and the pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots N$ formulate the model runs (training data). We then generated a

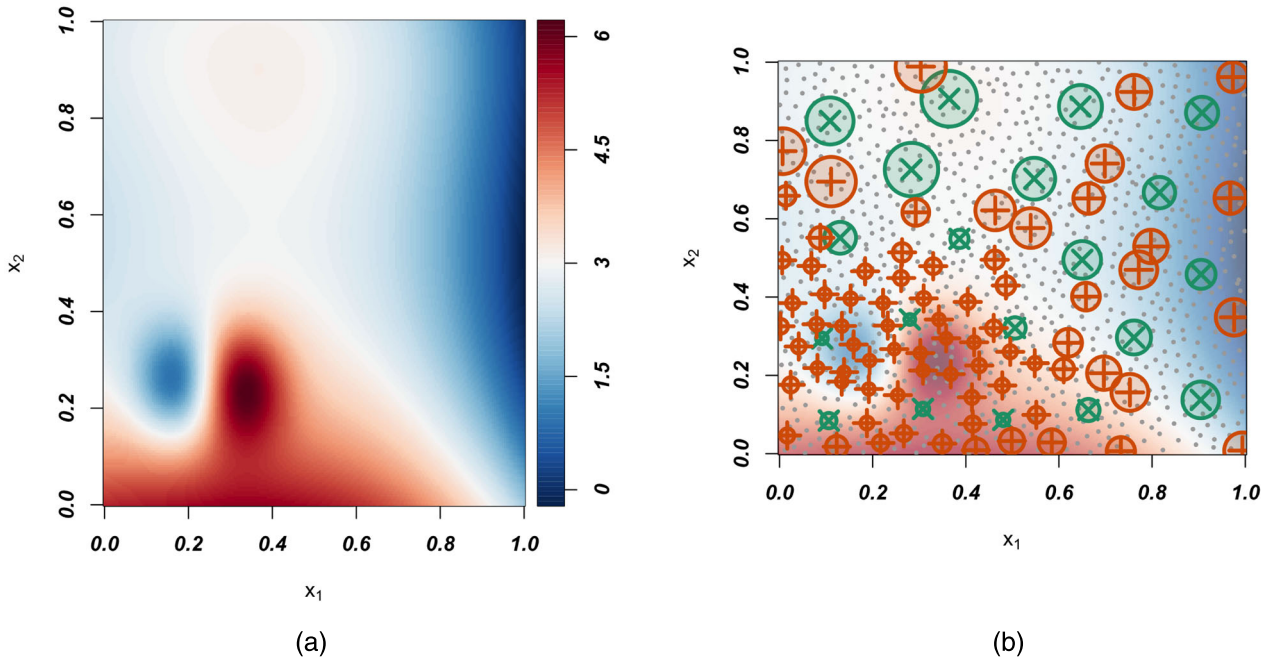

FIGURE 1 The Twin Galaxies function. (a) The Twin Galaxies function is a synthetic computer model with smooth long range structure, with regions of high activity (in the lower left corner). (b) An illustration of hub placement for leapGP. The dark grey points represent the data x. Dark green x symbols represent the initial placement of 20 hubs during the training phase. The orange + symbols represent additional hub locations that were added during the prediction phase. The radius of the circle is proportional $\sqrt{-k_2 \log \rho}$ to for each hub. Note that more hubs are needed to predict areas of high activity.

second set of $N = 1000$ distinct observations for prediction, also using a Latin hypercube design, which we assume must be predicted sequentially. Figure 1b shows the location of the prediction hubs for the leapGP algorithm with $M_0 = 20$ and $\rho = 0.9$. The green $\times$ symbols show the location of the hubs placed during the training phase, which were chosen using the fast PAM algorithm, leading to good separation of the laGP models. The orange $+$ symbols show the location of the 74 prediction hubs which were added during the prediction phase, in locations which previous hubs were deemed too far away to make an accurate prediction. The radii of the circles surrounding each hub are proportional to (but not equal to, for illustrative purposes) $\sqrt{-\kappa_2 \log(\rho)}$. We note that these radii are larger in locations where the response surface is more flat and easier to predict. Conversely, more prediction hubs are needed (and radii of hubs are smaller) in regions of high local activity in the response surface.

Where the standard laGP algorithm (with $n = 30$) required 62.3 seconds to make 1000 predictions, the leapGP algorithm $(n = 30)$ with parameters $M_0 = 20$ and $\rho = 0.9$ required just 1.3 seconds for training and 2.9 seconds to make the predictions. Moreover, the root mean square prediction error (RMSPE) for the laGP algorithm is 0.105, while the leapGP algorithm is substantially more accurate with a RMSPE of 0.032. The ability of leapGP to outperform laGP in terms of prediction (while also being significantly faster) is not uncommon in our experiments. If laGP were able to select sub-designs which were truly optimal, then this results could not occur, but this fact suggests that the greedy algorithm for sub-design construction can lead to overfitting. The leapGP algorithm induces a local smoothing on the emulation surface, a form of local regularization, where nearby locations share suboptimal sub-designs, which can sometimes lead to better predictive performance.

We also consider the extremely thrifty case where $\rho = 0$ (with $M_0 = 20$), so that only the original 20 prediction hubs will be used during prediction. In this case, the prediction phase takes just 0.098 seconds with a reasonable RMSPE of 0.32. Although the RMSPE is three times larger than that of laGP, it demonstrates that a reasonable emulator can be obtained for this problem using extremely limited resources.

To further demonstrate the effect of ρ, we fit a series of emulators for various values of $\rho \in (0.5, 0.999)$ with $M_0 = 0$, so that we skip the optional training phase altogether. Figure 2a shows the time in seconds to make each prediction from 1 to 1000. For all values of ρ, the initial behaviour of the leapGP algorithm is similar to that of laGP, but the amortized cost of each successive prediction is lowered as previous work is reused. Figure 2b shows the RMSPE of each algorithm across the 1000 predictions. When ρ is small, we see that predictions are very fast to obtain, although the resulting accuracy is worse than laGP, as expected. When $\rho \approx 1$ (e.g., $\rho = 0.999$), the behaviour of leapGP is nearly equivalent to laGP (though there is a small amount of overhead in timing). For values of ρ between about 0.8 and 0.99, however, our proposed algorithm is both markedly faster than laGP and considerably more accurate due to implicit regularization.

4.2 | Simulation study: The piston function

In this example, we compare the efficiency and accuracy of leapGP (at various parameter settings) to several other emulators for the piston simulation function (Ben-Ari & Steinberg, 2007; Zacks, 1998). The piston simulation function models the cycle time, in minutes, of a piston with inputs given in Table 1 and is defined as

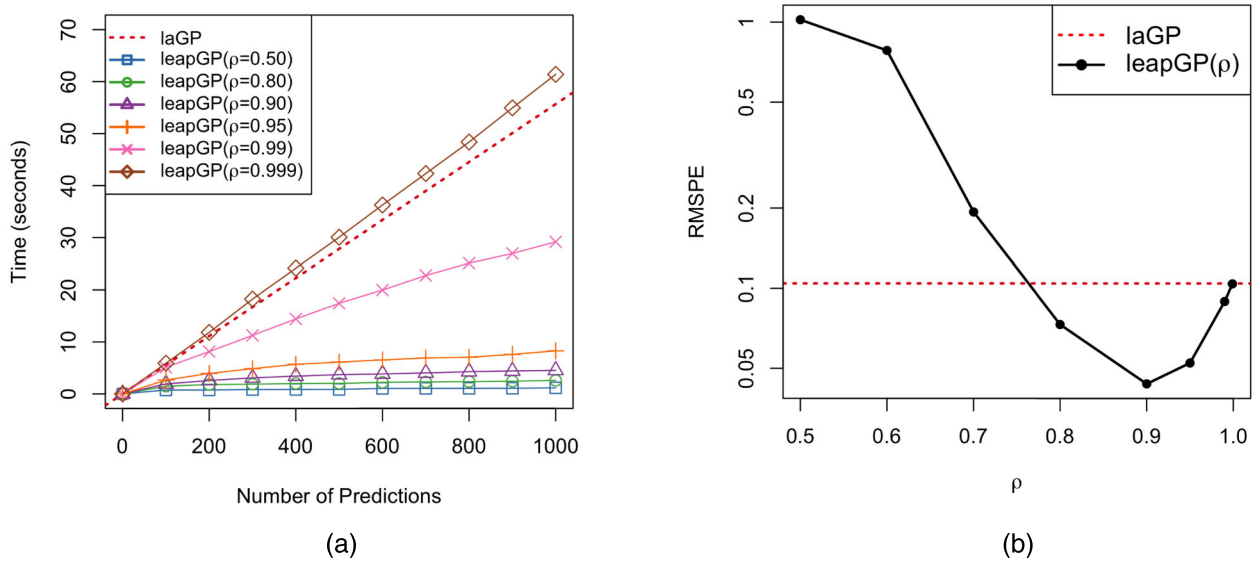

FIGURE 2 Timing and root mean square prediction error (RMSPE) results for the Twin Galaxies function. (a) Time per prediction for laGP and leapGP with $M_0 = 0$ and $\rho \in (0.5, 0.999)$. Initially, leapGP takes just as long as laGP, but becomes faster as more hubs are placed. (b) Root mean square prediction error for laGP and leapGP with $M_0 = 0$ and $\rho \in (0.5, 0.999)$. For $\rho \approx 1$, behavior is nearly equivalent to laGP. In this particular case, leapGP can outperform laGP for $\rho \approx 0.9$.

TABLE 1 Description of input variables and their ranges for the piston simulation function.

Input	Range	Description
M	[30,60]	Piston weight (kg)
S	[0.005,0.020]	Piston surface area (m^2)
V_0	[0.002,0.010]	Initial gas volume (m^3)
k	[1000,5000]	Spring coefficient (N/m)
P_0	[90000, 110000]	Atmospheric pressure (N/m^2)
T_a	[290,296]	Ambient temperature (K)
T_0	[340,360]	Filling gas temperature (K)

$$\begin{aligned} f(\boldsymbol{x}) &= 120\pi\sqrt{\frac{M}{k+S^2\frac{P_0V_0T_a}{T_0V^2}}}, \text{ where} \\ V &= \frac{S}{2k}\left(\sqrt{A^2+4k\frac{P_0V_0T_a}{T_0}}-A\right) \text{ and } A = P_0S+19.62M-\frac{kV_0}{S}. \end{aligned} \tag{9}$$

We perform a simulation study using N computer model runs for $N=4000$ and $N=40{,}000$. In both cases, we ask the emulators to produce $T=10{,}000$ predictions across the input space. Latin hypercube designs were used to generate the input locations for both the training and prediction data.

For each emulator, we record the time taken to train the emulator, the time taken to produce all predictions, the accuracy of the emulator (measured by RMSPE) and the empirical coverage of all 95% predictive confidence/credible intervals. The emulators compared include (i) Bayesian MARS using the default settings of the `BASS` package (Francom & Sansó, 2020), (ii) Bayesian additive regression trees using the default settings of the `BART` package (Sparapani et al., 2021), (iii) Bayesian PPR using the `BayesPPR` package (Collins, 2022), (iv) the fully independent training conditional sparse GP using the `gplite` package with `method = method_fitc(num_inducing = sqrt(N))` (Snelson & Ghahramani, 2005), (v) a Vecchia approximation based GP using the `GPvecchia` package (Katzfuss et al., 2020), (vi) treed regression using the default settings of the `bcart` function in the `tgp` package (Gramacy, 2007) and (vii) laGP using the `laGP` package (Gramacy, 2016).

Recall that our primary goal is to construct an emulator which is appropriate for use within an MCMC algorithm (or another sequential setting) and thus all predictions are wrapped in a simple *for loop* in R, which disallows any benefits that arise from vectorization. We recognize that some implementations, such as `wbart` in the `BART` package, are not designed for this case and have a large overhead for sequential prediction. Thus, the results of this simulation study should not necessarily be taken as a direct critique of the methods but rather a shortcoming of the method/implementation combination for the current setting of interest. We view this as further evidence that the problem of efficient online prediction is often overlooked and is one of the key motivations for leapGP. From the `tgp` package, we attempted to use the treed GP approach of Gramacy and Lee (2008), but training the model on the piston data using just the first thousand model runs took several hours and the cubic growth of the fitting process rendered it impractical for inclusion in this comparison. We use the Bayesian CART algorithm from the `tgp` package as a fast tree-based alternative. The optimization routine in `GPvecchia` was unable to converge under the default settings, so we increased the tuning parameter (`m=30` and `m=40` for $N=4000$ and $N=40{,}000$, respectively) until convergence could be obtained. Both laGP and leapGP employ sub-designs of size $n=60$ and $n=120$ for the $N=4000$ and $N=40{,}000$ cases, respectively. We train each leapGP model using $M_0=60$ and $M_0=200$ for the respective cases and obtain the 10,000 predictions using values $\rho=0.0,0.8,0.9,0.95,0.97,0.99$. The speed-accuracy trade-off is demonstrated in Figure 3a,b for the $N=4000$ case and in Figure 4a,b for the $N=40{,}000$ case. The full results of this simulation study are tabulated in Table 2.

In the $N=4000$ case, the BPPR and BASS emulators are the most accurate, followed closely by laGP and the leapGP emulators with $\rho\approx 1$. BART, BPPR and GPvecchia, while producing accurate predictions are the slowest emulators by a large margin. FITC and BCART are fast but produce the largest RMSPE. The BASS emulator (Bayesian MARS) represents a nice trade-off between time and accuracy. The laGP approach also manages to find a reasonable balance, although it is less accurate and slower than BASS for the task of sequential prediction. The leapGP models strike a similar balance, though with a tendency to emphasize making fast predictions. For example, the leapGP cases with $\rho=0{,}0.8$ and 0.9 are orders of magnitude faster than their competitors (up to 628 times faster than laGP) and although they produce worse predictions than laGP, BPPR, BASS and BART, the predictions are still quite good; the RMSPE for these models is between 2 and 2.5 times larger than the RMSPE for laGP, but it is roughly 30% lower than the inducing point GP (FITC), 5.5 times smaller than Bayesian CART and about 12.2 times smaller than the standard deviation of f across the input space. The leapGP models with $\rho\in(0.95,0.99)$ approach laGP in their behaviour, taking more time for prediction and becoming increasingly accurate. For instance, the case where $\rho=0.97$ achieves a RMSPE of 0.34 (compared with 0.30 for laGP) and is nearly twice as fast. We note that all leapGP models will (eventually) reach a point where new prediction hubs are seldom needed, and

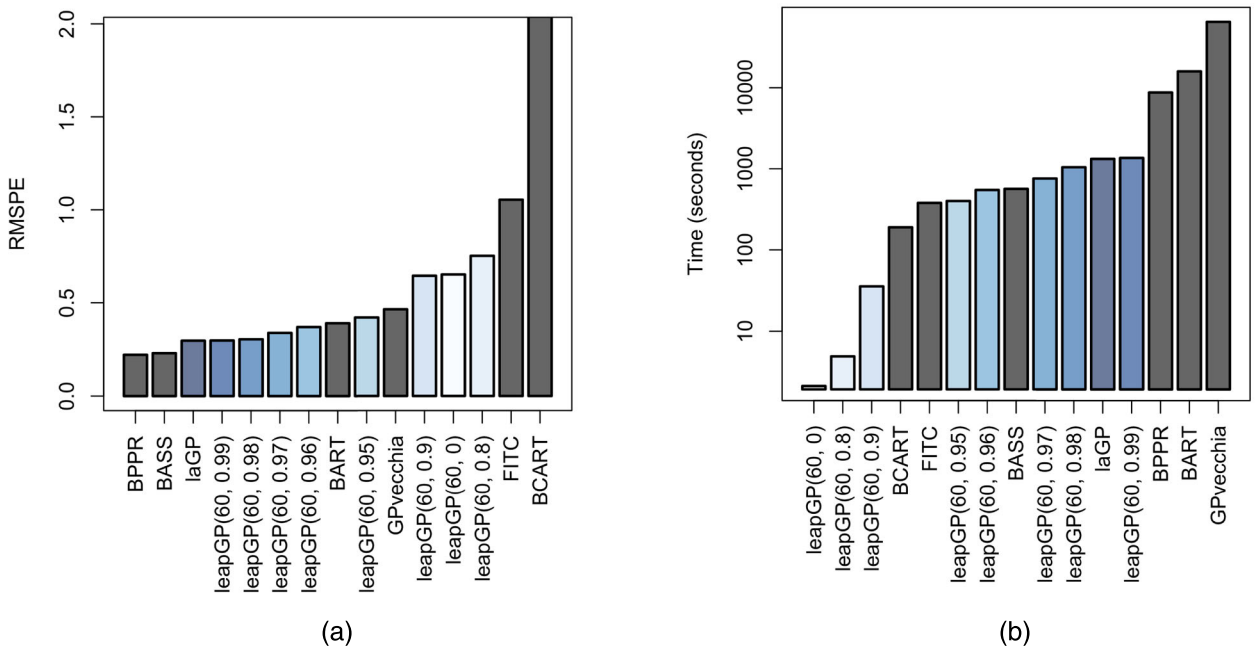

FIGURE 3 Simulation study results for the piston function with $N = 4000$. (a) RMSPE results for the suite of emulators on the piston function with $N = 4000$. BPPR and BASS have the lowest RMSPE followed by local approximate Gaussian process (laGP). The leapGP emulators are comparable with laGP, especially when ρ is near 1.0. Blue shading is used for the laGP and leapGP models, with lighter shades used to denote smaller values of ρ. The y-axis of the plot is truncated for better visualization. (b) Timing results for the suite of emulators on the piston function with $N = 4000$. The time in seconds (plotted on a log-scale) needed to make $T = 10{,}000$ sequential predictions. The leapGP methods are the most efficient by far when ρ is small. The timing results are comparable with laGP and far superior to some of the Bayesian methods when ρ is close to 1.0. Blue shading is used for the laGP and leapGP models, with lighter shades used to denote smaller values of ρ.

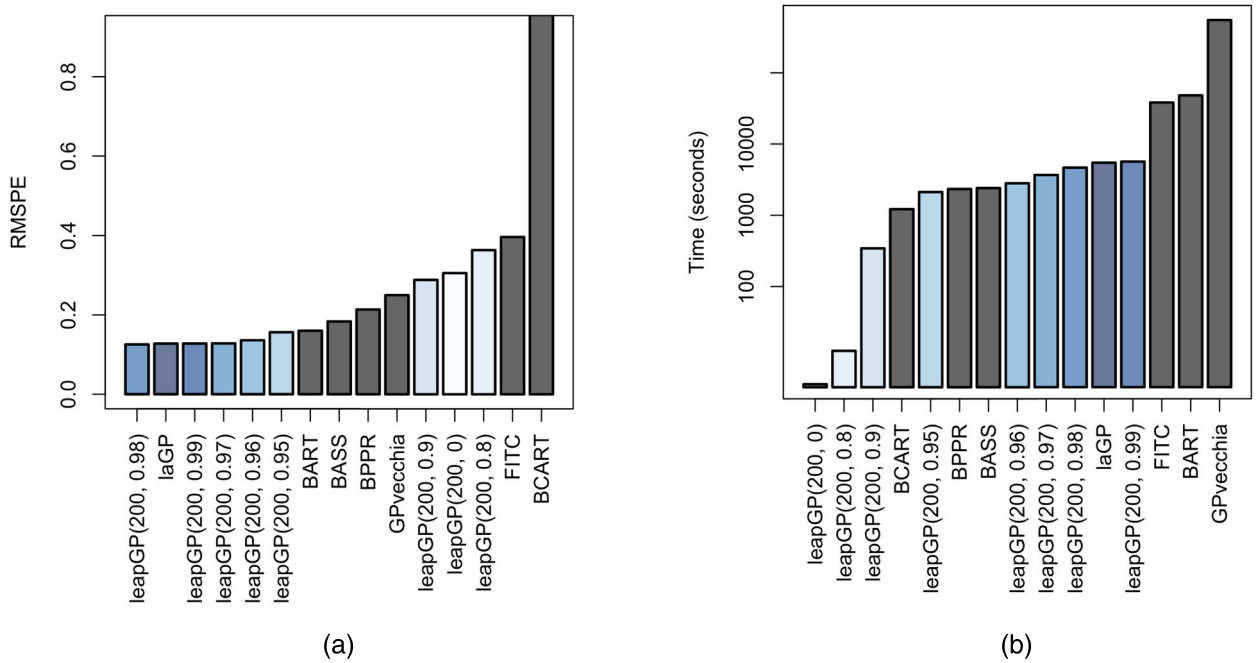

FIGURE 4 Simulation study results for the piston function with $N = 40{,}000$. (a) RMSPE results for the suite of emulators on the piston function with $N = 40{,}000$. BPPR and BASS have the lowest RMSPE followed by local approximate Gaussian process (laGP). The leapGP emulators are comparable to laGP, especially when ρ is near 1.0. Blue shading is used for the laGP and leapGP models, with lighter shades used to denote smaller values of ρ. The y-axis of the plot is truncated for better visualization. (b) Timing results for the suite of emulators on the piston function with $N = 40{,}000$. The time in seconds (plotted on a log-scale) needed to make $T = 10{,}000$ sequential predictions. The leapGP methods are the most efficient by far when ρ is small. The timing results are comparable to laGP and far superior to some of the Bayesian methods when ρ is close to 1.0. Blue shading is used for the laGP and leapGP models, with lighter shades used to denote smaller values of ρ.

TABLE 2 Piston function simulation study results for $N = 4000$ and $N = 40{,}000$.

	$N = 4000$				$N = 40{,}000$			
	Train (s)	Pred (s)	RMSPE	Coverage	Train (s)	Pred (s)	RMSPE	Coverage
BASS	37.65	565.03	0.23	0.94	295.56	2,407.75	0.18	0.95
BART	29.51	15,910.46	0.39	0.95	412.31	48,153.98	0.16	0.99
BPPR	464.19	8,734.77	0.22	0.93	34,697.81	2,334.62	0.21	0.94
FITC	13.81	379.24	1.05	0.92	1,964.64	38,204.74	0.40	0.88
GPvecchia	136.99	64,873.31	0.46	0.89	11,114.48	549,556.40	0.25	0.94
BCART	1.69	190.05	3.87	0.82	18.91	1,217.31	4.55	0.83
laGP	0	1,325.16	0.30	0.98	0	5,461.87	0.13	1.00
leapGP(M_0, 0)	24.91	2.11	0.65	0.88	144.10	4.25	0.31	0.86
leapGP(M_0, 0.8)	24.91	4.90	0.75	0.89	144.10	12.50	0.36	0.85
leapGP(M_0, 0.9)	24.91	35.45	0.65	0.87	144.10	343.23	0.29	0.85
leapGP(M_0, 0.95)	24.91	401.12	0.42	0.88	144.10	2,121.23	0.16	0.89
leapGP(M_0, 0.96)	24.91	548.59	0.37	0.90	144.10	2,813.10	0.14	0.91
leapGP(M_0, 0.97)	24.91	759.20	0.34	0.91	144.10	3,668.03	0.13	0.93
leapGP(M_0, 0.98)	24.91	1,045.43	0.31	0.94	144.10	4,660.80	0.13	0.96
leapGP(M_0, 0.99)	24.91	1,360.95	0.30	0.97	144.10	5,670.14	0.13	0.98

Note: The leapGP parameters are given as leapGP(M_0, ρ), where $M_0 = 60$ for $N = 4000$ and $M_0 = 200$ for $N = 40{,}000$. The "Predict" column gives the time (in seconds) needed to make $T = 10{,}000$ predictions. The "Coverage" column gives the empirical coverage for the T predictive 95% confidence/credible intervals. For computational reasons, BART, BPPR and GPvecchia results are extrapolated using just the first $T = 1000$ predictions.
Abbreviation: RMSPE, root mean square prediction error.

future predictions become fast, so that if this simulation was continued for another $T = 10{,}000$ predictions, we would expect the leapGP models with $\rho \approx 1$ to gain computational efficiency compared to laGP.

When we increase the size of the training set to $N = 40{,}000$, the relative performance of leapGP improves even further. The laGP and leapGP models with $\rho = 0.98$ and 0.99 are essentially tied for the lowest RMSPE. Despite some loss in predictive performance, BASS and BPPR perform well again and are reasonably fast for prediction (although we note that training for BPPR took over 9.5 hours). The leapGP emulator with ρ between 0.9 and 0.95 seems to offer an excellent and appealing trade-off between computation and accuracy. We note again that if a practitioner is willing to take a small hit in terms of accuracy, leapGP is capable of remarkable prediction speeds for small values of ρ.

5 | CONCLUSIONS

Our primary goal with this work was to develop an extension of laGP which, by trading memory for time, is able to make online predictions rapidly, while requiring little to no training time. A major advantage of leapGP is that the overhead is usually quite small compared with laGP, and so it offers a low risk alternative with possibilities for a high reward. Choosing a good value for ρ remains an important practical problem. One option is to select ρ using cross-validation, optionally penalizing larger values of ρ in order to favour faster emulation. Setting ρ close to 1 will lead to very comparable predictions, and the computational advantage is guaranteed if enough predictions are obtained. The fast predictive capabilities of leapGP with small ρ makes leapGP highly desirable for testing and debugging purposes, and the ρ parameter can always be turned up for more reliable predictions as needed.

Further analysis of the complexity and accuracy of laGP as a function of ρ is a useful avenue for future work. For instance, how many prediction hubs will be needed in the limit as $T \to \infty$. An answer to this question, which depends on the regularity of the computer model f, could provide valuable information about the number of predictions necessary for leapGP to gain a significant computational advantage over laGP as well as the values of ρ which lead to comparable predictions. Similarly, it may shed insight into the properties of f for which the naturally induced regularization of leapGP can lead to superior emulation compared to laGP (see Section 4.1).

Viewing leapGP as a greedy tree-based GP approach opens some intriguing avenues for exploration. An adaptive algorithm in which prediction hubs can be pruned or altered could immediately produce a modified algorithm which is both faster and more memory-frugal than the current version. Ideas for modifying the sub-design selection process such as using a "hot-start" approach when building a new hub or early-stopping for sub-design construction could benefit both leapGP and laGP and are completely in-line with the goals of this paper.

ORCID

Kellin N. Rumsey https://orcid.org/0000-0002-2989-965X

J. Derek Tucker https://orcid.org/0000-0001-8844-2169

REFERENCES

Ben-Ari, E. N., & Steinberg, D. M. (2007). Modeling data from computer experiments: An empirical comparison of kriging with mars and projection pursuit regression. *Quality Engineering*, *19*(4), 327–338.

Bertsimas, D., & Tsitsiklis, J. (1993). Simulated annealing. *Statistical Science*, *8*(1), 10–15.

Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*(2), 197–227.

Chipman, H. A., George, E. I., & McCulloch, R. E. (2010). Bart: Bayesian additive regression trees. *The Annals of Applied Statistics*, *4*(1), 266–298.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, *4*, 129–145.

Collins, G. (2022). Bayesppr: Bayesian projection pursuit regression. R package version 0.0.0.9000.

Collins, G., Francom, D., & Rumsey, K. N. (2022). Bayesian projection pursuit regression. arXiv preprint arXiv:2210.09181.

Cressie, N. (1991). Geostatistical analysis of spatial data. *Spatial Statistics and Digital Image Analysis*, *1991*, 87–108.

Edwards, A. M., & Gramacy, R. B. (2021). Precision aggregated local models. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *14*(6), 676–697.

Francom, D., & Sansó, B. (2020). Bass: An r package for fitting and performing sensitivity analysis of bayesian adaptive spline surfaces. *Journal of Statistical Software*, *94*, LA–UR-20-23587.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, *19*(1), 1–67.

Friedman, J. H., & Stuetzle, W. (1981). Projection pursuit regression. *Journal of the American Statistical Association*, *76*(376), 817–823.

Gramacy, R. B. (2007). TGP: An R package for bayesian nonstationary, semiparametric nonlinear regression and design by treed Gaussian process models. *Journal of Statistical Software*, *19*, 1–46.

Gramacy, R. B. (2016). LAGP: Large-scale spatial modeling via local approximate gaussian processes in r. *Journal of Statistical Software*, *72*, 1–46.

Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*: Chapman and Hall/CRC.

Gramacy, R. B., & Apley, D. W. (2015). Local Gaussian process approximation for large computer experiments. *Journal of Computational and Graphical Statistics*, *24*(2), 561–578.

Gramacy, R. B., & Lee, H. K. H. (2008). Bayesian treed Gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, *103*(483), 1119–1130.

Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., & Zhang, H. (2011). Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-second international joint conference on artificial intelligence*.

Hensman, J., Fusi, N., & Lawrence, N. D. (2013). Gaussian processes for big data. arXiv preprint arXiv:1309.6835.

Higdon, D., Kennedy, M., Cavendish, J. C., Cafeo, J. A., & Ryne, R. D. (2004). Combining field data and computer simulations for calibration and prediction. *SIAM Journal on Scientific Computing*, *26*(2), 448–466.

Hoffman, M. D., Gelman, A., et al. (2014). The no-u-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *15*(1), 1593–1623.

Holland, J. H. (1992). Genetic algorithms. *Scientific American*, *267*(1), 66–73.

Katzfuss, M., & Guinness, J. (2021). A general framework for vecchia approximations of gaussian processes. *Statistical Science*, *36*(1), 124–141.

Katzfuss, M., Guinness, J., Gong, W., & Zilber, D. (2020). Vecchia approximations of gaussian-process predictions. *Journal of Agricultural, Biological and Environmental Statistics*, *25*, 383–414.

Katzfuss, M., Jurek, M., Zilber, D., & Gong, W. (2020). GPvecchia: Scalable gaussian-process approximations. https://CRAN.R-project.org/package=GPvecchia, R package version 0.1.3.

Kennedy, M. C., & O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *63*(3), 425–464.

Lim, Y. B., Sacks, J., Studden, W. J., & Welch, W. J. (2002). Design and analysis of computer experiments when the output is highly correlated over the input space. *Canadian Journal of Statistics*, *30*(1), 109–126.

Liu, H., Ong, Y.-S., Shen, X., & Cai, J. (2020). When Gaussian process meets big data: A review of scalable GPS. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(11), 4405–4423.

Moran, K. R., & Wheeler, M. W. (2020). Fast increased fidelity approximate gibbs samplers for bayesian gaussian process regression. arXiv preprint arXiv:2006.06537.

Nguyen-Tuong, D., Peters, J., & Seeger, M. (2008). Local gaussian process regression for real time online model learning. Advances in neural information processing systems, 21.

Park, J.-S. (1994). Optimal latin-hypercube designs for computer experiments. *Journal of Statistical Planning and Inference*, *39*(1), 95–111.

Procopiuc, O., Agarwal, P. K., Arge, L., & Vitter, J. S. (2003). Bkd-tree: A dynamic scalable kd-tree. In *International symposium on spatial and temporal databases*, Springer, pp. 46–65.

Ranjan, P., Bingham, D., & Michailidis, G. (2008). Sequential experiment design for contour estimation from complex computer codes. *Technometrics*, *50*(4), 527–541.

Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer school on machine learning*, Springer, pp. 63–71.

Rumsey, K., Grosskopf, M., Lawrence, E., Biswas, A., & Urban, N. (2022). A hierarchical sparse gaussian process for in situ inference in expensive physics simulations. In *Applications of machine learning 2022*, 12227, SPIE, pp. 126–138.

Rumsey, K. N., & Huerta, G. (2021). Fast matrix algebra for bayesian model calibration. *Journal of Statistical Computation and Simulation*, *91*(7), 1331–1341.

Sauer, A., Cooper, A., & Gramacy, R. B. (2022). Vecchia-approximated deep gaussian processes for computer experiments. *Journal of Computational and Graphical Statistics*, *2022*, 1–14.

Schubert, E., & Rousseeuw, P. J. (2019). Faster k-Medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms. In *International conference on similarity search and applications*, Springer, pp. 171–187.

Snelson, E., & Ghahramani, Z. (2005). Sparse gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, *18*.
Solin, A., & Särkkä, S. (2020). Hilbert space methods for reduced-rank gaussian process regression. *Statistics and Computing*, *30*(2), 419–446.
Sparapani, R., Spanbauer, C., & McCulloch, R. (2021). Nonparametric machine learning and efficient computation with bayesian additive regression trees: The Bart R package. *Journal of Statistical Software*, *97*, 1–66.
Stein, M. L., Chi, Z., & Welty, L. J. (2004). Approximating likelihoods for large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *66*(2), 275–296.
Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, PMLR, pp. 567–574.
Vecchia, A. V. (1988). Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, *50*(2), 297–312.
Zacks, S. (1998). *Modern industrial statistics: Design and control of quality and reliability*: Cengage Learning.

How to cite this article: Rumsey, K. N., Huerta, G., & Derek Tucker, J. (2023). A localized ensemble of approximate Gaussian processes for fast sequential emulation. *Stat*, *12*(1), e576. https://doi.org/10.1002/sta4.576