Faster Kernel Interpolation for Gaussian Processes

Mohit Yadav, Daniel Sheldon, Cameron Musco
{ymohit, sheldon, cmusco}@cs.umass.edu
University of Massachusetts Amherst

Abstract

A key challenge in scaling Gaussian Process (GP) regression to massive datasets is that exact inference requires computation with a dense $n \times n$ kernel matrix, where n is the number of data points. Significant work focuses on approximating the kernel matrix via interpolation using a smaller set of m "inducing points". *Structured kernel interpolation* (SKI) is among the most scalable methods: by placing inducing points on a dense grid and using structured matrix algebra, SKI achieves per-iteration time of $\mathcal{O}(n + m \log m)$ for approximate inference. This linear scaling in n enables inference for very large data sets; however the cost is *per-iteration*, which remains a limitation for extremely large n. We show that the SKI per-iteration time can be reduced to $\mathcal{O}(m \log m)$ after a single $\mathcal{O}(n)$ time precomputation step by reframing SKI as solving a natural Bayesian linear regression problem with a fixed set of m compact basis functions. With per-iteration complexity *independent of the dataset size* n for a fixed grid, our method scales to truly massive data sets. We demonstrate speedups in practice for a wide range of m and n and apply the method to GP inference on a three-dimensional weather radar dataset with over 100 million points.

1 Introduction

GPs are a widely used and principled class of methods for predictive modeling. They have a long history in spatial statistics and geostatistics for spatio-temporal interpolation problems (Matheron, 1973; Cressie and Johannesson, 2008). They were later adopted in ML as general-purpose predictive models (Rasmussen, 2004), motivated in part by connections to neural networks (Neal, 1995; Williams, 1997). More recently, similar connections have been identified between GPs and deep networks (Lee et al., 2018; Matthews et al., 2018; Garriga-Alonso et al., 2019; Novak et al., 2019; Cheng et al., 2019). GPs can be used for general-purpose Bayesian regression (Williams and Rasmussen, 1996; Rasmussen, 1999), classification (Williams and Barber, 1998), and many other applications (Rasmussen, 2004; Wilson and Adams, 2013).

A well-known limitation of GPs is running-time scalability. The basic inference and learning tasks require linear algebraic operations (e.g., matrix inversion, linear solves, computing log-determinants) with an $n \times n$ kernel matrix, where n is the number of data points. Exact computations require $\Theta(n^3)$ time — e.g., using the Cholesky decomposition — which severely limits applicability to large problems. Hence, a large amount of work has been devoted to improving scalability of GP inference and learning through approximation. Most of this work is based on the idea of forming an approximate kernel matrix that includes low-rank structure, e.g., through the use of inducing points or random features (Williams and Seeger, 2001; Snelson and Ghahramani, 2005; Quiñonero-Candela and Rasmussen, 2005; Rahimi and Recht, 2007; Le et al., 2013). With rank-m structure, the running time of key tasks can be reduced to $\Theta(nm^2 + m^3)$ (Quiñonero-Candela and Rasmussen, 2005). However, this scaling with n and m continues to limit the size of input data sets that can be handled, the approximation accuracy, or both.

Structured kernel interpolation (SKI) is a promising approach to further improve the scalability of GP methods on relatively low-dimensional data (Wilson and Nickisch, 2015). In SKI, m inducing points are placed on a regular grid, which, when combined with any stationary kernel covariance function (e.g., RBF kernel), imposes extra structure in the approximate kernel matrix. Kernel matrix operations on the grid (i.e., multiplying with a vector) require only $\mathcal{O}(m \log m)$ time, and interpo-

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

lating from the grid requires $\mathcal{O}(n)$ time. By combining structured kernel operations with iterative methods for numerical linear algebra, the running time to solve core GP tasks becomes $\mathcal{O}(k(n + m \log m))$ where k is the number of iterations, and is usually much less than m or n. The modest per-iteration runtime of $\mathcal{O}(n + m \log m)$ allows the modeler to select a large number of inducing points.

We show how to further improve the scalability of SKI with respect to n and scale to truly massive data sets. We first show that the SKI approximation corresponds to exact inference in a Bayesian regression problem with a fixed set of m compact spatial basis functions. This lets us reduce the per-iteration runtime to $\mathcal{O}(m \log m)$ — completely *independent* of n — after a one-time preprocessing cost of $\mathcal{O}(n)$ to compute sufficient statistics of the regression problem. However, naive application of these ideas introduces undesirable trade-offs: while the per-iteration cost is better, we must solve linear systems that are computationally less desirable than the original ones — e.g., they are asymmetric instead of symmetric, or have worse condition number. To avoid these trade-offs, we contribute novel "factorized" conjugate gradient and Lanczos methods, which allow us to solve the *original* linear systems in $\mathcal{O}(m \log m)$ time per-iteration instead of $\mathcal{O}(n + m \log m)$.

Our techniques accelerate SKI inference and learning across a wide range of settings. They apply to each of the main sub-tasks of GP inference: computing the posterior mean, posterior covariance, and log-likelihood. We demonstrate runtime improvements across different data sets and grid sizes, and the ability to scale GP inference to datasets well outside the typical range that can be handled by SKI or other approaches, such as inducing point methods. For example, we demonstrate the ability to perform GP inference on a three-dimensional weather radar dataset with 120 million data points, using a grid of $128,000$ inducing points.

1.1 Related Work

Outside of SKI and its variants (Wilson et al., 2015; Gardner et al., 2018a), a variety of scalable GP methods have been proposed. Most notable are inducing point methods (sparse GP approximations), such as the Nyström, SoRs, FITC, and SMGP methods (Williams and Seeger, 2001; Snelson and Ghahramani, 2005; Quiñonero-Candela and Rasmussen, 2005; Walder et al., 2008). These methods require either $\Omega(nm^2)$ time for direct solves, or $\Omega(nm)$ per-iteration cost if using iterative methods. Our approach significantly improves the dependence on both n and m to just $\mathcal{O}(n)$ preprocessing time and $\mathcal{O}(m \log m)$ per-iteration cost.

While inducing point methods generally cannot leverage structured matrix methods like SKI, especially in higher dimensions, they may achieve comparable accuracy with a smaller number of inducing points. Directly comparing SKI to popular inducing point methods is beyond the scope of this work, however prior work shows significant performance gains on large, relatively low-dimensional datasets (Wilson and Nickisch, 2015; Dong et al., 2017). We note that very recent work (Meanti et al., 2020) seeks to push the limits of inducing point methods via careful systems and hardware level implementations. Like our work, they achieve scaling to datasets with over 100 million points.

Many scalable GP methods have also been proposed in the geostatistics literature – see (Heaton et al., 2019) for a survey. These include structured methods when observations lie on a grid (Guinness and Fuentes, 2017; Stroud et al., 2017). Most closely related to our work is *fixed rank kriging* (FRK) (Cressie and Johannesson, 2008), which can be viewed as a generalization of our Bayesian regression interpretation of SKI. Like inducing point methods, FRK using m basis functions requires $\Omega(nm^2)$ time. We show that SKI can also be viewed as a special case of FRK with a fixed kernel function and a particular set of basis functions arising from interpolation. These two choices allow our faster runtime, through the application of structured matrix vector multiplication techniques and factorized iterative methods, which in turn significantly increases the number of basis functions that can be used.

2 Background

Notation: Throughout we use bold letters to represent vectors and capitals to represent matrices. $I \in \mathbb{R}^{n \times n}$ represents the identity matrix, with dimension apparent from context. For $M \in \mathbb{R}^{p \times r}$, $mv(M)$ denotes the time required to multiply M by any vector in $\mathbb{R}^r$.

2.1 Gaussian Process Regression

In GP regression (Rasmussen, 2004), response values are modeled as noisy measurements of a random function f on input data points. Let $\mathcal{D}$ be a set of n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ with corresponding response values $y_1, \ldots, y_n \in \mathbb{R}$. Let $\mathbf{y} \in \mathbb{R}^n$ have its i^{th} entry equal to y_i and $X \in \mathbb{R}^{n \times d}$ have its i^{th} row equal to $\mathbf{x}_i$. A Gaussian process with kernel (covariance) function $k(\mathbf{x}, \mathbf{x}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a random function f such, for any $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$:

$$\mathbf{f} = [f(\mathbf{x}_1), ..., f(\mathbf{x}_n)] \sim \mathcal{N}(0, K_X), \tag{1}$$

where $K_X = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n \in \mathbb{R}^{n \times n}$ is the kernel (covariance) matrix on the data points X.

We assume without loss of generality that f is zero-

mean. The responses $\mathbf{y}$ are modeled as measurements of $\mathbf{f}_X$ with i.i.d. Gaussian noise, i.e., $\mathbf{y} \sim \mathcal{N}(\mathbf{f}, \sigma^2 I)$.

The posterior distribution of f given the data $\mathcal{D} = (X, \mathbf{y})$ is itself a Gaussian process. The standard GP inference tasks are to compute the posterior mean and covariance and the log-likelihood, given in Fact 1.

Fact 1 (Exact GP Inference). *The posterior mean, covariance, and log likelihood for Gaussian process regression are given by:*

mean: $\mu_{f|\mathcal{D}}(\mathbf{x}) = \mathbf{k}_{\mathbf{x}}^T \mathbf{z}$

covariance: $k_{f|\mathcal{D}}(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - \mathbf{k}_{\mathbf{x}}^T (K_X + \sigma^2 I)^{-1} \mathbf{k}_{\mathbf{x}'}$

log likelihood: $\log p(\mathbf{y}) = -\frac{1}{2}[\log\det(K_X + \sigma^2 I) + \mathbf{y}^T \mathbf{z} + n \log(2\pi)]$

where $\mathbf{k}_{\mathbf{x}} \in \mathbb{R}^n$ *has* i^{th} *entry* $k(\mathbf{x}, \mathbf{x}_i)$ *and* $\mathbf{z} = (K_X + \sigma^2 I)^{-1} \mathbf{y}$.

Evaluating the posterior mean $\mu_{f|\mathcal{D}}(\mathbf{x})$, covariance $k_{f|\mathcal{D}}(\mathbf{x}, \mathbf{x}')$, and log likelihood require matrix-vector multiplication (MVM) with $(K_X + \sigma^2 I)^{-1}$, which is the major run-time bottleneck for GP inference. Computing this inverse directly requires $\Theta(n^3)$ time. The log likelihood requires a further computation of $\log\det(K_X + \sigma^2 I)$, which again naively takes $\Theta(n^3)$ time using the Cholesky decomposition. Computing its gradient, which is necessary, e.g., in hyper-parameter tuning with gradient based methods, requires a further trace computation involving $(K_X + \sigma^2 I)^{-1}$.

Inference Via Iterative Methods. One way to accelerate GP inference is to avoid full matrix factorization like Cholesky decomposition and instead use iterative methods. Gardner et al. (2018b) detail how to compute or approximate each term in Fact 1 using a modified version of the conjugate gradient (CG).

For example, the vector $\mathbf{z} = (K_X + \sigma^2 I)^{-1} \mathbf{y} \in \mathbb{R}^n$ is computed by using CG to solve $(K_X + \sigma^2 I)\mathbf{z} = \mathbf{y}$, which yields the posterior mean as $\mu_{f|\mathcal{D}}(\mathbf{x}) = \mathbf{k}_{\mathbf{x}}^T \mathbf{z}$, and is also used in the calculation of the log-likelihood.

We will use two iterative algorithms in our work: (1) conjugate gradient to solve linear systems $A\mathbf{v} = \mathbf{b}$ for symmetric positive definite (SPD) A, (2) the Lanczos algorithm to (sometimes partially) *tridiagonalize* an SPD matrix $A \in \mathbb{R}^{p \times p}$ as $A = QTQ^T$ where $Q \in \mathbb{R}^{p \times p}$ is orthonormal and $T \in \mathbb{R}^{p \times p}$ is tridiagonal. Tridiagonalization is used to approximate $\log\det(\cdot)$ terms (Dong et al., 2017; Ubaru et al., 2017) and to compute a low-rank factorization for approximate posterior covariance evaluation (Pleiss et al., 2018).

Each iteration of CG or Lanczos for GP inference requires matrix-vector multiplication with $A = K_X + \sigma^2 I$, which in general takes $mv(K_X) + n = \mathcal{O}(n^2)$ time, but may be faster if K_X has special structure. The number of iterations required to reach a given error tolerance depends on the eigenspectrum of A, and is usually much less than n, often mostly between 50 to 100. It can be even lower with preconditioning (Gardner et al., 2018b). Recent works thus often informally considers the number of iterations to be a constant independent of n (Gardner et al., 2018b,a).

2.2 Structured Kernel Interpolation

For large n, the $\Theta(n^2)$ per-iteration cost of iterative methods is still prohibitively expensive. Structured kernel interpolation (SKI) is a method to accelerate MVMs by using an approximate kernel matrix with a special structure (Wilson and Nickisch, 2015). SKI approximates the kernel $k(\mathbf{x}, \mathbf{x}')$ as:

$$\tilde{k}(\mathbf{x}, \mathbf{x}') = \mathbf{w}_{\mathbf{x}}^T K_G \mathbf{w}_{\mathbf{x}'} \tag{2}$$

where $K_G \in \mathbb{R}^{m \times m}$ is the kernel matrix for the set of m points on a dense d-dimensional grid, and the vector $\mathbf{w}_{\mathbf{x}} \in \mathbb{R}^m$ contains interpolation weights to interpolate from grid points to arbitrary $\mathbf{x} \in \mathbb{R}^d$. That is, the kernel values between any two points is approximated by interpolating kernel inner products among grid points.

SKI can use any interpolation strategy (e.g., linear or cubic); typically, the strategy is local, so that $\mathbf{w}_{\mathbf{x}}$ has only a constant number of non-zero entries corresponding to the grid points closest to $\mathbf{x}$. E.g., for linear interpolation, $\mathbf{w}_x$ has 2^d non-zeros. Let $W \in \mathbb{R}^{n \times m}$ have i^{th} row equal to $\mathbf{w}_{\mathbf{x}_i}$. SKI approximates the true kernel matrix K_X as $\tilde{K}_X = W K_G W^T$. Plugging this approximation directly into the GP inference equations of Fact 1 yields the SKI inference scheme in Def. 1.

Definition 1 (SKI Inference). *The SKI approximate inference equations are given by:*

mean: $\mu_{f|\mathcal{D}}(\mathbf{x}) \approx \mathbf{w}_{\mathbf{x}}^T K_G W^T \tilde{\mathbf{z}}$

covariance: $k_{f|\mathcal{D}}(\mathbf{x}, \mathbf{x}') \approx \mathbf{w}_{\mathbf{x}}^T K_G \mathbf{w}_{\mathbf{x}'} - \mathbf{w}_{\mathbf{x}}^T K_G W^T (\tilde{K}_X + \sigma^2 I)^{-1} W K_G \mathbf{w}_{\mathbf{x}'}$

log likelihood: $\log p(\mathbf{y}) \approx -\frac{1}{2}[\log\det(\tilde{K}_X + \sigma^2 I) + \mathbf{y}^T \tilde{\mathbf{z}} + n \log(2\pi)]$

where $\tilde{K}_X = W K_G W^T$ *and* $\tilde{\mathbf{z}} = \left(\tilde{K}_X + \sigma^2 I\right)^{-1} \mathbf{y}$.

SKI Running Time and Memory. The SKI method admits efficient approximate inference due to:

(1) the sparsity of the interpolation weight matrix W, and (2) the structure of the on-grid kernel matrix K_G.

The cost for each iteration of CG or Lanczos is $\mathcal{O}(mv(\tilde{K}_X + \sigma^2 I)) = \mathcal{O}(mv(W) + mv(K_G) + n)$. This runtime is $\mathcal{O}(n + m \log m)$ per iteration assuming: (1) W has $\mathcal{O}(1)$ entries per row and so $mv(W) = \mathcal{O}(n)$, and (2) K_G is multilevel Toeplitz, so $mv(K_G) = \mathcal{O}(m \log m)$ via fast Fourier transform (Lee, 1986). The matrix K_G is multilevel Toeplitz (also known as block Toeplitz with Toeplitz blocks or BTTB) whenever G is an equally-spaced grid and $k(\cdot, \cdot)$ is stationary. The memory footprint is roughly $\texttt{nnz}(W)+m+n = \mathcal{O}(n+m)$ to store W, K_G, and $\mathbf{y}$, where $\texttt{nnz}(A)$ denotes the number of non zeros of A.

Overall, SKI improves significantly on the naive $\mathcal{O}(n^2)$ per-iteration cost required to apply the true kernel matrix K_X, to per-iteration runtime of $\mathcal{O}(n+m \log m)$. However, when n is very large, the $\mathcal{O}(n)$ term (for both runtime and memory) can become a bottleneck. Our main contribution is to remove this cost, giving methods with $\mathcal{O}(m \log m)$ per-iteration runtime with $\mathcal{O}(m)$ memory after $\mathcal{O}(n)$ preprocessing.

3 SKI as Bayesian Linear Regression with Fixed Basis Functions

Our first contribution is to reformulate SKI as *exact inference* in a Bayesian linear regression problem with compact basis functions associated with grid points. This lets us use standard conjugate update formulas for Bayesian linear regression to reduce SKI's per-iteration runtime to $\mathcal{O}(m \log m)$, with $\mathcal{O}(n)$ preprocessing.

Definition 2 (**Grid-Structured Gaussian Process; GSGP**). *Let $G = \{\mathbf{g}_1, \ldots, \mathbf{g}_m\} \subseteq \mathbb{R}^d$ be a set of grid points and $k(\mathbf{x}, \mathbf{x}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a positive-definite kernel function. A* grid-structured Gaussian process *f is defined by the following generative process:*

$$\boldsymbol{\theta} \sim \mathcal{N}(0, K_G),$$
$$f(\mathbf{x}) = \mathbf{w}_{\mathbf{x}}^T \boldsymbol{\theta}, \quad \forall \mathbf{x} \in \mathbb{R}^d.$$

where $\mathbf{w}$ maps $\mathbf{x} \in \mathbb{R}^d$ to $\mathbf{w}_{\mathbf{x}} \in \mathbb{R}^m$.

Notice that a GSGP is a classical Bayesian linear regression model. In principle, $\mathbf{w}$ can be any mapping from $\mathbb{R}^d$ to $\mathbb{R}^m$. However, for computational efficiency and to match the notion of interpolation, the vector $\mathbf{w}_{\mathbf{x}}$ will be taken to be the set of weights used by any fixed scheme to interpolate values from grid points to arbitrary locations $\mathbf{x} \in \mathbb{R}^d$. The generative process is illustrated in Figure 1 for $d = 1$ and cubic interpolation on an integer grid (Keys, 1981). The basis functions

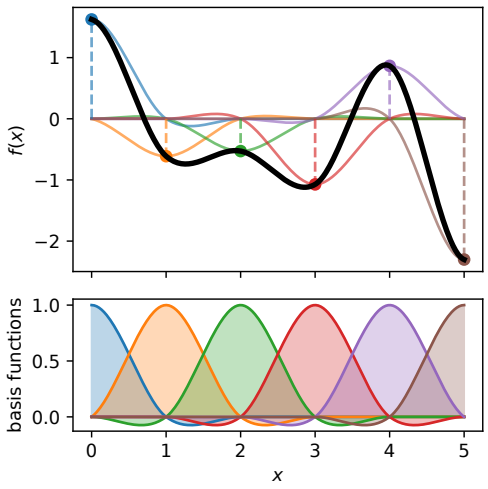

Figure 1: GSGP illustration. Bottom: basis functions for cubic interpolation are compact and centered at grid points. Top: a GSGP (thick black curve) is formed as the sum of scaled basis functions (lighter colored curves) with random weights at grid points (vertical dashed lines) drawn from the original GP.

$\mathbf{w}_{\mathbf{x}}^j$ for each grid point j and for all $\mathbf{x}$ are shown in the bottom panel. The jth basis function is centered at j and supported on $[j-2, j+2)$. For any fixed $\mathbf{x}$, the vector $\mathbf{w}_{\mathbf{x}}$ has at most four nonzero entries corresponding to the four nearest grid points. The GSGP f is generated by first drawing random weights $\boldsymbol{\theta}$ as the values of the *original* GP — with covariance function $k(\cdot, \cdot)$ — at grid points. The generated function $f(\mathbf{x})$ can be interpreted in two ways: (1) a sum of scaled basis functions $f(\mathbf{x}) = \sum_j \boldsymbol{\theta}_j \mathbf{w}_{\mathbf{x}}^j$, (2) the result of interpolating the grid values $\boldsymbol{\theta}$ to $\mathbb{R}^d$ using the interpolation scheme.

Claim 1. *A GSGP with grid weights $\boldsymbol{\theta}$ drawn from a GP with covariance function $k(\mathbf{x}, \mathbf{x}')$ is itself a GP with covariance function $\tilde{k}(\mathbf{x}, \mathbf{x}') = \mathbf{w}_{\mathbf{x}}^T K_G \mathbf{w}_{\mathbf{x}'}$.*

Proof. For any finite number of input points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the joint distribution of $(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))$ is zero-mean Gaussian, since each $f(\mathbf{x}_i) = \mathbf{w}_{\mathbf{x}_i}^T \boldsymbol{\theta}$ is a linear transformation of the same zero-mean Gaussian random variable $\boldsymbol{\theta}$. Therefore, $f(\mathbf{x})$ is a zero-mean GP. For a particular pair of function values $f(\mathbf{x})$ and $f(\mathbf{x}')$, the covariance is given by

$$\begin{aligned} \operatorname{Cov}(f(\mathbf{x}), f(\mathbf{x}')) &= \operatorname{Cov}(\mathbf{w}_{\mathbf{x}}^T \boldsymbol{\theta}, \mathbf{w}_{\mathbf{x}'}^T \boldsymbol{\theta}) \\ &= \mathbf{w}_{\mathbf{x}}^T \operatorname{Cov}(\boldsymbol{\theta}, \boldsymbol{\theta}) \mathbf{w}_{\mathbf{x}'} \\ &= \mathbf{w}_{\mathbf{x}}^T K_G \mathbf{w}_{\mathbf{x}'} = \tilde{k}(\mathbf{x}, \mathbf{x}'). \end{aligned} \tag{3}$$

Therefore, f has the claimed covariance function. □

In other words: the *exact* covariance function of the

Fact 2 (**GSGP Inference**). *The posterior mean, covariance, and log likelihood functions for the grid-structured Gaussian process (Def. 2) are given by:*

$$\textit{mean: } \mu_{f|\mathcal{D}}(\mathbf{x}) = \mathbf{w}_{\mathbf{x}}^T \bar{\mathbf{z}}$$

$$\textit{covariance: } k_{f|\mathcal{D}}(\mathbf{x}, \mathbf{x}') = \mathbf{w}_{\mathbf{x}}^T \bar{\mathbf{C}} \mathbf{w}_{\mathbf{x}'}$$

$$\textit{log likelihood: } \log p(\mathbf{y}) = -\frac{1}{2}\Big[\log\det(K_G W^T W + \sigma^2 I) + \frac{\mathbf{y}^T(\mathbf{y} - W\bar{\mathbf{z}})}{\sigma^2} + n\log(2\pi) + (n-m)\log\sigma^2\Big],$$

where $\bar{\mathbf{z}} = \mathbb{E}[\boldsymbol{\theta}|\mathbf{y}] = (K_G W^T W + \sigma^2 I)^{-1} K_G W^T \mathbf{y}$ *is the posterior mean of* $\boldsymbol{\theta}$ *and* $\bar{\mathbf{C}} = \mathrm{Var}(\boldsymbol{\theta}|\mathbf{y}) = \sigma^2 \left(K_G W^T W + \sigma^2 I\right)^{-1} K_G$ *is the posterior variance of* $\boldsymbol{\theta}$.

GSGP is the same as the SKI approximation in Eq. (2). Now, suppose noisy observations $\mathbf{y} \sim \mathcal{N}(\mathbf{f}_X, \sigma^2)$ are made of the GSGP at input locations X. It is well known that the posterior distribution of f is a Gaussian process, with mean, covariance and log likelihood given in Fact 2 (Neal, 2011; Bishop, 2006) and the proof of the same is presented in Appendix A.1. From Claim 1 it follows that:

Theorem 1 (Equivalence of GSGP Inference and SKI Approximation)**.** *The inference expressions of Fact 2 are identical to the SKI approximations of Def. 1.*

For illustrative purposes, we provide a purely linear algebraic proof of Theorem 1 in Appendix A.2.

3.1 GSGP Running Time and Memory

By Theorem 1, we can apply the SKI approximation using the GSGP inference equations of Fact 2; these also involve structured matrices that are well suited to iterative methods. In particular, they require linear solves and logdet computation for the $m \times m$ matrix $K_G W^T W + \sigma^2 I$ rather than the $n \times n$ matrix $W K_G W^T + \sigma^2 I$. Under the standard SKI assumptions, this leads to $\mathcal{O}(m \log m)$ per-iteration run time and $\mathcal{O}(m)$ memory footprint with $\mathcal{O}(n)$ precomputation.

Precomputation: GSGP involves precomputing $W^T W \in \mathbb{R}^{m \times m}$, $W^T \mathbf{y} \in \mathbb{R}^m$, and $\mathbf{y}^T \mathbf{y} \in \mathbb{R}$, which are the sufficient statistics of a linear regression problem with feature matrix W. Each is a sum over n data points and has fixed size depending only on m. Once computed, each expression in Fact 2 can be computed without referring back to the data W and $\mathbf{y}$. It is clear that $\mathbf{y}^T \mathbf{y} = \sum_{i=1}^n y_i^2$ can be computed in $\mathcal{O}(n)$ time with a single pass over the data. Assume the interpolation strategy is local (e.g., linear or cubic interpolation), so that each $\mathbf{w}_{\mathbf{x}_i}$ has $\mathcal{O}(1)$ non-zeros. Then $W^T \mathbf{y} = \sum_{i=1}^n \mathbf{w}_{\mathbf{x}_i}^T \mathbf{y}$ can also be computed in $\mathcal{O}(n)$ time with one pass over the data, since each inner product accesses only a constant number of entries of $\mathbf{y}$. $W^T W$ also has desirable computational properties:

Claim 2. *Assume that* $G = \{\mathbf{g}_1, \ldots, \mathbf{g}_m\}$ *has spacing* s*, i.e.,* $\|\mathbf{g}_i - \mathbf{g}_j\|_\infty \geq s$ *for any* $i, j \in m$ *and that* $\mathbf{w}_{\mathbf{x}}^j$ *is non-zero only if* $\|\mathbf{g}_j - \mathbf{x}\|_\infty < r \cdot s$ *for some fixed integer* r*. Then* $W^T W$ *can be computed in* $\mathcal{O}(n(2r)^{2d})$ *time and has at most* $(4r-1)^d$ *entries per row. Therefore* $mv(W^T W) = \mathcal{O}(m(4r-1)^d)$.

Proof. First, write $W^T W$ as a sum over data points:

$$W^T W = \sum_{i=1}^n \mathbf{w}_{\mathbf{x}_i} \mathbf{w}_{\mathbf{x}_i}^T.$$

The vector $\mathbf{w}_{\mathbf{x}_i}$ has nonzeros only for grid points $\mathbf{g}_j$ such that $\|\mathbf{g}_j - \mathbf{x}_i\|_\infty \leq r \cdot s$. Since the grid points have spacing s, there are at most $(2r)^d$ such grid points. Therefore, the outer product $\mathbf{w}_{\mathbf{x}_i} \mathbf{w}_{\mathbf{x}_i}^T$ has at most $(2r)^{2d}$ non-zeros, and the sum can be computed in $\mathcal{O}(n(2r)^{2d})$ time. Also note that the sum is also sparse.

The entry $(W^T W)_{jk}$ is non-zero only if there is *some* data point $\mathbf{x}_i$ within distance r from both $\mathbf{g}_j$ and $\mathbf{g}_k$ in each dimension. This is true only if $\|\mathbf{g}_j - \mathbf{g}_k\|_\infty < 2r \cdot s$. For a given grid point $\mathbf{g}_j$, there are at most $(4r-1)^d$ grid points $\mathbf{g}_k$ satisfying $\|\mathbf{g}_j - \mathbf{g}_k\|_\infty < 2r \cdot s$ (e.g., in 1 dimension there are $2r-1$ neighbors to the left, $2r-1$ neighbors to the right, plus the case $k = j$). Therefore the jth row of $W^T W$ has as most $(4r-1)^d$ nonzeros, as claimed. □

For example, $r = 1$ for linear interpolation and $r = 2$ for cubic interpolation. The upshot is that $W^T W$ can be precomputed in $\mathcal{O}(n)$ time, after which matrix-vector multiplications take $\mathcal{O}(m)$ time, with dependence on r and d similar to that of $mv(W) = \mathcal{O}(n(2r)^d)$.

Per-Iteration: As discussed in Section 2.2, due to its grid structure, K_G admits fast matrix-vector multiplication: $mv(K_G) = \mathcal{O}(m \log m)$ for stationary kernels. Since $W^T W$ is sparse, $mv(W^T W) = \mathcal{O}(m)$. Overall, $mv(K_G W^T W + \sigma^2 I) = \mathcal{O}(m \log m)$, giving per-iteration runtime of $\mathcal{O}(m \log m)$ for computing the approximate mean, covariance, and log-likelihood in Fact 2 via iterative methods. Importantly, this complexity is *independent* of the number of data points.

Memory: GSGP uses $\texttt{nnz}(W^T W) + m + m = \mathcal{O}(m)$ memory to store $W^T W$, $W^T \mathbf{y}$ and K_G.

Limitations of GSGP. Directly replacing the classic SKI method with the GSGP inference equations of Fact 2 reduces per-iteration cost but has some undesirable trade-offs. In particular, unlike $W K_G W^T + \sigma^2 I$,

the matrix $K_G W^T W + \sigma^2 I$ is *asymmetric*. Thus, conjugate gradient and Lanczos—which are designed for *symmetric* positive semidefinite matrices—are not applicable. Asymmetric solvers like GMRES (Saad and Schultz, 1986) can be used, and seem to work well in practice for posterior mean estimation, but do not enjoy the same theoretical convergence guarantees, nor do they as readily provide the approximate tridiagonalization for low-rank approximation for predictive covariance (Pleiss et al., 2018) or log-likelihood estimation (Dong et al., 2017). It is possible to algebraically manipulate the GSGP expressions to yield *symmetric* $m \times m$ systems, but these lose the desirable 'regularized form' $A + \sigma^2 I$ and have worse conditioning, leading to more iterations being required in practice (see Appendix A.3).

4 Efficient SKI via Factorized Iterative Methods

In this section we show how to achieve the best of both SKI and the GSGP reformulation: we design 'factorized' versions of the CG and Lanczos methods used in SKI with just $\mathcal{O}(m \log m)$ per-iteration complexity. These methods are mathematically equivalent to the full methods, and so enjoy identical convergence rates, avoiding the complications of the asymmetric solves required by GSGP inference.

4.1 The Factorized Approach

Our approach centers on a simple observation relating matrix-vector multiplication with the SKI kernel approximation $\tilde{K}_X = W K_G W^T + \sigma^2 I$ and the GSGP operator $K_G W^T W + \sigma^2 I$. To apply the SKI equations of Def. 1 via an iterative method, a key step at each iteration is to multiply some iterate $\mathbf{z}_i \in \mathbb{R}^n$ by $\tilde{K}_X$, requiring $\mathcal{O}(n + m \log m)$ time. We avoid this by maintaining a compressed representation of any iterate as $\mathbf{z}_i = W\hat{\mathbf{z}}_i + c_i \mathbf{z}_0$, where $\hat{\mathbf{z}}_i \in \mathbb{R}^m$, $c_i \in \mathbb{R}$ is a scalar coefficient, and $\mathbf{z}_0$ is an initial value. At initialization, $\hat{\mathbf{z}}_0 = \mathbf{0}$ and $c_0 = 1$. Critically, this compressed representation can be updated with multiplication by $\tilde{K}_X$ in just $\mathcal{O}(m \log m)$ time using the following claim:

Claim 3 (Factorized Matrix-Vector Multiplication)**.** *For any $\mathbf{z}_i \in \mathbb{R}^n$ with $\mathbf{z}_i = W\hat{\mathbf{z}}_i + c_i \mathbf{z}_0$,*

$$(W K_G W^T + \sigma^2 I)\mathbf{z}_i = W\hat{\mathbf{z}}_{i+1} + c_{i+1}\mathbf{z}_0,$$

where $\hat{\mathbf{z}}_{i+1} = (K_G W^T W + \sigma^2 I)\hat{\mathbf{z}}_i + c_i K_G W^T \mathbf{z}_0$ and $c_{i+1} = \sigma^2 \cdot c_i$. Call this operation a factorized update *and denote it as $(\hat{\mathbf{z}}_{i+1}, c_{i+1}) = \mathcal{A}(\hat{\mathbf{z}}_i, c_i)$. If the vector $K_G W^T \mathbf{z}_0$ is precomputed in $\mathcal{O}(n + m \log m)$ time, each subsequent factorized update takes $\mathcal{O}(m \log m)$ time.*

The proof of the above claim is given in Appendix B.1.

For algorithms such as CG, we also need to support additions and inner products in the compressed representation. Additions are simple via linearity. Inner products can be computed efficiently as well:

Claim 4 (Factorized Inner Products)**.** *For any $\mathbf{z}_i, \mathbf{y}_i \in \mathbb{R}^n$ with $\mathbf{z}_i = W\hat{\mathbf{z}}_i + c_i \mathbf{z}_0$ and $\mathbf{y}_i = W\hat{\mathbf{y}}_i + d_i \mathbf{y}_0$,*

$$\begin{aligned}\mathbf{z}_i^T \mathbf{y}_i = \hat{\mathbf{z}}_i^T W^T W \hat{\mathbf{y}}_i + d_i \hat{\mathbf{z}}_i^T W^T \mathbf{y}_0 \\ + c_i \hat{\mathbf{y}}_i^T W^T \mathbf{z}_0 + c_i d_i \mathbf{y}_0^T \mathbf{z}_0.\end{aligned}$$

We denote the above operation by $\langle(\hat{\mathbf{z}}_i, c_i), (\hat{\mathbf{y}}_i, d_i)\rangle$. If $W^T \mathbf{z}_0$, $W^T \mathbf{y}_0$, and $\mathbf{y}_0^T \mathbf{z}_0$ are precomputed in $\mathcal{O}(n)$ time, then $\langle(\hat{\mathbf{z}}_i, c_i), (\hat{\mathbf{y}}_i, d_i)\rangle$ can be computed using just one matrix-vector multiplication with $W^T W$ and $\mathcal{O}(m)$ additional time.

Algorithm 1 Conjugate gradient

procedure CG($K_G, W, \mathbf{b}, \sigma, \mathbf{x}_0, \epsilon$)
  $\mathbf{r}_0 = \mathbf{b} - \tilde{K}\mathbf{x}_0$
  $\mathbf{p}_0 = \mathbf{r}_0$
  for $k = 0$ to maxiter **do**
    $\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \tilde{K} \mathbf{p}_k}$
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \cdot \mathbf{p}_k$
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \cdot \tilde{K}\mathbf{p}_k$
    if $\mathbf{r}_{k+1}^T \mathbf{r}_{k+1} \leq \epsilon$ break
    $\beta_k = \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
  return $\mathbf{x}_{k+1}$

Algorithm 2 Factorized conjugate gradient (FCG)

procedure FCG($K_G, W, \mathbf{b}, \sigma, \mathbf{x}_0, \epsilon$)
  $\mathbf{r_0} = \mathbf{b} - \tilde{K}\mathbf{x}_0$, $\hat{\mathbf{r}}_\mathbf{0} = \mathbf{0}$, $c_0^r = 1$
  $\hat{\mathbf{p}}_0 = \mathbf{0}$, $c_0^p = 1$, $\hat{\mathbf{x}}_0 = \mathbf{0}$, $c_0^x = 0$
  for $k = 0$ to maxiter **do**
    $\alpha_k = \frac{\langle(\hat{\mathbf{r}}_k, c_k^r), (\hat{\mathbf{r}}_k, c_k^r)\rangle}{\langle(\hat{\mathbf{p}}_k, c_k^p), \mathcal{A}(\hat{\mathbf{p}}_k, c_k^p)\rangle}$
    $(\hat{\mathbf{x}}_{k+1}, c_{k+1}^x) = (\hat{\mathbf{x}}_k, c_k^x) + \alpha_k \cdot (\hat{\mathbf{p}}_k, c_k^p)$
    $(\hat{\mathbf{r}}_{k+1}, c_{k+1}^r) = (\hat{\mathbf{r}}_k, c_k^r) - \alpha_k \cdot \mathcal{A}(\hat{\mathbf{p}}_k, c_k^p)$
    if $\langle(\hat{\mathbf{r}}_{k+1}, c_{k+1}^r), (\hat{\mathbf{r}}_{k+1}, c_{k+1}^r)\rangle \leq \epsilon$ break
    $\beta_k = \frac{\langle(\hat{\mathbf{r}}_{k+1}, c_{k+1}^r), (\hat{\mathbf{r}}_{k+1}, c_{k+1}^r)\rangle}{\langle(\hat{\mathbf{r}}_k, c_k^r), (\hat{\mathbf{r}}_k, c_k^r)\rangle}$
    $(\hat{\mathbf{p}}_{k+1}, c_{k+1}^p) = (\hat{\mathbf{r}}_{k+1}, c_{k+1}^r) + \beta_k \cdot (\hat{\mathbf{p}}_k, c_k^p)$
  return $\mathbf{x}_{k+1} = W\hat{\mathbf{x}}_{k+1} + c_{k+1}^x \cdot \mathbf{r}_0 + \mathbf{x}_0$

Figure 2: Above $\tilde{K} = W K_G W^T + \sigma^2 I$. $\mathcal{A}(\cdot)$ and $\langle \cdot, \cdot \rangle$ denote the factorized matrix-vector multiplication and inner product updates of Claims 3 and 4. The vector $\mathbf{x}_0 \in \mathbb{R}^n$ is an initial solution, the scalar $\epsilon > 0$ is a tolerance parameter, and maxiter is the maximum number of iterations.

4.2 Factorized Conjugate Gradient

We now give an example of this approach by deriving a "factorized conjugate gradient" algorithm. Factorized

CG has lower per-iteration complexity than standard CG for computing the posterior mean, covariance, and log likelihood in the SKI approximations of Def. 1. In Appendix B, we apply the same approach to the Lanczos method, which can be used in approximating the logdet term in the log likelihood, and for computing a low-rank approximation of the posterior covariance.

Figure 2 shows both the classic CG method and our factorized variant. CG maintains three iterates, the current solution estimate $\mathbf{x}_k$, the residual $\mathbf{r}_k$, and a search direction $\mathbf{p}_k$. We maintain each in a compressed form with $\mathbf{x}_k = W\hat{\mathbf{x}}_k + c_k^x \mathbf{r}_0 + \mathbf{x}_0$, $\mathbf{r}_k = W\hat{\mathbf{r}}_k + c_k^r \mathbf{r}_0$, and $\mathbf{p}_k = W\hat{\mathbf{p}}_k + c_k^p \mathbf{r}_0$. Note that the initialization term $\mathbf{r}_0$ is shared across all iterates with a different coefficient, and that $\mathbf{x}_k$ has an additional fixed component $\mathbf{x}_0$. This is an initial solution guess for the system solve, frequently zero. With these invariants, simply applying Claims 3 and 4 gives the factorized algorithm.

Proposition 1 (Factorized CG Equivalence and Runtime)**.** *The outputs of Algos. 1 and 2 on the same inputs are identical. Alg. 2 performs two matrix-vector multiplications with K_G and three with W initially. In each iteration, it performs a constant number of multiplications with K_G and W^TW plus $\mathcal{O}(m)$ additional work. If W^TW is sparse and K_G has multilevel Toeplitz structure, its per iteration runtime is $\mathcal{O}(m \log m)$.*

Appendix B presents a further optimization to only require one matrix-vector multiplication with K_G and one with W^TW per iteration. A similar optimization applies to the factorized Lanczos method.

5 Experiments

We conduct experiments to evaluate our "GSGP approach" of using factorized algorithms to solve SKI inference tasks. We use: (1) factorized CG to solve the linear systems for the SKI posterior mean and covariance expressions of Def. 1 and for approximate tridiagonalization within stochastic log-likelihood approximations (Gardner et al., 2018b), and (2) factorized Lanczos for low-rank approximations of the SKI predictive covariance (Pleiss et al., 2018).

Our goals are to: (1) evaluate the running-time improvements of GSGP, (2) examine new speed-accuracy tradeoffs enabled by GSGP, and (3) demonstrate the ability to push the frontier of GP inference for massive data sets. We use a synthetic data set and three real data sets from prior work (Dong et al., 2017; Wilson and Nickisch, 2015; Angell and Sheldon, 2018), summarized in Table 1. We focus on large, relatively low-dimensional datasets – the regime targeted by structured kernel interpolation methods. The Radar dataset is a subset of a larger 120M point dataset. While SKI cannot scale to the full data size without significant computational resources, at the end of the section we demonstrate that GSGP's ability to scale to this regime with modest runtime and memory usage.

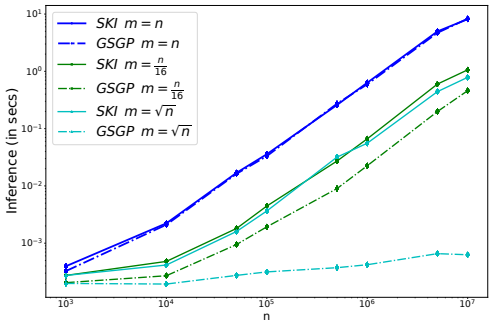

Figure 3: Per-iteration time taken by the SKI and GSGP methods on a synthetic dataset for posterior mean approximation. We see significant speedups when the grid size m is relatively small compared to n. Even when m is larger, e.g., $m = \Theta(n)$, GSGP performs no worse than and sometimes improves upon SKI's runtime, e.g., when $m = n/16$.

Dataset	n	d	m	Time	Memory
Sound	59.3K	1	8K	0.433	0.247
	59.3K	1	60K	0.941	0.505
Radar	10.5M	3	51.2K	0.014	0.007
	10.5M	3	6.4M	0.584	0.425
Precipitation	528K	3	128K	0.326	0.366
	528K	3	528K	0.491	0.628
	528K	3	1.2M	0.806	2.941

Table 1: Ratios of GSGP to SKI per-iteration time and memory usage for posterior mean approximation for different values of m and n. GSGP shows large improvements in a range of settings, even with very large grid size m.

All linear solves use a tolerance of 0.01 and all kernels are squared exponential. We utilize cubic interpolation for all experiments and provide details on hardware and hyperparameters in Appendix C.1. In all cases, error bars show 95% confidence intervals of mean running time over independent trials.

Per-iteration resource usage. We first compare the per-iteration runtime for posterior mean calculation using CG and Factorized CG on synthetic data of varying sizes. The function $f(x)$ is a sine wave with two periods in the interval $[0, 1]$. Random x locations are sampled in the interval and $y = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.25)$; grid points are equally spaced on $[0, 1]$. Figure 3 shows the average per-iteration inference time over all iterations of 8 independent trials for increasing n and three different settings of grid size: $m = n$, $m = n/16$ and $m = \sqrt{n}$. GSGP is substantially faster when $m < n$ (note log scale) and no slower when $m = n$.

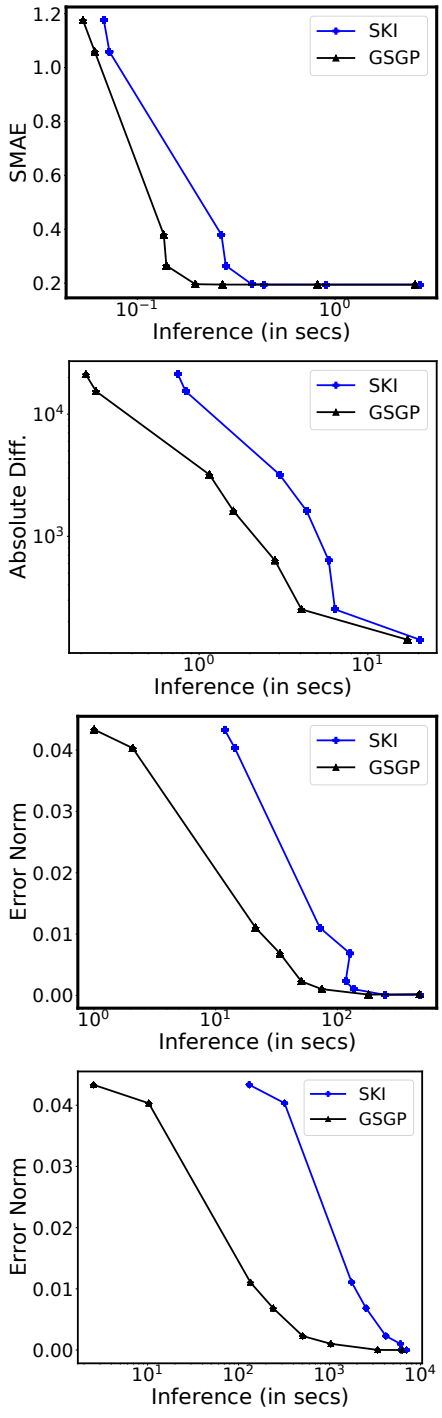

Figure 4: Error vs. runtime for approximate inference tasks on the sound dataset with varying grid size m. GSGP gives much faster runtimes for fixed m, allowing one to use a larger grid and achieve better error-runtime tradeoffs than SKI. Times are averaged over 20 trials and include pre-processing. See text for description of error metrics.

Memory usage is another important consideration. Table 1 compares both per-iteration running time and memory usage for posterior mean inference on our real data sets. Per-iteration time is averaged over one run of CG and Factorized CG for each setting; memory usage is calculated as $\texttt{nnz}(W) + m + n$ for SKI and $\texttt{nnz}(W^TW) + 2m$ for GSGP, where $\texttt{nnz}(A)$ is the number of nonzero entries of A. The gains are significant, especially when $m \ll n$, and gains in time and/or memory are possible even when m equals or exceeds n (e.g., precipitation, $m \in \{528\text{K}, 1.2\text{M}\}$); for very large m it is more resource efficient to run the original algorithm (or use Factorized CG without precomputing W^TW).

Inference accuracy vs. time. A significant advantage of GSGP is the ability to realize error-runtime tradeoffs that were not previously possible. Figure 4 illustrates this for the sound data set ($n = 59.3\text{K}$) by comparing error vs. running time for four different GP inference tasks for grid sizes $m \in \{1\text{K}, 2\text{K}, 5\text{K}, 6\text{K}, 8\text{K}, 10\text{K}, 30\text{K}, 60\text{K}\}$. For mean estimation we compute SMAE (mean absolute error normalized by the mean of observations) on a held-out test set of 691 points. For other tasks (log-likelihood, covariance) we compute error relative to a reference value computed with SKI for the highest m using absolute difference for log-likelihood and Frobenius norm from the reference value for covariance matrices. For log-likelihood, we use 30 samples and $tol = 0.01$ for stochastic logdet approximation (Dong et al., 2017). For covariance, we compute the 691×691 posterior covariance matrix for test points, first using the exact SKI expressions (which requires 691 linear solves) and then using a rank-k approximation (Pleiss et al., 2018), that, once computed, yields $\mathcal{O}(k)$ time approximations of posterior covariances, for $k = \min\{m, 10000\}$. For each task, GSGP is faster when $m < n$, sometimes substantially so, and achieves the same accuracy, leading to strictly better error-runtime tradeoffs.

Very large n. Figure 5 shows running time vs. m for GP inference tasks on a data set of $n = 10.5\text{M}$ radar reflectivity measurements in three dimensions from 13 radar stations in the northeast US (Angell and Sheldon, 2018). This is a situation where $m \ll n$ is highly relevant: even the *smallest* grid size of $51.2\text{K} = 80 \times 80 \times 8$ is of scientific value for summarizing broad-scale weather and biological activity.

GSGP is much faster, e.g., roughly 150x and 15x faster for $m = 51.2K$ on mean inference and log-likelihood estimation respectively after one-time pre-processing (first panel). Pre-processing is up to 3x slower for GSGP due to the need to compute W^TW. To perform only one mean inference, the overall time of GSGP and SKI *including* pre-processing is similar, which is consistent with the observation that typical solves use

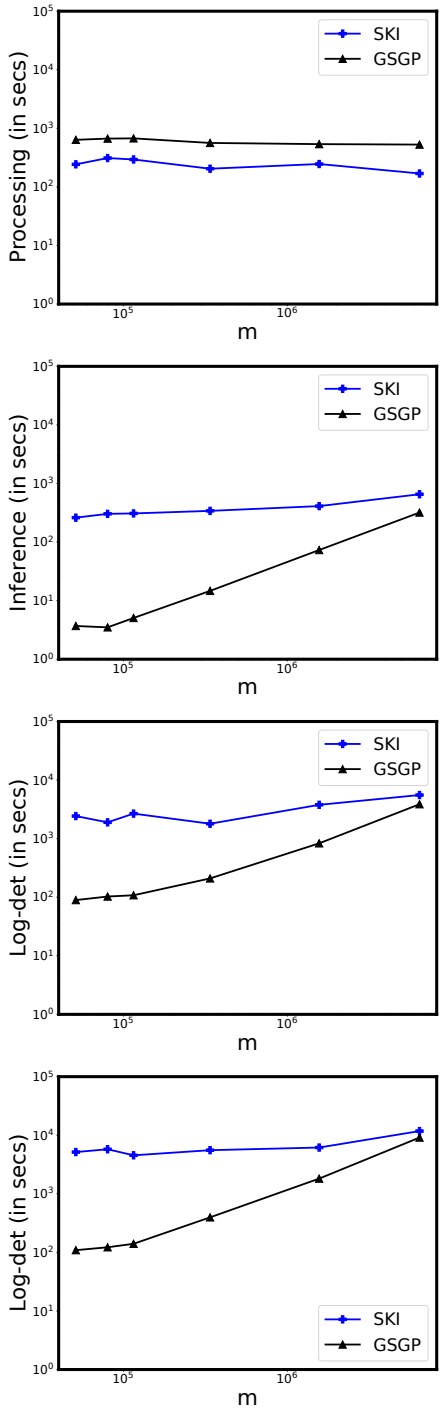

Figure 5: Runtime vs. grid size m for GP inference tasks on the radar data set with 10.5M data points. We observe significantly faster runtimes for GSGP across all tasks and a wide range of grid sizes. From top to bottom: pre-processing time, mean inference runtime, and log-determinant runtime for $tol = 0.1$ and $tol = 0.01$ respectively. 30 random vectors are used in both log-determinant computations.

only tens of iterations, and some of the per-iteration gain is offset by pre-processing. However, scientific modeling is highly iterative, and tasks other than mean inference perform *many* more iterations of linear solvers; the total time for GSGP in these cases is much smaller than SKI. In realistic applications with massive data sets, we expect $W^T W$ to be computed once and saved to disk. GSGP also has the significant advantage that its memory footprint is $\mathcal{O}(m)$, while SKI is $\mathcal{O}(m+n)$.

The data above is a *subset* from a national radar network, which was the limit on which SKI could run without exceeding 10GB of memory. To demonstrate scalability of GSGP, we ran on data from the *entire* national radar network with $n = 120M$ for $m = 128K$, on which SKI far exceeds this memory limit. On this problem, GSGP takes 4861.60 ± 233.42 seconds for pre-processing, and then 9.33 ± 0.31 seconds for mean inference (averaged over 10 trials).

6 Conclusions and Future Work

Our work shows that the SKI method for approximate Gaussian process regression in fact performs exact inference for a natural grid-structured Gaussian process. We leverage this observation to give an implementation for the method with per-iteration complexity *independent of the dataset size* n. This leads to significantly improved performance on a range of problems, including the ability to scale GP inference to radar datasets with over 100 million data points – a regime far beyond what can typically be handled by SKI or other approximation methods, such as inducing point and random feature approaches.

Our work leaves open a number of questions. Algorithmically, it would be interesting to explore if SKI can be efficiently implemented using direct, rather than iterative, solvers that take advantage of the Toeplitz and band-like structures of K_G and $W^T W$ respectively. Theoretically, it would be interesting to further explore the grid-structured Gaussian process for which SKI performs exact inference. Intuitively, by interpolating data points to a grid, this method seems to suppress 'high-frequency' components of the kernel covariance function. Can this be analyzed to lead to formal approximation guarantees or practical guidance in how to choose the grid size?

Acknowledgements

This material is based upon work supported by the National Science Foundation (NSF) under grant numbers 1522054 and 1661259.

References

Georges Matheron. The Intrinsic Random Functions and Their Applications. *Advances in Applied Probability*, 5(3):439–468, 1973.

Noel Cressie and Gardar Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):209–226, 2008.

Carl Edward Rasmussen. Gaussian Processes in Machine Learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer, 2004.

Radford M Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.

Christopher KI Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems 10 (NeurIPS)*, pages 295–301, 1997.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Sam Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep Neural Networks as Gaussian Processes. *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=B1EA-M-0Z`.

Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *International Conference on Learning Representations (ICLR)*, 2018. URL `https://openreview.net/forum?id=H1-nGgWC-`.

Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep Convolutional Networks as Shallow Gaussian Processes. *International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=BklfsiOcKm`.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Jiri Hron, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian Deep Convolutional Networks with Many Channels are Gaussian Processes. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019. URL `https://openreview.net/forum?id=B1g30j0qF7`.

Zezhou Cheng, Matheus Gadelha, Subhransu Maji, and Daniel Sheldon. A bayesian perspective on the deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5443–5451, 2019.

Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian Processes for Regression. In *Advances in Neural Information Processing Systems 9 (NeurIPS)*, pages 514–520, 1996.

Carl Edward Rasmussen. *Evaluation of Gaussian Processes and Other Methods for Non-linear Regression.* University of Toronto, 1999.

Christopher KI Williams and David Barber. Bayesian Classification with Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(12):1342–1351, 1998.

Andrew Wilson and Ryan Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1067–1075, 2013.

Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 14 (NeurIPS)*, pages 682–688, 2001.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18 (NeurIPS)*, pages 1257–1264, 2005.

Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20 (NeurIPS)*, pages 1177–1184, 2007.

Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 85, 2013.

Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1775–1784, 2015.

Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on massively scalable gaussian processes. *arXiv:1511.01870*, 2015.

Jacob R Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q Weinberger, and Andrew Gordon Wilson. Product kernel interpolation for scalable Gaussian processes. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018a.

Christian Walder, Kwang In Kim, and Bernhard Schölkopf. Sparse multiscale Gaussian process regression. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1112–1119, 2008.

Kun Dong, David Eriksson, Hannes Nickisch, David Bindel, and Andrew Gordon Wilson. Scalable log

determinants for Gaussian process kernel learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, pages 6327–6337, 2017.

Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. *arXiv:2006.10350*, 2020.

Matthew J Heaton, Abhirup Datta, Andrew O Finley, Reinhard Furrer, Joseph Guinness, Rajarshi Guhaniyogi, Florian Gerber, Robert B Gramacy, Dorit Hammerling, Matthias Katzfuss, et al. A case study competition among methods for analyzing large spatial data. *Journal of Agricultural, Biological and Environmental Statistics*, 24(3):398–425, 2019.

Joseph Guinness and Montserrat Fuentes. Circulant embedding of approximate covariances for inference from Gaussian data on large lattices. *Journal of Computational and Graphical Statistics*, 26(1):88–97, 2017.

Jonathan R Stroud, Michael L Stein, and Shaun Lysen. Bayesian and maximum likelihood estimation for Gaussian processes on an incomplete lattice. *Journal of computational and Graphical Statistics*, 26(1):108–120, 2017.

Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 7576–7586, 2018b.

Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of $tr(f(a))$ via stochastic Lanczos quadrature. *Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

Geoff Pleiss, Jacob R. Gardner, Kilian Q. Weinberger, and Andrew Gordon Wilson. Constant-time predictive distributions for Gaussian processes. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 4114–4123, 2018.

David Lee. Fast multiplication of a recursive block Toeplitz matrix by a vector and its application. *Journal of Complexity*, 2(4):295–305, 1986.

Robert G. Keys. Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160, 1981.

Radford Neal. Lecture notes for sta 414: Statistical methods for machine learning and data mining, 2011. Accessed at: `http://www.utstat.utoronto.ca/~radford/sta414.S11/week4a.pdf`.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Youcef Saad and Martin H Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

Rico Angell and Daniel R Sheldon. Inferring latent velocities from weather radar data using Gaussian processes. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, pages 8984–8993, 2018.

John A. Gubner. Block matrix formulas, 2015. Accessed at: `https://gubner.ece.wisc.edu/notes/BlockMatrixFormulas.pdf`.