Kernel Interpolation for Scalable Online Gaussian Processes

Samuel Stanton[1,*] ss13641@nyu.edu
Wesley J. Maddox[1,*] wjm363@nyu.edu
Ian Delbridge[2] iad35@cornell.edu
Andrew Gordon Wilson[1] andrewgw@cims.nyu.edu
[1]New York University, [2]Cornell University
[*] Equal contribution.

Abstract

Gaussian processes (GPs) provide a gold standard for performance in online settings, such as sample-efficient control and black box optimization, where we need to update a posterior distribution as we acquire data in a sequential fashion. However, updating a GP posterior to accommodate even a single new observation after having observed n points incurs at least $\mathcal{O}(n)$ computations in the exact setting. We show how to use structured kernel interpolation to efficiently recycle computations for constant-time $\mathcal{O}(1)$ online updates with respect to the number of points n, while retaining exact inference. We demonstrate the promise of our approach in a range of online regression and classification settings, Bayesian optimization, and active sampling to reduce error in malaria incidence forecasting. Code is available at `https://github.com/wjmaddox/online_gp`.

1 INTRODUCTION

The ability to repeatedly adapt to new information is a defining feature of intelligent agents. Indeed, these *online* or *streaming* settings, where we observe data in an incremental fashion, are ubiquitous — from real-time adaptation in robotics (Nguyen-Tuong et al., 2008) to click-through rate predictions for ads (Liu et al., 2017).

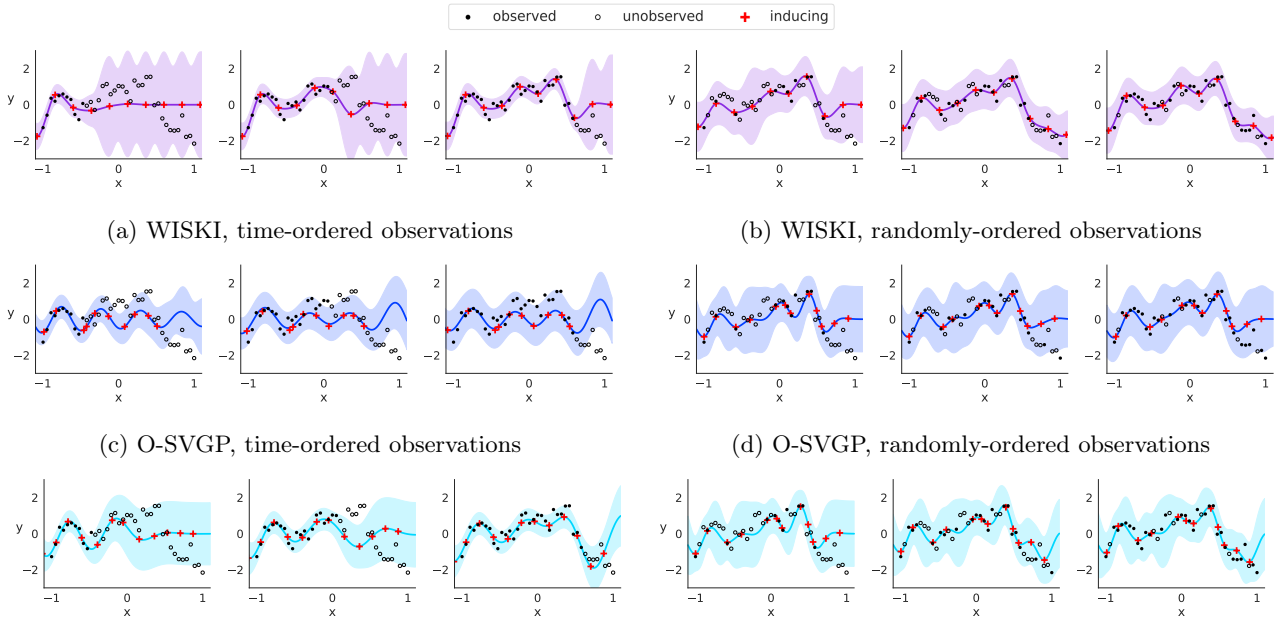
Bayesian inference is naturally suited to the online setting, where after each new observation, an old posterior becomes a new prior. However, these updates can be prohibitively slow. For Gaussian processes, if we have already observed n data points, observing even a single new point requires introducing a new row and column into an $n \times n$ covariance matrix, which can incur $\mathcal{O}(n^2)$ operations for the predictive distribution and $\mathcal{O}(n^3)$ operations for kernel hyperparameter updates.

Since Gaussian processes are now frequently applied in online settings, such as Bayesian optimization (Yamashita et al., 2018; Letham et al., 2019), or model-based robotics (Xu et al., 2014; Mukadam et al., 2016), this scaling is particularly problematic. Moreover, despite the growing need for scalable online inference, recent research on this topic is scarce.

Existing work has typically focused on data sparsification schemes paired with low-rank kernel updates (e.g., Nguyen-Tuong et al., 2008), or sparse variational posterior approximations (Cheng and Boots, 2016; Bui et al., 2017). Low-rank kernel updates are sensible but still costly, and data-sparsification can incur significant error. Variational approaches, while promising, can provide miscalibrated uncertainty representations compared to exact inference (Jankowiak et al., 2020; Lázaro-Gredilla and Figueiras-Vidal, 2009; Bauer et al., 2016), and often involve careful tuning of many hyperparameters. In the online setting, these limitations are especially acute. Uncertainty representation can be particularly crucial for determining the balance of exploration and exploitation in choosing new query points. Moreover, while tuning of hyperparameters and manual intervention may be feasible for a fixed dataset, it can become particularly burdensome in the online setting if it must be repeated after we observe each new point.

Intuitively, we ought to be able to recycle computations to efficiently update our predictive distribution after observing an additional point, rather than starting training anew on $n+1$ points. However, it is extremely challenging to realize this intuition in practice, for if we observe a new point, we must compute its interaction with every previous point. In this paper, we show it is in fact possible to perform constant-time $\mathcal{O}(1)$ updates in n, and $\mathcal{O}(m^2)$ for m inducing points, to the Gaussian process predictive distribution, marginal

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



(a) WISKI, time-ordered observations

(b) WISKI, randomly-ordered observations

(c) O-SVGP, time-ordered observations

(d) O-SVGP, randomly-ordered observations

(e) O-SGPR, time-ordered observations

(f) O-SGPR, randomly-ordered observations

Figure 1: Online GP regression on exchange rate time series data ($N = 40$). The shaded regions in each panel corresponds to a 95% credible interval. In each subplot, the left subpanel shows the predictive distribution of the corresponding model after training in batch on an initial set of 10 observations. The middle and right subpanels show the evolution of the predictive distribution after 10 and 20 online updates, respectively. The **left** plots, **(a,c,e)**, show WISKI, O-SVGP, and O-SGPR using spectral mixture kernels (Wilson and Adams, 2013) trained on observations in a time-ordered fashion. O-SVGP heavily overfits to the initial data by interpolating the first batch of data points, and struggles to recover on the next batches. WISKI and O-SGPR perform well in this situation by picking up the signal on the first batches and updating the mean as the data comes in. The **right** plots, **(b,d,f)** show the methods trained on observations in a randomly ordered fashion. Here, O-SVGP is still very under-confident, while O-SGPR clumps its inducing points in the middle of the data. By comparison, WISKI learns more of the high frequency trend than either variational approach.

likelihood, and its gradients, *while retaining exact inference*. We achieve this scaling through a careful combination of caching, structured kernel interpolation (SKI) (Wilson and Nickisch, 2015), and reformulations involving the Woodbury identity. We name our approach **W**oodbury **I**nversion with **SKI** (WISKI). We find that WISKI achieves promising results across a range of online regression and classification problems, Bayesian optimization, and an active sampling problem for estimating malaria incidence where fast online updates, exact inference for calibrated uncertainty, and fast test-time predictions are particularly crucial.

As a motivating example, in Figure 1, we fit GPs with spectral mixture kernels (Wilson and Adams, 2013) on British pound to USD foreign exchange data.[1] In this task, we observe points one at a time, after observing the first 10 points in batch, and update the predictive distributions for WISKI, O-SVGP and O-SGPR (Bui et al., 2017), state-of-the-art streaming sparse variational GPs. We illustrate snapshots after having observed $n = 10, 20$, and 30 points. We see that WISKI is able to more easily capture signal in the data, whereas O-SVGP tends to underfit and O-SGPR underfits on the random data setting. In addition to the general tendency of stochastic variational GP (SVGP) models to underfit the data and overestimate noise variance (Lázaro-Gredilla and Figueiras-Vidal, 2009; Bauer et al., 2016), the variational posterior of an O-SVGP is discouraged from adapting to surprising new observations (See Appendix B). We also see that O-SVGP particularly struggles when we observe new points in a time-ordered fashion, which is a standard setup in the online setting.

The initialization heuristics used to train SVGPs in the batch setting, such as initializing the inducing points with k-means or freezing the GP hyperparameters at

[1] `https://raw.githubusercontent.com/trungngv/cogp/master/data/fx/fx2007-processed.csv`, fourth column. We rescaled the inputs to $[-1, 1]$ and standardized the responses.

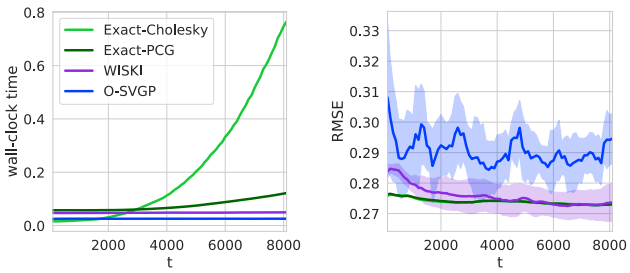


Figure 2: **Left:** Incorporating new observations becomes increasingly expensive for exact GPs (Exact-Cholesky), even when preconditioned conjugate gradients (Exact-PCG), as quantified in the left panel by the wall-clock time per iteration on the UCI Powerplant dataset. Variational GPs (O-SVGP) are an economical alternative by virtue of being constant time. WISKI has the constant-time profile of a variational method, but retains exact inference, is simple to train, and does not underfit. **Right:** RMSE on the UCI power plant dataset. Shown are mean and two standard deviations over 10 trials. O-SVGP tends to overestimate noise and converges to a sub-optimal solution, while WISKI matches the performance of the exact methods trained in an incremental fashion.

the beginning of training, are not effective for O-SVGPs since the full dataset is not available. In order to obtain reasonable fits with O-SVGP on even this motivating example, we carefully tuned tempering parameters using generalized variational inference (Knoblauch et al., 2019), executed 6 optimization steps for each new observation, and trained in batch on the first 10 points. WISKI, by contrast, requires no tuning, only 1 optimization step for each new observation, and does not require any batch training to find reasonable solutions.

These issues with O-SVGP[2] are particularly visible when we move beyond time series. In Figure 2, we plot the incremental RMSE on a held out test set on the UCI powerplant dataset, while optimizing for only a single step as we observe new data points, finding that O-SVGP underfits and sub-optimal solution, while WISKI matches the performance of an exact GP also fit incrementally. The exact GP also uses preconditioned conjugate gradients (Gardner et al., 2018) here. However, WISKI and O-SVGP are both constant time (shown in the left panel), while using an exact GP with Cholesky factorization is cubic time, and using CG with the GP is quadratic time. Both are much slower than WISKI and O-SVGP after $t = 5000$.

[2]O-SGPR has different weaknesses, including numerical instability. We further consider O-SGPR in our larger study of incremental regression.

2 RELATED WORK

2.1 Prior Approaches

Despite its timeliness, there has not been much recent work on online learning with GPs. Older work considers sparse variational approximations to GPs in the streaming setting. Csató and Opper (2002) proposed a variational sparse GP based algorithm in $\mathcal{O}(nm^2+m^3)$ time, specifically for deployment in streaming tasks; however, it assumes that the hyperparameters are fixed. Nguyen-Tuong et al. (2008) proposed local fits to GPs with weightings based on the distance of the test point to the local models. More recently, Koppel (2019) extended the types of distances used for these types of models while using an iteratively constructed coreset of data points. Evans and Nair (2018) proposed a structured eigenfunction based approach that requires one $\mathcal{O}(n)$ computation of the kernel and uses fixed kernel hyperparameters but learns interpolation weights. Cheng and Boots (2016) proposed a variational stochastic functional gradient descent method in incremental setting with the same time complexity; however, like stochastic variational GPs (Hensman et al., 2013), Cheng and Boots (2016) assumes the number of data points the model will see is known and set before training begins. Hoang et al. (2015) proposed a similar variational natural gradient ascent approach, but assumed that the hyper-parameters are fixed during the training procedure, a major limitation for flexible kernel learning.

2.2 Streaming SVGP and Streaming SGPR

The current state-of-the-art for streaming Gaussian processes is the sparse variational O-SVGP approach of Bui et al. (2017) and its "collapsed" non-stochastic variant, O-SGPR, which does not use an explicit variational distribution, like its batch equivalent, SGPR (Titsias, 2009).

O-SVGP: Unlike its predecessors, O-SVGP is fully compatible with online inference, since it has no requirements to choose the number of data points a priori, and it can update both model parameters and inducing point locations; however, it has the same time complexity as its predecessors: $\mathcal{O}(bm^2 + m^3)$, where b is the size of the batch used to update the predictive distribution and model hyper-parameters. Bui et al. (2017)'s experiments primarily focused on large batch sizes — practically $b = \mathcal{O}(n)$ — rather than the pure streaming setting. A major limitation of variational methods in the streaming setting is that conditioning on new observations effectively requires the model parameters to be re-optimized to a minima after every new batch, increasing latency. In Appendix B we include a detailed discussion of the requirements of the original

O-SVGP algorithm, and provide a modified generalized variational update by downweighting the prior by a factor of $\beta < 1$ better adapted to the streaming setting to have a strong baseline for comparison. We compare to the generalized O-SVGP implementation in our experiments as O-SVGP.

O-SGPR: O-SGPR is also potentially promising but like O-SVGP falls prey to several key limitations. First, O-SGPR relies on analytic marginalization and so can only be used for Gaussian likelihoods. In Figure 1, we implemented the O-SGPR bound in GPyTorch (Gardner et al., 2018) and it has fair performance for both the random ordering and time ordering settings, though not as good as WISKI. However, this performance comes with two caveats. First, we need to re-sample the inducing points to include some of the new data at each iteration, as is done in Bui et al. (2017)'s implementation. Second, we found that even in double precision we needed to add a large amount of jitter $\epsilon = 0.01$, while doing the required Cholesky decompositions (there is a matrix subtraction) to prevent numerical instability.

3 BACKGROUND

For a complete treatment on Gaussian Processes, see Williams and Rasmussen (2006). Here we briefly review the key ideas for efficient exact GPs, SKI, and the conditioning of GPs on new observations online. We note SKI provides scalable exact inference through creating an approximate kernel which admits fast computations.

3.1 Exact GP Regression

Starting with the regression setting, suppose $\mathbf{y} = f(x) + \varepsilon$, $f \sim \mathcal{GP}(0, k_\theta(x, x'))$, and $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Here, $k_\theta(x, x')$ is the kernel function with hyperparameters θ, and $K_{AB} := k_\theta(A, B)$ is the covariance between two sets of data inputs A and B. Given training data $\mathcal{D} = (X, \mathbf{y})$, we can train the GP hyperparameters by maximizing the marginal log-likelihood,

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2}\mathbf{y}^\top (K_{XX} + \sigma^2 I)^{-1}\mathbf{y} - \frac{1}{2}\log|K_{XX} + \sigma^2 I| - \frac{n}{2}\log 2\pi. \quad (1)$$

Conventionally, solving the linear system, $(K_{XX} + \sigma^2 I)^{-1}\mathbf{y}$, in Eq. 1 costs $\mathcal{O}(n^3)$ operations. The posterior predictive distribution of a new test point $p(f(\mathbf{x}^*)|\mathbf{x}^*, \mathcal{D}, \theta) = \mathcal{N}(\mu_{f|\mathcal{D}}, \sigma^2_{f|\mathcal{D}})$, where

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = K_{\mathbf{x}^*X}(K_{XX} + \sigma^2 I)^{-1}\mathbf{y}, \quad (2)$$

$$\sigma^2_{f|\mathcal{D}}(\mathbf{x}^*) = K_{\mathbf{x}^*\mathbf{x}^*} - K_{\mathbf{x}^*X}(K_{XX} + \sigma^2 I)^{-1}K_{X\mathbf{x}^*} \quad (3)$$

We build on previous work on scaling GP training and prediction by exploiting kernel structure and efficient GPU matrix vector multiply routines to quickly compute gradients (CG) of Eq. 1 for training, and by caching terms in Eq. 2 and Eq. 3 for fast prediction (Gardner et al., 2018). Conjugate gradient methods improve the asymptotic complexity of GP regression and to $\mathcal{O}(jn^2)$, where j is the number of CG steps used. These recent advances in GP inference have enabled exact GP regression on datasets of up to one million data points in the batch setting (Wang et al., 2019).

3.2 SKI and Lanczos Variance Estimates

GPs are often *sparsified* through the introduction of inducing points (also known as pseudo-inputs), which are small subset of fixed points (Snelson and Ghahramani, 2006). In particular, Wilson and Nickisch (2015) proposed structured kernel interpolation (SKI) to approximate the kernel matrix as $K_{XX} \approx \tilde{K}_{XX} = WK_{UU}W^\top$, where U represents the m inducing points, and $W \in \mathbb{R}^{n\times m}$ is a sparse cubic interpolation matrix composed of n vectors $\mathbf{w}_i \in \mathbb{R}^m$. Each vector $\mathbf{w}_i$ is sparse, containing 4^d non-zero entries, where d is the dimensionality of the input data. SKI places the inducing points on a multi-dimensional grid. When k_θ is stationary and factorizes across dimensions, K_{UU} can often be expressed as a Kronecker product of Toeplitz matrices, leading to fast multiplies. Overall multiplies with $\tilde{K}_{XX}$ take $\mathcal{O}(n + g(m))$ time, where $g(m) \approx m$ (Wilson and Nickisch, 2015), compared to the $\mathcal{O}(nm^2 + m^3)$ complexity associated with most inducing point methods (Quinonero-Candela and Rasmussen, 2005). In short, SKI provides scalable exact inference, through introducing an approximate kernel that admits fast computations.

Pleiss et al. (2018) propose to cache (i.e. to store in memory) all parts of the predictive mean and covariance that can be computed before prediction, enabling constant time predictive means and covariances. Directly substituting the SKI kernel matrix, $\tilde{K}_{XX}$, into Eq. 2, the predictive mean becomes

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = \mathbf{w}_{\mathbf{x}^*}^\top \underbrace{K_{UU}W^\top \left(WK_{UU}W^\top + \sigma^2 I\right)^{-1}\mathbf{y}}_{\mathbf{a}},$$

where **a** is the *predictive mean cache*.[3] Similarly, Eq. 3 becomes

$$\sigma^2_{f|\mathcal{D}}(\mathbf{x}_i^*, \mathbf{x}_j^*) = k(\mathbf{x}_i, \mathbf{x}_j^*) - \mathbf{w}_{\mathbf{x}^*}^\top \underbrace{K_{UU}W^\top \left(\tilde{K}_{XX} + \sigma^2 I\right)^{-1} WK_{UU}}_{C}\mathbf{w}_{\mathbf{x}^*},$$

[3] We refer to entities that can be computed, stored in memory, and used in subsequent computations as *caches*. We use blue font to identify which cached expressions.

where $C \approx SS^\top$, is the *predictive covariance cache*. S is formed by computing a rank-k root decomposition of $(\tilde{K}_{XX} + \sigma^2 I)^{-1} \approx RR^\top$ and taking $S = K_{UU} W^\top R$. The complexity of the root decomposition is $\mathcal{O}(km^2)$, requiring $k \leq m$ iterations of the Lanczos algorithm (Lanczos, 1950) and a subsequent eigendecomposition of the resulting $k \times k$ symmetric tridiagonal matrix. Further details on Lanczos decomposition and the caching methods of (Pleiss et al., 2018) are in Appendix A.

3.3 Online Conditioning and Low-Rank Matrix Updates

GP models are conditioned on new observations through Gaussian marginalization (Williams and Rasmussen, 2006, Chapter 2). Suppose we have past observations $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ used to make predictions via $p(y^*|\mathbf{x}^*, \mathcal{D}, \theta)$. We subsequently observe a new data point $(\mathbf{x}', y')$. For clarity, let $X = \mathbf{x}_{1:n}$, $X' = X \cup \{\mathbf{x}'\}$. The new kernel matrix is

$$K_{X'X'} = \begin{pmatrix} K_{XX} & k(X, \mathbf{x}') \\ k(\mathbf{x}', X) & k(\mathbf{x}', \mathbf{x}') \end{pmatrix} \tag{4}$$

We would like to update our posterior predictions to incorporate the new data point without recomputing our caches that are not hyper-parameter dependent from scratch. If the hyperparameters are fixed, this can be a $\mathcal{O}(n^2)$ low-rank update to the predictive covariance matrix (e.g. a Schur complement update or low rank Cholesky update to a decomposition of $(K_{XX}+\sigma^2 I)^{-1}$). If we additionally wish to update hyper-parameters, we must recompute the marginal log likelihood in Eq. 1, which costs $\mathcal{O}(n^3)$. Similarly, if we naively use the SKI approximations in Eq. 4 we additionally have an $\mathcal{O}(n)$ cost for both adding a new data point and to update the hyper-parameters afterwards. Thus, as n increases, training and prediction will slow down (Figure 2).

4 WISKI: ONLINE CONSTANT TIME SKI UPDATES

We now propose WISKI, which through a careful combination of caching, SKI, and the Woodbury identity, achieves constant time (in n) updates in the streaming setting, while retaining exact inference. To begin, we present two key identities that result from the application of the Woodbury matrix identity to the inverse of the updated SKI approximated kernel, $\tilde{K}_{X_t X_t}$, after having received t data points. First, we can rewrite the SKI kernel inverse as

$$(\tilde{K}_{XX} + \sigma^2 I)^{-1} = \frac{1}{\sigma^2} I - \frac{1}{\sigma^2} W M W^\top. \tag{5}$$

$$M := (\sigma^2 K_{UU}^{-1} + W^\top W)^{-1}. \tag{6}$$

Second, after observing a new data point at time $t+1$, the inner matrix inverse term M can be updated via a rank-one update,

$$M_{t+1}^{-1} = M_t^{-1} + \mathbf{w}_{t+1} \mathbf{w}_{t+1}^\top, \tag{7}$$

where $\mathbf{w}_{t+1}$ is an interpolation vector for the $t+1$th data point.[4] The exploitation of this rank-one update on a fixed size matrix by storing $W^\top W$ will form the basis of our work.

Computing Eq. 7 as written requires explicit computation of K_{UU}^{-1}. In general, K_{UU} will have significant structure as U is a dense grid, which yields fast matrix inversion algorithms; however, the inverse will be very ill-conditioned because many kernel matrices on gridded data have (super-)exponentially decaying eigenvalues (Bach and Jordan, 2002). We will instead focus on reformulating SKI into expressions that depend only on K_{UU}, W, and $\mathbf{y}$ with a constant $\mathcal{O}(m^2)$ memory footprint and can be computed in $\mathcal{O}(m^2)$ time.

4.1 Computing the Marginal Log-Likelihood, Predictive Mean and Predictive Variance

Substituting Eq. (5) into Eqs. (1), (2), and (3), we obtain the following expressions for the marginal log-likelihood (MLL), predictive mean, and predictive variance[5]:

$$\log p(\mathbf{y}|X, \theta) = -\frac{1}{2\sigma^2}(\mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top W M W^\top \mathbf{y}) - \frac{1}{2}\left(\log|K_{UU}| - \log|M| + (n-m)\log\sigma^2\right), \tag{8}$$

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = \mathbf{w}_{\mathbf{x}^*}^\top M W^\top \mathbf{y}, \tag{9}$$

$$\sigma^2_{f|\mathcal{D}}(\mathbf{x}_i^*, \mathbf{x}_j^*) = \sigma^2 \mathbf{w}_{\mathbf{x}_i^*}^\top M \mathbf{w}_{\mathbf{x}_j^*}. \tag{10}$$

For all derivations see Appendix A. We begin by constructing a rank r root decomposition of the matrix $W^\top W \approx LL^\top$, along with the factorization of the (pseudo-)inverse, $JJ^\top \approx (W^\top W)^+$. The root decomposition $LL^\top$ can be a full Cholesky factorization ($r = m$) for relatively small m (i.e. $m \leq 1000$) or an approximate Lanczos decomposition for larger m, at a one-time cost of $\mathcal{O}(m^2 r)$. Applying the Woodbury matrix identity to Eq. (6) and substituting $W^\top W \approx LL^\top$, we have

$$M = \sigma^{-2} K_{UU} - \sigma^{-2} K_{UU} L Q^{-1} L^\top \sigma^{-2} K_{UU}, \tag{11}$$

$$Q := I + L^\top \sigma^{-2} K_{UU} L. \tag{12}$$

$Q^{-1} L^\top$ is a $r \times r$ system, so directly computing Eq. (11) requires $\mathcal{O}(r^2 m)$ time for the solve using conjugate

[4] We have dropped the dependence on $\mathbf{x}$ for simplicity of notation.

[5] A similar result holds for fixed noise heteroscedastic likelihoods as well. See Appendix A.5 for further details.

gradients, $\mathcal{O}(rm \log m)$ time for the matrix multiplications with K_{UU} if it has Toeplitz structure, and $\mathcal{O}(m^2)$ for the dense matrix additions, and $\mathcal{O}(km)$ for the root decomposition of $W^\top W$, for a final total of $\mathcal{O}(r^2 m + km \log m + m^2)$. However, we do not *explicitly* store the matrix M as doing so would require m solves of a $r \times r$ system since $L \in \mathbb{R}^{m \times r}$.

Eqs. (8) - (10) involve computations of the form

$$M\mathbf{v} = \sigma^{-2} K_{UU}\mathbf{v} - \sigma^{-2} K_{UU} L Q^{-1} L \sigma^{-2} K_{UU}\mathbf{v},$$

which can be computed using only a *single* solve against the matrix Q via first multiplying out $\mathbf{a} = L\sigma^{-2}K_{UU}\mathbf{v}$, and then computing $\mathbf{b} = Q^{-1}\mathbf{a}$. Applying the matrix determinant identity to $\log|M|$ results in a simplified expression in terms of $\log|Q|$. Taking $\mathbf{v} = W^\top \mathbf{y}$, we obtain a practical expression for the MLL,

$$\log p(\mathbf{y}|X,\theta) = -\frac{1}{2\sigma^2}\left(\mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top W K_{UU} W^\top \mathbf{y} + \mathbf{a}^\top Q^{-1}\mathbf{a}\right) - \frac{1}{2}\left(-\log|Q| + (n-m)\log\sigma^2\right). \quad (13)$$

Computing $\mathbf{a}$ costs $\mathcal{O}(m \log m + rm)$, so computing the two quadratic forms are $\mathcal{O}(m \log m + m)$ and $\mathcal{O}(jr^2)$ respectively, assuming j steps of conjugate gradients. We use stochastic Lanczos quadrature to compute the log determinant of $|Q|$ which costs $\mathcal{O}(jr^2)$ (Gardner et al., 2018). Overall, computation of the MLL becomes $\mathcal{O}(rm + m \log m + jr^2)$.

The predictive mean is similarly computed by taking $\mathbf{v} = W^\top y$, resulting in the expression

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = \mathbf{w}_{\mathbf{x}^*}^\top \left(\sigma^{-2} K_{UU}(W^\top \mathbf{y} - L\mathbf{b})\right). \quad (14)$$

The only term that remains is the predictive variance, for which we take $\mathbf{v} = \mathbf{w}_{\mathbf{x}_j^*}$ and obtain

$$\sigma^2_{f|\mathcal{D}}(\mathbf{x}_i^*, \mathbf{x}_j^*) = \sigma^2 \mathbf{w}_{\mathbf{x_i}^*}^\top (K_{UU}(\mathbf{w}_{\mathbf{x}_j^*} - L\mathbf{b})). \quad (15)$$

4.2 Conditioning on New Observations

When we observe a new data point $(\mathbf{x}_{t+1}, y_{t+1})$, we need to update $(W^\top \mathbf{y})_t$, $(\mathbf{y}^\top \mathbf{y})_t$, and $L_t L_t^\top = (W^\top W)_t$. The update to the first two terms is simple:

$$(W^\top y)_{t+1} = (W^\top \mathbf{y})_t + y_{t+1}\mathbf{w}_{\mathbf{x}_{t+1}} \quad (16)$$

$$(\mathbf{y}^\top \mathbf{y})_{t+1} = (\mathbf{y}^\top \mathbf{y})_t + y_{t+1}^2 \quad (17)$$

We can update L_t in $\mathcal{O}(mr + r)$ time by exploiting the rank-one structure of the expression

$$(W^\top W)_{t+1} = (W^\top W)_t + \mathbf{w}_{\mathbf{x}_{t+1}} \mathbf{w}_{\mathbf{x}_{t+1}}^\top.$$

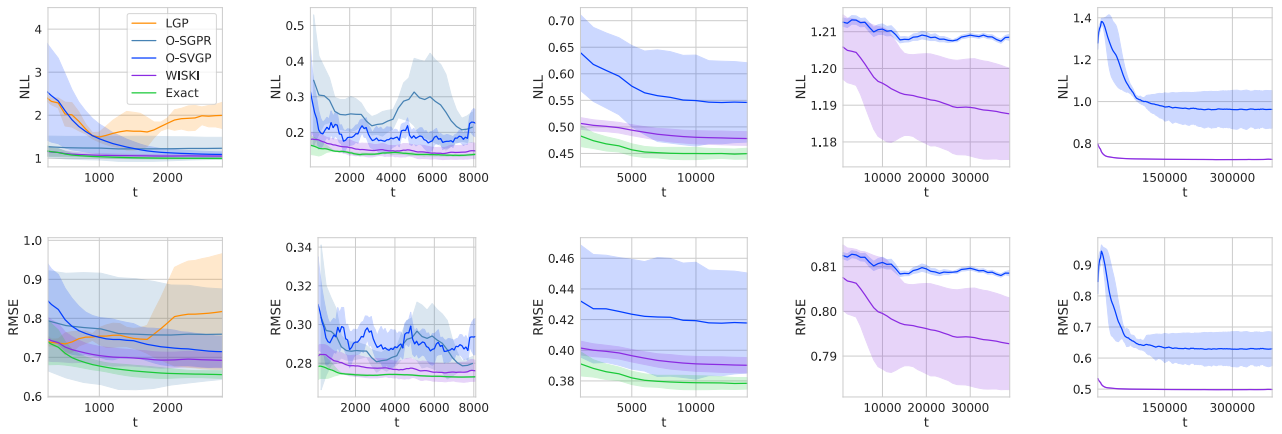
Recalling that $JJ^\top = (W^\top W)^+$, let $\mathbf{p} = J_t{}^\top \mathbf{w}_{\mathbf{x}_{t+1}}$. We compute the decomposition $BB^\top = I_r + \mathbf{p}\mathbf{p}^\top$ and obtain the expression for the updated root $L_{t+1} = L_t B$. Since $BB^\top$ is a decomposition of I_r plus a rank-one correction, it can be computed in $\mathcal{O}(r)$ time. Since the updates to the first two caches are $\mathcal{O}(1)$ and $\mathcal{O}(m)$, respectively, the total complexity of conditioning on a new observation is $\mathcal{O}(mr^2 + r)$. Further details and an extended proof are given in Appendix A.

4.3 Updating Kernel Hyperparameters

Conventionally, learning the kernel hyperparameters θ of a GP online presents two major challenges. First, the basic form of the gradient of the MLL naively costs at least $\mathcal{O}(n)$, even if scalable methods are employed. Second, after a parameter update, any cached terms that depend on the kernel matrix must be recomputed (e.g. a new factorization of K_{XX}). The reformulation of the MLL in Eq. (13) addresses the first challenge, with a complexity of $\mathcal{O}(rm + m \log m + jr^2)$ (after computing the necessary caches). To address the second challenge, we observe that the combination of the SKI approximation to the kernel matrix and the Woodbury matrix identity in Section 4.1 has allowed us to reformulate GP inference entirely in terms of computations whose cost depends only on the number of inducing points and the rank of the matrix decompositions (which is at most m, and typically much less than m). As a result, we can recompute the necessary caches as needed without any increase in computational effort as n increases.

The computational efficiency of SKI is a direct result of the grid structure imposed on the inducing points. The reduced computational complexity comes at the cost of memory complexity that is exponential in the dimension of the input. In practice, if the input data has more than three or four dimensions, the inputs must be projected into a low-dimensional space. The projection may be random (Delbridge et al., 2020) or learned (Wilson et al., 2016), depending on the requirements of the task. If the projection is learned, then the parameters ϕ of the projection operator h are treated as additional kernel hyperparameters and trained through the marginal log-likelihood.

In the batch setting the interpolation weights W are updated after every optimization iteration to adapt to the new projected features $h(\mathbf{x}_{1:n}; \phi)$. In the online setting, updating W for every previous observation would be $\mathcal{O}(n)$. Since we cannot update the interpolation weights for old observations, the gradient for the projection parameters at time t can be rewritten as



(a) Skillcraft ($n = 2855$) (b) Powerplant ($n = 8182$) (c) Elevators ($n = 14193$) (d) Protein ($n = 39100$) (e) 3DRoad ($n = 391000$)

Figure 3: Online homoskedastic regression on UCI datasets. We compare to local GPs (LGP), O-SGPR, O-SVGP, and exact GPs. Due to memory constraints or numerical issues for other methods, only O-SVGP and WISKI were easily capable of running on the larger tasks (Protein and 3DRoad). WISKI has comparable accuracy to exact methods, with comparable runtime to scalable approximations like OSVGP. **Top:** Test set NLLs. **Bottom:** Test set RMSEs.

follows:

$$\nabla_\phi \mathcal{L}(\phi) = \nabla_\phi \frac{1}{2}\Bigg((\mathbf{y}^\top W)_t M_{t-1} (W^\top \mathbf{y})_t - \frac{1}{1+\mathbf{v}^\top \mathbf{w}} \left(\mathbf{v}_t^\top (W^\top \mathbf{y})_t\right)^2 - \log(1+\mathbf{v}_t \mathbf{w}_t)\Bigg), \quad (18)$$

$$\mathbf{w}_t = w(h(\mathbf{x}_t;\phi)), \quad \mathbf{v}_t = M_{t-1}\mathbf{w}_t,$$

$$(W^\top \mathbf{y})_t = (W^\top \mathbf{y})_{t-1} + y_t \mathbf{w}_t.$$

The gradient in Eq. (18) will move the projection parameters ϕ in a direction that maximizes the marginal likelihood, assuming $\mathbf{w}_{1:t-1}$ are fixed. That is, only projections on new data are updated, while the old projections remain fixed. In contrast to the batch setting, where ϕ is jointly optimized with θ, the online update is sequential; whenever a new observation is received ϕ is updated through Eq. (18), then the GP is conditioned on $(h(\mathbf{x}_t;\phi_t), y_t)$, and finally θ is updated through Eq. (1). See Appendix A.4 for the full derivation.

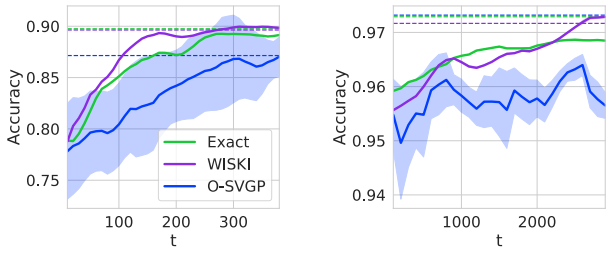
5 EXPERIMENTS

To evaluate WISKI, we first consider online regression and binary classification. We then demonstrate how WISKI can be used to accelerate Bayesian optimization, a fundamentally online algorithm often applied to experiment design and hyperparameter tuning (Frazier, 2018). Finally we consider an active learning problem for measuring malaria incidence, and show that the scalability of WISKI enables much longer horizons than a conventional GP.

We compare against exact GPs (no kernel approximations), O-SGPR, O-SVGP (Bui et al., 2017), sparse variational methods that represents the current gold standard for scalable online Gaussian processes, and local GPs (LGPs) (Nguyen-Tuong et al., 2008). All experimental details (hyper-parameters, data preparation, etc.) are given in Appendix C, where we also include ablation studies on the β parameter for O-SVGP as well as the the number of inducing points (as we had to modify it to achieve good results in the incremental setting for O-SVGP), m, for WISKI and O-SVGP. Unless stated otherwise, shaded regions in the plots correspond to $\overline{\mu} \pm 2\overline{\sigma}$, estimated from 10 trials.

5.1 Regression

We first consider online regression on several datasets from the UCI repository (Dua and Graff, 2017). In each trial we split the dataset into a 90%/10% train/test split. We scaled the raw features to the unit hypercube $[-1, 1]^d$ and normalized the targets to have zero-mean and unit variance. Each model learned a linear projection from $\mathbb{R}^d$ to $\mathbb{R}^2$ that was transformed via a batch-norm operation and the non-linear tanh activation to ensure that the features were constrained to $[-1, 1]^2$. Each model learned an RBF-ARD kernel on the transformed features, except on the *3DRoad* dataset, which did not require dimensionality reduction. We used the same number of inducing points for WISKI, O-SGPR, and O-SVGP and set $n_{\max} = m$ for local GPs. The models were pretrained on 5% of the training examples, and then trained online for the remaining 95%. When



(a) Banana ($n = 400$) (b) SVM Guide 1 ($n = 3000$)

Figure 4: A comparison of Dirichlet-based exact and WISKI GP classifiers against an O-SVGP with a binomial likelihood. The exact and WISKI models overfit less to the initial data and ultimately match the performance of their hindsight counterparts trained on the full dataset, shown as a dotted line.

adding a new data point, we update with a single optimization step for each method, such that the runtime is similar for the scalable methods. Since O-SVGP can be sensitive to the number of gradient updates per timestep, in Figure A.2 in the Appendix we provide results for an ablation.

We show the test NLL and RMSE for each dataset in Figure 3. For the two largest datasets we only report results for WISKI and O-SVGP. We found that O-SGPR was fairly unstable numerically, even after using an exceptionally large jitter value (0.01) and switching from single to double precision. The exact baseline and the WISKI model overfit less to the initial examples than O-SGPR or O-SVGP. Note that O-SVGP is a much stronger baseline in this experiment since the observations are independently observed than would typically be the case for correlated time-series data, as seen in Figure 1.

5.2 Classification

We extend WISKI to classification though the Dirichlet-based GP (GPD) classification formulation of Milios et al. (2018), which reformulates classification as a regression problem with a fixed noise Gaussian likelihood. Empirically the approach has been found to be competitive with the conventional softmax likelihood formulation. In Figure 4 we compare exact and WISKI GPD classifiers to O-SVGP with binomial likelihood on two binary classification tasks, Banana[6] and SVM Guide 1 (Chang and Lin, 2011). Banana has 2D features, and SVM Guide has 4D features, so we did not need to learn a projection. As in the UCI regression tasks, WISKI and O-SVGP both had 256 inducing point, each model used an RBF-ARD kernel, and the classifiers were pretrained on 5% of the training examples and trained online on the remaining 95%. In both cases the WISKI classifier outperformed the O-SVGP baseline, and matched the accuracy of the the exact baseline.

5.3 Bayesian Optimization

In Bayesian optimization (BO) one optimizes a black-box function by iteratively conditioning a surrogate model on observed data and choosing new observations by optimizing an acquisition function based on the model posterior (Frazier, 2018). Thus, BO requires efficient posterior predictions, updates to caches as new data are observed, and hyperparameter updates. While BO has historically been applied only to expensive-to-query objective functions, we demonstrate here that large-scale Bayesian optimization is possible with WISKI. Our BO experiments are conducted as follows: we choose 5 initial observations using random sampling, then iteratively optimize a batched version of upper confidence bound (UCB) (with $q = 3$) using BoTorch (Balandat et al., 2020) and compute an online update to each GP model, before re-fitting the model. Accurate model fits are critical to high performance; therefore, we wish to use as many inducing points for WISKI and O-SVGP as possible. For both methods, we 1000 inducing points. We show the results over four trials plotting mean and two standard deviations in Figure 5a for the Ackley benchmarks. On both problems, WISKI is significantly faster than the exact GP and O-SVGP, while achieving comparable performance on Levy. We show results over a wider range of test functions in Appendix C.2, along with the best achieved point plotted against the number of steps and the average time per step.

5.4 Active Learning

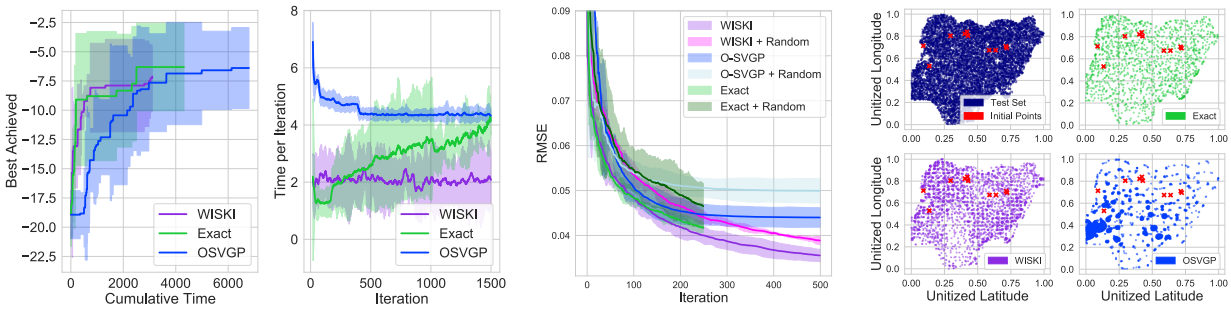
Finally, we apply WISKI to an active learning problem inspired by Balandat et al. (2020). We consider data describing the infection rate of *Plasmodium falciparum* (a parasite known to cause malaria)[7] in 2017. We wish to choose spatial locations to query malaria incidence in order to make the best possible predictions on withheld samples. To selectively choose points, we minimize the negative integrated posterior variance (NIPV, Seo et al., 2000), defined as

$$\text{NIPV}(x) := -\int_{\mathcal{X}} \mathbb{E}(\mathbb{V}(f(x)|\mathcal{D}_{\boldsymbol{x}})|\mathcal{D})dx.$$

Optimizing this acquisition function amounts to finding the batch of data points $\mathbf{x}_{1:q}$, the fantasy points,

[6] https://raw.githubusercontent.com/thangbui/streaming_sparse_gp/master/data

[7] Downloaded from the Malaria Global Atlas.



(a) Bayesian optimization. (b) Active learning, RMSE. (c) Active learning, fantasy points.

Figure 5: **(a):** Objective value as a function of cumulative time and time per iteration on the Levy test problem with noise standard deviation 10.0, while performing Bayesian optimization with EI acquisition for 1500 steps with a batch size of 3 so that by the end 4500 data points have been acquired. WISKI allows rapid updates of the posterior surrogate objective out to thousands of observations, while preserving the rapid convergence rate and asymptotic optimality of the exact GP. **(b):** RMSE on the test set after choosing new points either randomly or with qnIPV (for WISKI and exact GPs) or by the maximal posterior variance (for O-SVGP). WISKI is able to continue improving the downstream error throughout the entire experiment matching the performance of the exact GP, while O-SVGP's performance flatlines. We also compare against the RMSE of models that have data points randomly selected (shown with Random in the legend). **(c):** The test set (navy), as well as points chosen for all three methods with initial points (red). WISKI and the exact GP query the entire support, while O-SVGP queries clump together.

which when added into the GP model will reduce the variance on the domain of the model the most. Here, we randomly sample $10,000$ data points in Nigeria to serve as a test set that we wish to reduce variance on and select $q = 6$ data points at a time from a held-out training set (to act as a simulator) at a time; the inner expectation drops out because the posterior variance only depends on the fantasy points and the currently observed data, and not any fantasized responses. Stochastic variational models do not have a straightforward mechanism for fantasizing (i.e. recomputing the posterior variance after updating a new data point conditional on the fantasy points), so we instead query the test set predictive variance and then choose the training points closest to the six test set points with maximum predictive variance.

As both mean and variances are available for the given locations, we model the data with a fixed noise Gaussian process with scaled Matern 0.5 kernels, beginning with an initial set of 10 data points, and iterating out for 500 iterations for WISKI and O-SVGP and 250 iterations for an exact GP model (the limit of data points that a single GPU could handle due to the large amount of test points). We show the results of the experiment in Figure 5b across three trials, finding that all of the models reduce the RMSE considerably from the initial fits; however, O-SVGP and its random counterpart stagnate in RMSE after about 250 trials, while both the exact GP and WISKI continue improving throughout the entire experiment. Closer examination of the points queried by all of the three methods in Figure 5c, we find that the points queried by O-SVGP tend to clump together, locally reducing variance, while WISKI and the exact GP choose points throughout the entire support of the test set, choosing points which better reduce global variance.

6 CONCLUSION

We have shown how to achieve constant-time online updates with Gaussian processes while retaining exact inference. Our approach, WISKI, achieves comparable performance to Gaussian processes with exact kernels, and comparable speed to state-of-the-art streaming Gaussian processes based on variational inference. Despite the present day need for scalable online probabilistic inference, recent research into online Gaussian processes has been relatively scarce. We hope that our work is a step towards making streaming Bayesian inference more widely applicable in cases when both speed and accuracy are crucial for online decision making.

Acknowledgements

WJM, SS, AGW are supported by an Amazon Research Award, NSF I-DISRE 193471, NIH R01 DA048764-01A1, NSF IIS-1910266, and NSF 1922658 NRT-HDR: FUTURE Foundations, Translation, and Responsibility for Data Science. WJM was additionally supported by an NSF Graduate Research Fellowship under Grant No. DGE-1839302. SS is additionally supported by the United States Department of Defense through the

National Defense Science & Engineering Graduate (NDSEG) Fellowship Program. We'd like to thank Max Balandat, Jacob Gardner, and Greg Benton for helpful comments.

References

Bach, F. R. and Jordan, M. I. (2002). Kernel independent component analysis. *Journal of machine learning research*, 3(Jul):1–48.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*.

Bauer, M., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse gaussian process approximations. In *Advances in neural information processing systems*, pages 1533–1541.

Bui, T. D., Nguyen, C. V., and Turner, R. E. (2017). Streaming sparse Gaussian process approximations. In *Advances in Neural Information Processing Systems 31*, pages 3301–3309, Long Beach, California, USA. Curran Associates Inc.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cheng, C.-A. and Boots, B. (2016). Incremental variational sparse Gaussian process regression. In *Advances in Neural Information Processing Systems 30*, pages 4410–4418, Barcelona, Spain. Curran Associates Inc.

Csató, L. and Opper, M. (2002). Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668.

Delbridge, I., Bindel, D., and Wilson, A. G. (2020). Randomly projected additive Gaussian processes for regression. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2453–2463, Virtual. PMLR.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Evans, T. and Nair, P. (2018). Scalable gaussian processes with grid-structured eigenfunctions (gp-grief). In *International Conference on Machine Learning*, pages 1417–1426.

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

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, pages 7576–7586.

Gill, P. E., Golub, G. H., Murray, W., and Saunders, M. A. (1974). Methods for modifying matrix factorizations. *Mathematics of computation*, 28(126):505–535.

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU press.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, page 282–290, Arlington, Virginia, USA. AUAI Press.

Hoang, T. N., Hoang, Q. M., and Low, B. K. H. (2015). A unifying framework of anytime sparse gaussian process regression models with stochastic variational inference for big data. volume 37 of *Proceedings of Machine Learning Research*, pages 569–578, Lille, France. PMLR.

Jankowiak, M., Pleiss, G., and Gardner, J. (2020). Parametric Gaussian process regressors. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4702–4712, Virtual. PMLR.

Knoblauch, J., Jewson, J., and Damoulas, T. (2019). Generalized variational inference. *arXiv preprint arXiv:1904.02063*.

Koppel, A. (2019). Consistent Online Gaussian Process Regression Without the Sample Complexity Bottleneck. In *2019 American Control Conference (ACC)*, pages 3512–3518. ISSN: 2378-5861.

Lanczos, C. (1950). *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA.

Lázaro-Gredilla, M. and Figueiras-Vidal, A. (2009). Inter-domain gaussian processes for sparse inference using inducing features. In *Advances in Neural Information Processing Systems*, pages 1087–1095.

Letham, B., Karrer, B., Ottoni, G., Bakshy, E., et al. (2019). Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519.

Liu, X., Xue, W., Xiao, L., and Zhang, B. (2017). Pbodl: Parallel bayesian online deep learning for click-through rate prediction in tencent advertising system. *arXiv preprint arXiv:1707.00802*.

Milios, D., Camoriano, R., Michiardi, P., Rosasco, L., and Filippone, M. (2018). Dirichlet-based gaussian processes for large-scale calibrated classification. In

Advances in Neural Information Processing Systems, pages 6005–6015.

Mukadam, M., Yan, X., and Boots, B. (2016). Gaussian process motion planning. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 9–15. IEEE.

Nguyen-Tuong, D., Peters, J., and Seeger, M. (2008). Local Gaussian process regression for real time online model learning and control. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'08, pages 1193–1200, Vancouver, British Columbia, Canada. Curran Associates Inc.

Pleiss, G., Gardner, J., Weinberger, K., and Wilson, A. G. (2018). Constant-time predictive distributions for Gaussian processes. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4114–4123, Stockholmsmässan, Stockholm Sweden. PMLR.

Quinonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959.

Seo, S., Wallat, M., Graepel, T., and Obermayer, K. (2000). Gaussian process regression: Active data selection and test point rejection. In *Neural Networks, IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 3241–3241.

Snelson, E. and Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.

Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In *Artificial Intelligence and Statistics*, pages 567–574.

Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam.

Wang, K. A., Pleiss, G., Gardner, J. R., Tyree, S., Weinberger, K. Q., and Wilson, A. G. (2019). Exact Gaussian Processes on a Million Data Points. In *Advances in Neural Information Processing Systems*. arXiv: 1903.08114.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, pages 1067–1075.

Wilson, A. and Nickisch, H. (2015). Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784. ISSN: 1938-7228 Section: Machine Learning.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. (2016). Deep Kernel Learning. In *Artificial Intelligence and Statistics*. arXiv: 1511.02222.

Xu, N., Low, K. H., Chen, J., Lim, K. K., and Ozgul, E. B. (2014). Gp-localize: Persistent mobile robot localization using online sparse gaussian process observation model. *arXiv preprint arXiv:1404.5165*.

Yamashita, T., Sato, N., Kino, H., Miyake, T., Tsuda, K., and Oguchi, T. (2018). Crystal structure prediction accelerated by bayesian optimization. *Physical Review Materials*, 2(1):013803.

Kernel Interpolation for Scalable Online Gaussian Processes: Supplemental Material

A FULL DERIVATIONS

A.1 Efficient Computation of GP Predictive Distributions

In this section we provide a brief summary of a major contribution of Pleiss et al. (2018). Since our cached approach to online inference was partially inspired by the approach of Pleiss et al. (2018), it is helpful to first understand how predictive means and variances are efficiently computed in the batch setting.

The Lanczos algorithm is a Krylov subspace method that can be used as a subroutine to solve linear systems (i.e. the conjugate gradients algorithm) or to solve large eigenvalue problems (Golub and Van Loan, 2012). Given a square matrix $A \in \mathbb{R}^{n \times n}$ and initial vector $\mathbf{b} \in \mathbb{R}^n$, the d-rank Krylov subspace is defined as $\mathcal{K}_d(A, \mathbf{b}) := \text{span}\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \dots, A^{d-1}\mathbf{b}\}$. The Lanczos algorithm is an iterative method that produces (after d iterations) an orthogonal basis $Q_d \in \mathbb{R}^{n \times d}$ and symmetric tridiagonal matrix $T_d \in \mathbb{R}^{d \times d}$ such that $\mathbf{q}_i \in \mathcal{K}_d(A, \mathbf{b})$ and $T_d = Q_d^\top A Q_d$. [8] If we take $A \approx Q_d T_d Q_d^\top$ and compute the eigendecomposition $T_d = V_d \Lambda_d V_d^\top$, we can write $A \approx SS^\top$, where $S = Q_d V_d \Lambda_d^{1/2}$. Note that if $d = n$ then the decomposition is exact to numerical precision. Hence the computational cost of a root decomposition via Lanczos is $\mathcal{O}(dn^2 + d^2)$ (Trefethen and Bau III, 1997).

The predictive mean caches are straightforward, since they are just the solution $\mathbf{a} = (K_{XX} + \sigma^2 I)^{-1}\mathbf{y}$ which can be stored regardless of the method used to solve the system (i.e. preconditioned CG, Cholesky factorization, *e.t.c.*). Once computed, in the exact inference setting the predictive mean is given by

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = k(\mathbf{x}^*, X)\mathbf{a}.$$

For inference with SKI we take $\mathbf{a} = K_{UU}W^\top(WK_{UU}W^\top + \sigma^2 I)^{-1}\mathbf{y}$ and

$$\mu_{f|\mathcal{D}}(\mathbf{x}^*) = w(\mathbf{x}^*)^\top \mathbf{a}.$$

For exact inference, the predictive covariance caching procedure of Pleiss et al. (2018) begins with the root decomposition

$$K_{XX} + \sigma^2 I = (Q_d V_d \Lambda_d^{1/2})(\Lambda_d^{1/2} V_d^\top Q_d^\top). \tag{A.1}$$

Since predictive variances require a root decomposition of $(K_{XX} + \sigma^2 I)^{-1}$, they store $S = Q_d^\top V_d^\top \Lambda_d^{-1/2}$ and obtain predictive variances as follows:

$$\sigma^2_{f|\mathcal{D}}(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, X)SS^\top k(X, \mathbf{x}^*). \tag{A.2}$$

For inference with SKI the procedure is much the same, except the root decomposition in Eq. A.1 is replaced with that of the SKI kernel matrix,

$$WK_{UU}W^\top + \sigma^2 I = (Q_d V_d \Lambda_d^{1/2})(\Lambda_d^{1/2} V_d^\top Q_d^\top), \tag{A.3}$$

$S = K_{UU}W^\top Q_d^\top V_d^\top \Lambda_d^{-1/2}$, and Eq. A.2 is modified to

$$\sigma^2_{f|\mathcal{D}}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - w(\mathbf{x}^{(i)})^\top SS^\top w(\mathbf{x}^{(j)}). \tag{A.4}$$

[8] Q_d is conventionally used to denote the d-rank Lanczos basis.

A.2 Deriving the WISKI Predictive Mean and Variance

In this section we derive the Woodbury Inverse SKI predictive distributions. In contrast to Pleiss et al. (2018), the WISKI predictive mean and covariance can be formulated in terms of quantities that can be cached in $\mathcal{O}(m^2)$ space and updated with new observations in constant time. Recall the form of the Woodbury SKI inverse, $M := (\sigma^2 K_{UU}^{-1} + W^\top W)^{-1}$. For both the predictive mean and variance we begin with the standard SKI form, and show how to derive the WISKI form.

Predictive Mean

$$\begin{aligned}
\mu_{f|\mathcal{D}}(\mathbf{x}^*) &= w(\mathbf{x}^*)^\top K_{UU} W^\top (W K_{UU} W^\top + \sigma^2 I)^{-1} \mathbf{y}, \\
&= w(\mathbf{x}^*)^\top (I + \sigma^{-2} K_{UU} W^\top W)^{-1} (\sigma^{-2} K_{UU}) W^\top \mathbf{y}, \\
&= w(\mathbf{x}^*)^\top \left((\sigma^{-2} K_{UU})(\sigma^2 K_{UU}^{-1} + W^\top W)\right)^{-1} (\sigma^{-2} K_{UU}) W^\top \mathbf{y}, \\
&= w(\mathbf{x}^*)^\top (\sigma^2 K_{UU}^{-1} + W^\top W)^{-1} K_{UU}^{-1} K_{UU} W^\top \mathbf{y}, \\
&= w(\mathbf{x}^*)^\top M W^\top \mathbf{y}. \\
&= w(\mathbf{x}^*)^\top (\sigma^{-2} K_{UU} W^\top \mathbf{y} - \sigma^{-2} K_{UU} L (I + \sigma^{-2} L^\top K_{UU} L)^{-1} L^\top K_{UU} W^\top \mathbf{y}.
\end{aligned}$$

The second line follows from the push-through identity (a special case of the Woodbury matrix identity).

Predictive Covariance The low-rank SKI predictive covariance of a GP is given (elementwise) by

$$\begin{aligned}
\sigma^2_{f|\mathcal{D}}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) &= w(\mathbf{x}^{(i)})^\top K_{UU} w(\mathbf{x}^{(j)}) - w(\mathbf{x}^{(i)})^\top K_{UU} W^\top (W K_{UU} W^\top + \sigma^2 I)^{-1} W K_{UU} w(\mathbf{x}^{(j)}) \\
&= \sigma^2 w(\mathbf{x}^{(i)})^\top \left(\sigma^{-2} K_{UU} - (\sigma^{-2} K_{UU}) W^\top (W (\sigma^{-2} K_{UU}) W^\top + I)^{-1} W (\sigma^{-2} K_{UU})\right) w(\mathbf{x}^{(j)}) \\
&= \sigma^2 w(\mathbf{x}^{(i)})^\top (\sigma^2 K_{UU}^{-1} + W^\top W)^{-1} w(\mathbf{x}^{(j)}), \\
&= \sigma^2 w(\mathbf{x}^{(i)})^\top M w(\mathbf{x}^{(j)}) \\
&= w(\mathbf{x}^{(i)})^\top \left(K_{UU} - K_{UU} L (I + \sigma^{-2} L^\top K_{UU} L)^{-1} L^\top K_{UU}\right) w(\mathbf{x}^{(j)})
\end{aligned}$$

The third line immediately follows from an application of the Woodbury matrix identity to $(\sigma^2 K_{UU}^{-1} + W^\top W)^{-1}$. Following Pleiss et al. (2018), we may compute two root decompositions of the form $BB^\top = K_{UU}$ and $DD^\top \approx (I + \sigma^{-2} L^\top K_{UU} L)^{-1}$ to speed up predictive variance computation, as this yields efficient diagonal computation:

$$\sigma^2_{f|\mathcal{D}}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = w(\mathbf{x}^{(i)})^\top \left(BB^\top - BB^\top L D D^\top L^\top B B^\top\right) w(\mathbf{x}^{(j)}).$$

As we noted in the main text, at time $t+1$, Q_t can be updated with a new observation in constant time via a Sherman-Morrison update, and $(W^\top \mathbf{y})_{t+1} = (W^\top \mathbf{y})_t + y_{t+1} w(\mathbf{x}_{t+1})$.

A.3 Conditioning on New Observations

Updating the Marginal Likelihood For fixed n, the Woodbury version of the log likelihood in WISKI is constant in n after an initial $\mathcal{O}(n)$ precomputation of $W^\top W$, as the only terms that ever get updated are the scalar σ^2 and the $m \times m$ matrix K_{UU}^{-1}; this is in and of itself an advance over the computation speeds of other Gaussian process models, including SKI (Wilson and Nickisch, 2015).

Updating $W^\top W$. To update $W^\top W$ as we see new data points, we follow the general strategy of Gill et al. (1974) by performing rank-one updates to root decompositions:

$$\begin{aligned}
\tilde{A} = A + zz^\top &= L(I + pp^\top)L^\top, & p &= L^{-\top} z, \\
&= LBB^\top L^\top = \tilde{L}\tilde{L}^\top, & BB^\top &= I + pp^\top, \tilde{L} = LB
\end{aligned}$$

In our setting, the update is given by

$$(W^\top W)_{t+1} = (W^\top W)_t + \mathbf{w}_{\mathbf{x}_{t+1}} \mathbf{w}_{\mathbf{x}_{t+1}}^\top.$$

For full generality, we will assume q new points come at once, making $\mathbf{w}_{\mathbf{x}_{t+1}} \in \mathbb{R}^{m \times q}$. Recall that $JJ^\top = (W^\top W)^+$, let $\mathbf{p} = J_t{}^\top \mathbf{w}_{\mathbf{x}_{t+1}}$, which is the product of a $r \times m$ matrix and a $m \times q$ matrix, which costs $\mathcal{O}(mrq)$. To compute the decomposition $BB^\top = I_r + \mathbf{p}\mathbf{p}^\top$ in a numerically stable fashion, we compute the SVD of $\mathbf{p} = USV^\top$ and use it to update the root decomposition:

$$I_r + \mathbf{p}\mathbf{p}^\top = I_r + USV^\top VSU^\top = U\text{diag}((S_{ii}^2 + 1); \mathbf{1}_{r-q})U^\top = U\text{diag}(\sqrt{S_{ii}^2 + 1}, \mathbf{1}_{r-q})\text{diag}(\sqrt{S_{ii}^2 + 1}, \mathbf{1}_{r-q})U^\top.$$

The SVD of this matrix costs $\mathcal{O}(q^2 r)$, assuming $q < r$ ($q = 1$ for most applications). The inner root is $B = U\text{diag}(\sqrt{S_{ii}^2 + 1}, \mathbf{1}_{r-q})$, and a final matrix multiplication costing $\mathcal{O}(mr^2)$ obtains the expression for the updated root $L_{t+1} = L_t B$. The updated inverse root is obtained similarly by $J_{t+1} = J_t U\text{diag}(1./\sqrt{S_{ii}^2 + 1}, \mathbf{1}_{r-q})$ The overall computation cost is then $\mathcal{O}(mrq + q^2 r + mr^2)$.

A.4 Online SKI and Deep Kernel Learning

When combining deep kernel learning (DKL) and SKI, the interpolation weight vectors $\mathbf{w}_i = w(\mathbf{x}_i, \mathbf{u}_1, \ldots, \mathbf{u}_m)$ become $\mathbf{w}_i = w(h(\mathbf{x}_i; \phi), \mathbf{u}_1, \ldots, \mathbf{u}_m)$, where $h(\cdot; \phi)$ is a feature map parameterized by ϕ. One implication of this change is that if the parameters of h change, then the interpolation weights must change as well. In the batch setting, the features and associated interpolation weights can be recomputed after every optimization iteration, since the cost of doing so is negligible compared to the cost of computing the MLL. The online setting does not admit the recomputation of past features and interpolation weights, because doing so would require $\mathcal{O}(n)$ work. Hence at any time t we must consider the previous features and interpolation weights $(\mathbf{h}_1, \mathbf{w}_1), \ldots, (\mathbf{h}_{t-1}, \mathbf{w}_{t-1})$ to be fixed. As a result, when computing the gradient of the MLL w.r.t. ϕ, we need only consider the terms that depend on $\mathbf{w}_t$.

Claim:

$$\nabla_{\mathbf{w}_t} \log p(\mathbf{y}_t | \mathbf{x}_{1:t}, \theta) = \nabla_{\mathbf{w}_t} \frac{1}{2\sigma^2} \left(\mathbf{y}_t^\top W_t M_{t-1} W_t^\top \mathbf{y}_t - \frac{1}{1 + \mathbf{v}_t^\top \mathbf{w}_t} \left(\mathbf{v}_t^\top W_t^\top \mathbf{y}_t \right)^2 \right) - \frac{1}{2} \log(1 + \mathbf{v}_t \mathbf{w}_t), \tag{A.5}$$

where

$$\mathbf{w}_t = w(h(\mathbf{x}_t; \phi), \mathbf{u}_{1:m}), \qquad \mathbf{v}_t = M_{t-1}\mathbf{w}_t.$$

Proof:

$$\log p(\mathbf{y}_t | \mathbf{x}_{1:t}, \theta) = -\frac{1}{2\sigma^2}(\mathbf{y}_t \mathbf{y}_t^\top - \mathbf{y}_t^\top W_t M_t W_t^\top \mathbf{y}_t) - \frac{1}{2} \left(\log |K_{UU}| - \log |M_t| + (n - m) \log \sigma^2 \right) - \frac{t}{2} \log 2\pi.$$

Recalling $M_t := (K_{uu}^{-1} + W_t^\top W_t)^{-1} = (K_{uu}^{-1} + W_{t-1}^\top W_{t-1} + \mathbf{w}_t \mathbf{w}_t^\top)^{-1}$, by the Sherman-Morrison identity we have

$$M_t = M_{t-1} - \frac{1}{1 + \mathbf{v}_t^\top \mathbf{w}_t} \mathbf{v}_t \mathbf{v}_t^\top. \tag{A.6}$$

Since M_{t-1} is constant w.r.t. $\mathbf{w}_t$, we can substitute Eq. (A.6) into the MLL and differentiate w.r.t. $\mathbf{w}_t$ to obtain

$$\nabla_{\mathbf{w}_t} \log p(\mathbf{y}_t | \mathbf{x}_{1:t}, \theta) = \nabla_{\mathbf{w}_t} \frac{1}{2\sigma^2} \mathbf{y}_t^\top W_t (M_{t-1} - \frac{1}{1 + \mathbf{v}_t^\top w_t} v_t v_t^\top) W_t^\top \mathbf{y}_t + \frac{1}{2} \log |M_{t-1} - \frac{1}{1 + \mathbf{v}_t^\top w_t} \mathbf{v}_t \mathbf{v}_t^\top|$$

The quadratic term straightforwardly simplifies to the first two terms in Eq. (A.5). The final term results from an application of the matrix-determinant identity, once again dropping any terms with no dependence on $\mathbf{w}_t$,

$$\begin{aligned}
\log |M_{t-1} - \frac{1}{1 + \mathbf{v}^\top \mathbf{w}} \mathbf{v}\mathbf{v}^\top| &= \log |M_{t-1}| - \log \left(1 + \frac{1}{1 + \mathbf{v}^\top \mathbf{w}} \mathbf{v}^\top M_{t-1}^{-1} \mathbf{v} \right) \\
&= \log |M_{t-1}| - \log \left(1 + \frac{1}{1 + \mathbf{v}^\top \mathbf{w}} \mathbf{v}^\top \mathbf{w} \right) \\
&= \log |M_{t-1}| - \log \left(1 + \mathbf{v}^\top \mathbf{w} \right). \qquad \blacksquare
\end{aligned}$$

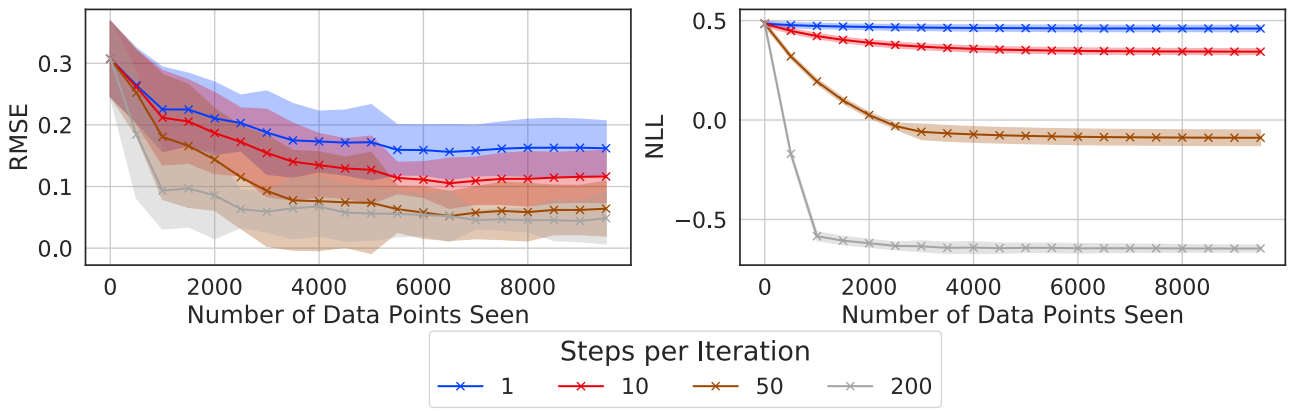


Figure A.1: (**Left:**) MSEs through the course of the dataset stream for up to 10,000 data points coming in batches of 500 data points for online SVGP. We varied the number of optimization steps per batch, finding that at least 10 steps were required to achieve good performance. The data points are drawn from a synthetic sine function corrupted by Gaussian noise. (**Right:**) NLLs over the course of the dataset stream; again, we see that many optimization steps are needed to decrease the NLL on the test set over the course of the stream.

A.5 Heteroscedastic Fixed Gaussian Noise Likelihoods and Dirichlet Classification

For a *fixed* noise term, the Woodbury identity still holds and we can still perform the updates in constant time. For fixed Gaussian noise, the term training covariance becomes

$$K_{XX} \approx K_{SKI} = W K_{UU} W + D,$$
$$K_{SKI}^{-1} = D^{-1} - D^{-1} W (K_{UU}^{-1} + W^\top D^{-1} W)^{-1} W^\top D^{-1}.$$

Plugging the second line into Eq. 13 tells us immediately that we need to store $\mathbf{y} D^{-1} \mathbf{y}$ instead of $\mathbf{y}\mathbf{y}$, $W^\top D^{-1} W$ instead of $W^\top W$, $W^\top D^{-1} \mathbf{y}$ instead of $W^\top \mathbf{y}$. The rest of the online algorithm proceeds in the same manner as at each step, we update these caches with new vectors.

The heteroscedastic fixed noise regression approach naturally allows us to perform GP inference as in Milios et al. (2018). Given a one-hot encoding of the class probabilities, e.g. $y = e_c$ where c is the class number, they derive an approximate likelihood so that the transformed regression targets are

$$\tilde{y}_i = \log \alpha_i - \tilde{\sigma}_i^2 / 2, \qquad \tilde{\sigma}_i^2 = \log(1 + 1/\alpha_i),$$

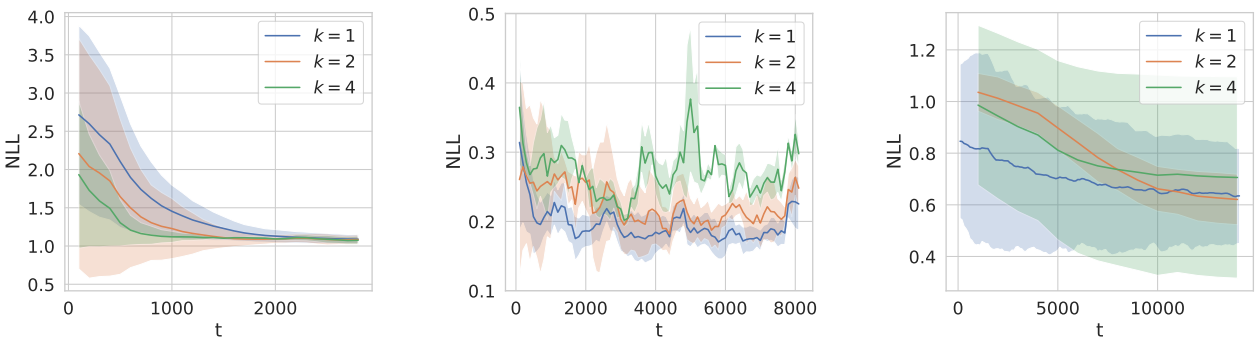
where $\alpha_i = I_{y_i=1} + \alpha_\epsilon$, where α_ϵ is a tuning parameter. We use $\alpha_\epsilon = 0.01$ in our classification experiments. As there are C classes, we must model each class regression target; Milios et al. (2018) use independent GPs to model each class as we do. The likelihood at each data point over each class target is then $p(\tilde{y}_i | \mathbf{f}) = \mathcal{N}(f_i, \tilde{\sigma}_i^2)$, which is simply a heteroscedastic fixed noise Gaussian likelihood. Posterior predictions are given by computing the arg max of the posterior mean, while posterior class probabilities can be computed by sampling over the posterior distribution and using a softmax (Equation 8 of Milios et al. (2018)).

B CHALLENGES OF STREAMING VARIATIONAL INFERENCE FOR GPS

B.1 A Closer Look at O-SVGP

In this paper, we focus primarily on the online SVGP objective of Bui et al. (2017), ignoring for the moment their α-divergence objective that is used in some of their models — which can itself be viewed as a type of generalized variational inference (Knoblauch et al., 2019).

Recalling the online uncollapsed bound of Bui et al. (2017) and adapting their notation — copying directly from



(a) Skillcraft (b) Powerplant (c) Elevators

Figure A.2: Here we ablate the number of O-SVGP gradient updates per timestep in the context of UCI regression. Notably we see that the value we chose for our comparison in the main text ($k = 1$) performs well in comparison to larger values of k. In contrast to the results in Figure A.1, there is relatively little benefit to increasing the number of gradient update steps per batch when the batch size is very small (in our case, 1).

their appendix, the objective becomes

$$\mathcal{F}\left(q_{\text{new}}\left(f\right)\right) = \int \mathrm{d}f\, q_{\text{new}}\left(f\right)\left[\log \frac{p\left(\mathbf{a}|\theta_{\text{old}}\right) q_{\text{new}}\left(\mathbf{b}\right)}{p\left(\mathbf{b}|\theta_{\text{new}}\right) q_{\text{old}}\left(\mathbf{a}\right) p\left(\mathbf{y}_{\text{new}}|f\right)}\right] \tag{A.7}$$
$$= -\mathbb{E}_{q_{\text{new}}\left(f\right)}(\log p\left(\mathbf{y}_{\text{new}}|f\right)) + \mathrm{KL}\left(q(\mathbf{b})\|p\left(\mathbf{b}|\theta_{\text{new}}\right)\right) + \mathrm{KL}\left(q_{\text{new}}\left(\mathbf{a}\right)\|q_{\text{old}}\left(\mathbf{a}\right)\right) - \mathrm{KL}\left(q_{\text{new}}\left(\mathbf{a}\right)\|p\left(\mathbf{a}|\theta_{\text{old}}\right)\right),$$

where $\mathbf{a}$ is the old set of inducing points, $\mathbf{b}$ is the new set of inducing points, $q(.)$ is the variational posterior on a set of points, θ_{new} are the current hyperparameters to the GP, and θ_{old} are the hyperparameters for the GP at the previous iteration. In Eq. A.7, the first two terms are the standard SVI-GP objective (e.g from (Hensman et al., 2013)), while the second two terms add to the standard objective allowing SVI to be applied to the streaming setting. Mini-batching can be achieved without knowing the number of data points a priori; however, this achievement comes at the expense of having to compute two new terms in the loss.

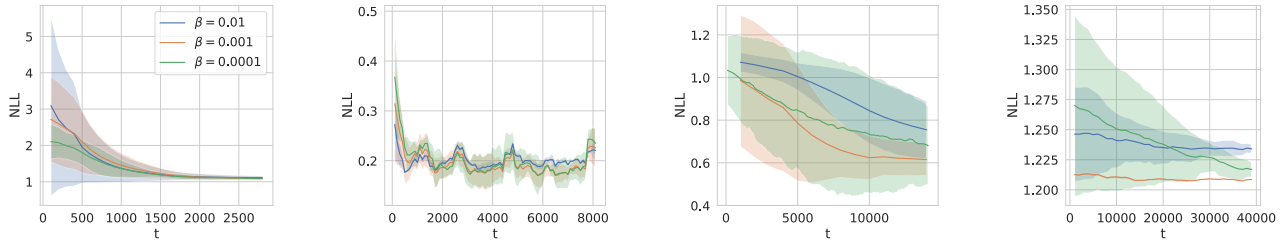
Since the bound in Eq. A.7 is uncollapsed, it must be optimized to a global maximum at every timestep to ensure both the GP hyperparameters and the variational parameters are at their optimal values. As noted in Bui et al. (2017), this optimization is extremely difficult in the streaming setting for two main reasons. 1) Observations may arrive in a non-iid fashion and violate the assumptions of SVI, and 2) each observation batch is seen once and discarded, preventing multiple passes through the full dataset, as is standard practice for SVI. Even in the batch setting, optimizing the SVI objective to a global maximum is notoriously difficult due to the proliferation of local maxima. This property makes a fair timing comparison with our approach somewhat difficult in that multiple gradient steps per data point will necessarily be slower than WISKI, which does not have any variational parameters to optimize, and thus can learn with fewer optimization steps per observation. We implemented Eq A.7 in GPyTorch (Gardner et al., 2018), but additionally attempted to use the authors' provided implementation of O-SVGP[9] finding similar results — many gradient steps are required to reduce the loss. As a demonstration, we varied the number of steps of optimization per batch in Figure A.1 with a large batch size 300 (the same as Bui et al. (2017)'s own experiments) in the online regression setting on synthetic sinusoidal data, finding that at least 10 optimization steps were needed to decrease the RMSE in a reasonable manner even on this simpler problem.

To remedy the convergence issue (and to create a fair comparison with *one* gradient step per batch of data), we down-weighted the KL divergence terms, producing a generalized variational objective (equivalent to taking the likelihood to a power $1/\beta$) (Knoblauch et al., 2019). Eq A.7 loss becomes:

$$\begin{aligned}\mathcal{F} = &-\mathbb{E}_{q_{\text{new}}\left(f\right)}(\log p\left(\mathbf{y}_{\text{new}}|f\right)) + \beta\mathrm{KL}\left(q(\mathbf{b})\|p\left(\mathbf{b}|\theta_{\text{new}}\right)\right) \\ &+ \beta\mathrm{KL}\left(q_{\text{new}}\left(\mathbf{a}\right)\|q_{\text{old}}\left(a\right)\right) - \beta\mathrm{KL}\left(q_{\text{new}}\left(\mathbf{a}\right)\|p\left(\mathbf{a}|\theta_{\text{old}}\right)\right),\end{aligned} \tag{A.8}$$

where all terms are as before except for $\beta < 1$. We found that $\beta << 1$ produces more reasonable results for a batch size of 1. Ablations for varying this hyperparameter are shown in Figure A.3. Using generalized variational

[9] `https://github.com/thangbui/streaming_sparse_gp`



(a) Skillcraft (b) Powerplant (c) Elevators (d) Protein

Figure A.3: Here we ablate the β hyperparameter in the GVI loss for O-SVGP on the UCI datasets considered in this paper. While there is not a clear winner, we find that β = 1e-3 works well for all datasets.

inference does not change the complexity of the streaming objective, which remains $\mathcal{O}(Bm^2 + m^3)$; the $\mathcal{O}(m^3)$ term stays the same due to the log determinant term in the KL objective.

Finally, we vary the number of inducing points in the O-SVGP bound in Figure A.4, finding that O-SVGP is quite sensitive to the number of inducing points.

C EXPERIMENTAL DETAILS

C.1 Regression and Classification

Algorithm 1: Online Learning with WISKI

Input: Kernel function k, inducing grid U, initial data $\mathbf{x}_{1:n}, \mathbf{y}_n$, GP parameters θ, feature map parameters ϕ, learning rate η.

Initialize $K_{UU}, L, W^\top \mathbf{y}_n, \mathbf{y}_n^\top \mathbf{y}_n$.

for $t = n+1, n+2, \ldots$ **do**

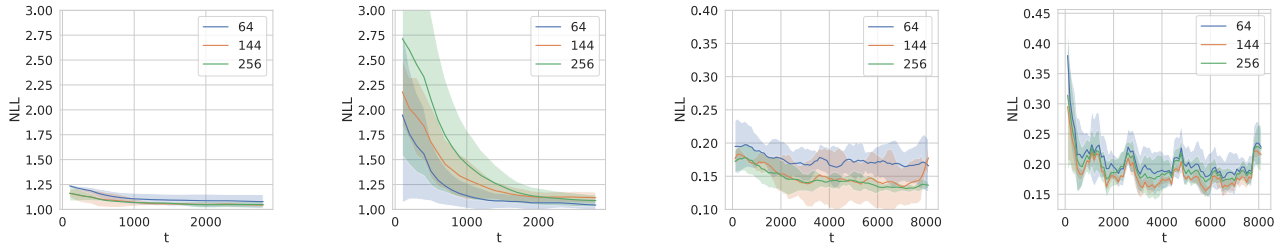
- Receive $\mathbf{x}_t$.
- Predict $\hat{p}(y_t | \mathbf{x}_{1:t}, \mathbf{y}_{t-1}, \theta, \phi)$ (Eq. 14).
- Observe y_t, compute $\mathbf{w}_t$.
- Update caches $L_t, W_t^\top \mathbf{y}_t, \mathbf{y}_t^\top \mathbf{y}_t$.
- $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}(\phi)$ (Eq. 18).
- $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$ (Eq. 13).

end

Algorithm 1 summarizes online learning with WISKI. If an input projection is not learned, then $h(\mathbf{x}; \phi)$ can be taken to be the identity map, and the projection parameter update is consequently a no-op. In the rest of this section we provide additional experimental results in the regression and classification setting. In Figure 3 we report the RMSE for each of the UCI regression tasks. Note that the RMSE is computed on the standardized labels. The qualitative behavior is identical to that of the NLL plots in the main text (Figure 3). Figure A.5 is a visualization of a WISKI classifier on non-i.i.d. data. Figures A.4 and A.3 report the results of our ablations on m and β, respectively. This section provides all necessary implementation details to reproduce our results.

Data Preparation For all datasets, we scaled input data to lie in $[-1, 1]^d$. For regression datasets we standardized the targets to have zero mean and unit variance. If the raw dataset did not have a train/test split, we randomly selected 10% of the observations to form a test dataset. From the remaining 90% we removed an additional 5% of the observations for pretraining.

Hyperparameters We pretrained all models for T_{batch} epochs, with learning rates η_{batch}. If we learned a projection of the inputs, we used a lower learning rate for the projection parameters. While a small learning rate worked well for all tasks, for the best performance we used a higher learning rate for easier tasks (Table C.1).



(a) WISKI, Skillcraft (b) O-SVGP, Skillcraft (c) WISKI, Powerplant (d) O-SVGP, Powerplant

Figure A.4: Here we ablate the number of inducing points for both WISKI and O-SVGP. We find that WISKI is not very sensitive to the number of inducing points, but always improves if more inducing points are added. O-SVGP sometimes performs better with fewer inducing points, a phenomenon we attribute to either 1) poor optimization of the GVI objective or 2) overfitting due to the downweighted KL terms in the GVI objective. In theory adding inducing points should only improve the performance of an SVGP. This observation highlights the difficulties O-SVGP often encounters in practice.

m	r	NLL
256	128	8.2e+6 $\pm$ 9.8e+6
256	192	1.000 $\pm$ 0.010
256	256	1.007 $\pm$ 0.015
1024	256	2.9e+7 $\pm$ 9.2e+7
1024	512	1.050 $\pm$ 0.082
1024	768	0.995 $\pm$ 0.007
1024	1024	1.007 $\pm$ 0.008

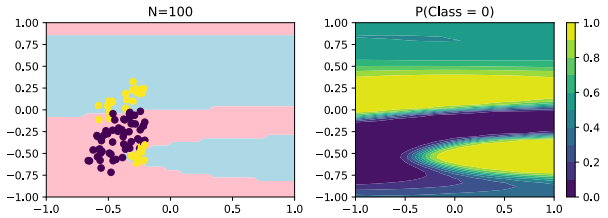
Table 1: Root rank (r) ablation by NLL across both $m = 256$ and $m = 1024$ inducing points on skillcraft. Too small of a rank fails to converge; however, once r is large enough (about $m/2$), the performance is unchanged.

Task	m	T_{batch}	$\eta_{\text{batch}}(\theta)$	$\eta_{\text{batch}}(\phi)$	$\eta_{\text{online}}(\theta)$	$\eta_{\text{online}}(\phi)$	β
Banana	256	200	5e-2	-	5e-3	-	1e-3
SVM Guide 1	256	200	5e-2	-	5e-3	-	1e-3
Skillcraft	256	200	5e-2	5e-3	5e-3	5e-4	1e-3
Powerplant	256	200	5e-2	5e-3	5e-3	5e-4	1e-3
Elevators	256	200	1e-2	1e-3	1e-3	1e-4	1e-3
Protein	256	200	1e-2	1e-3	1e-3	1e-4	1e-3
3DRoad	1600	800	1e-2	-	1e-3	-	1e-3

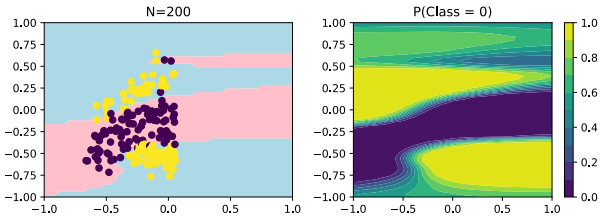
C.2 Bayesian Optimization Experimental Details and Further Results

For the Bayesian optimization experiments, we considered noisy three dimensional versions of the Bayesian optimization test functions available from BoTorch[10]. We sed the BoTorch implementation of the test functions with the *qUCB* acquisition function with $q = 3$, randomly choosing five points to initialize with and running 1500 BO steps, so that we end up with 4505 data points acquired from the models. We then followed BoTorch standard optimization of the acquisition functions by optimizing with LBFGS-B with 10 random restarts, 512 samples to initialize the optimization with, a batch limit of 5 and 200 iterations of LBFGS-B. We fit the model to convergence at each iteration as model fits are very important in BO using LBFGS-B for exact and WISKI while using Adam for OSVGP because the variational parameters are much higher dimensional so LBFGS-B is prohibitively slow. The timing results take into account the model re-fitting stage, the acquisition optimization stage, and the expense of adding a new datapoint into the model. We used a single AWS instance with eight Nvidia Tesla V100s for these experiments, running each experiment four times, except for StyblinskiTang, which we ran three times (as the exact GP ran out of memory during a bayes opt step on one of the seeds). We measure time per iteration by adding both the model fitting time and the acquisition function optimization time. While a

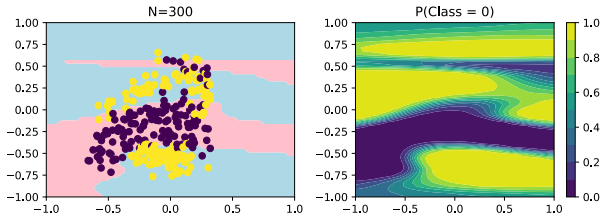
[10] `https://botorch.org/api/test_functions.html`



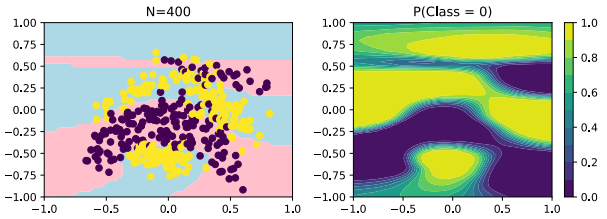
(a) Test Accuracy - 70%



(b) Test Accuracy - 70%



(c) Test Accuracy - 77%



(d) Test Accuracy - 88%

Figure A.5: Online Gaussian Process Dirichlet classification with WISKI on observations from the banana dataset arriving in non-i.i.d. fashion (shown). The WISKI classifier is updated with a single gradient step after each individual observation.

single training step is somewhat faster for O-SVGP than for WISKI, we found that it tended to take longer to optimize acquisition functions in BO loops.

Results over time per iteration and maximum achieved value by time for the rest of the test suite are shown in Figure A.6. Overall WISKI performs comparably in terms of maximum achieved value to the exact GP reaches that value in terms of quicker wallclock time. In Figure A.7, we show the maximum value achieved by iteration for each problem, finding that the exact GPs typically converge to their optimum first, while WISKI converges afterwards with OSVGP slightly after that. In Figure A.8, we show the time per iteration for all three methods, finding that WISKI is constant time throughout as is OSVGP, while the exact approach scales broadly quadratically (as expected given that the BoTorch default for sampling uses LOVE predictive variances and sampling). Digging deeper into the results, we found that the speed difference between OSVGP and WISKI is attributable to the increased predictive variance and sampling speed for OSVGP ($\mathcal{O}(m^3)$ compared to $\mathcal{O}(m^2)$).

Levy	Ackley	StyblinskiTang	Rastrigin	Griewank	Michalewicz
10.0	4.0	20.0	10.0	4.0	5.0

Table 2: Noise standard deviations used for the Bayesian optimization experiments.

C.3 Active Learning Experimental Details

For the active learning problem, we used a batch size of 6 for all models, with a base kernel that was a scaled ARD Matern-0.5 kernel, and lengthscale priors of Gamma(3, 6) and outputscale priors of Gamma(2, 0.15). For both OSVGP and WISKI, we used a grid size of 900 (30 per dimension); for OSVGP, we trained the inducing points, finding fixed inducing points did not reduce the RMSE. For O-SVGP, we used $\beta = 0.001$ and a learning rate of $1e-4$ with the Adam optimizer, while for WISKI and the exact GPs, we used a learning rate of 0.1. Here, we re-fit the models until the training loss stopped decaying, analogous to the BO experiments. The dataset can be downloaded at `https://wjmaddox.github.io/assets/data/malaria_df.hdf5`.

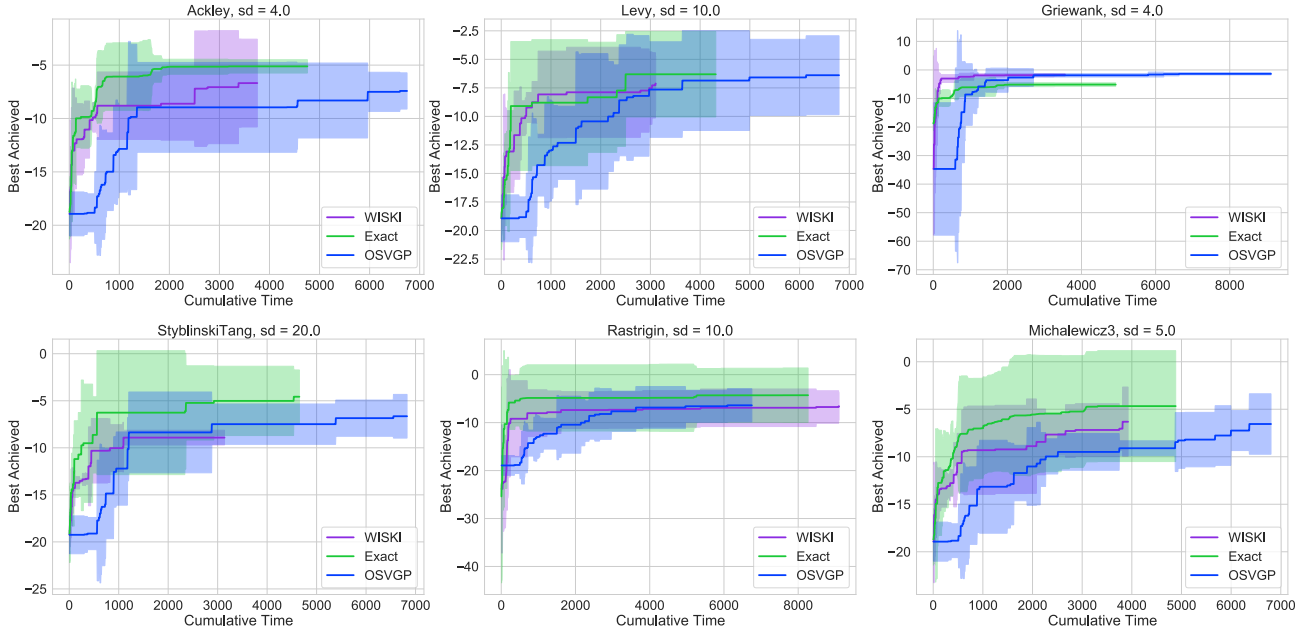


Figure A.6: Bayesian optimization results in terms of total optimization time. Throughout, WISKI is generally the fastest, except on Griewank, while reaching similar optimization performance to the exact GPs across the board. WISKI is somewhat better but significantly faster than the other methods on Griewank, but with similar performance to O-SVGP on StyblinskiTang and Michalewicz.

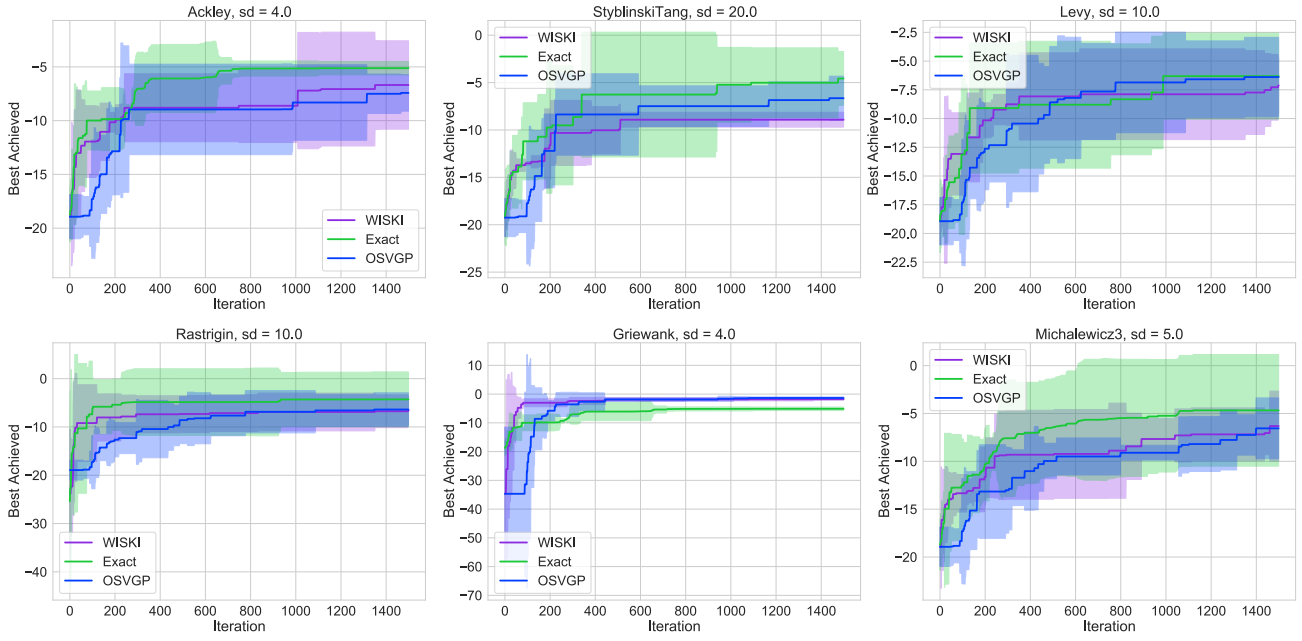


Figure A.7: Bayesian optimization results in terms of iteration complexity for noisy $3D$ test functions. Throughout, WISKI performs comparably to the exact GP.

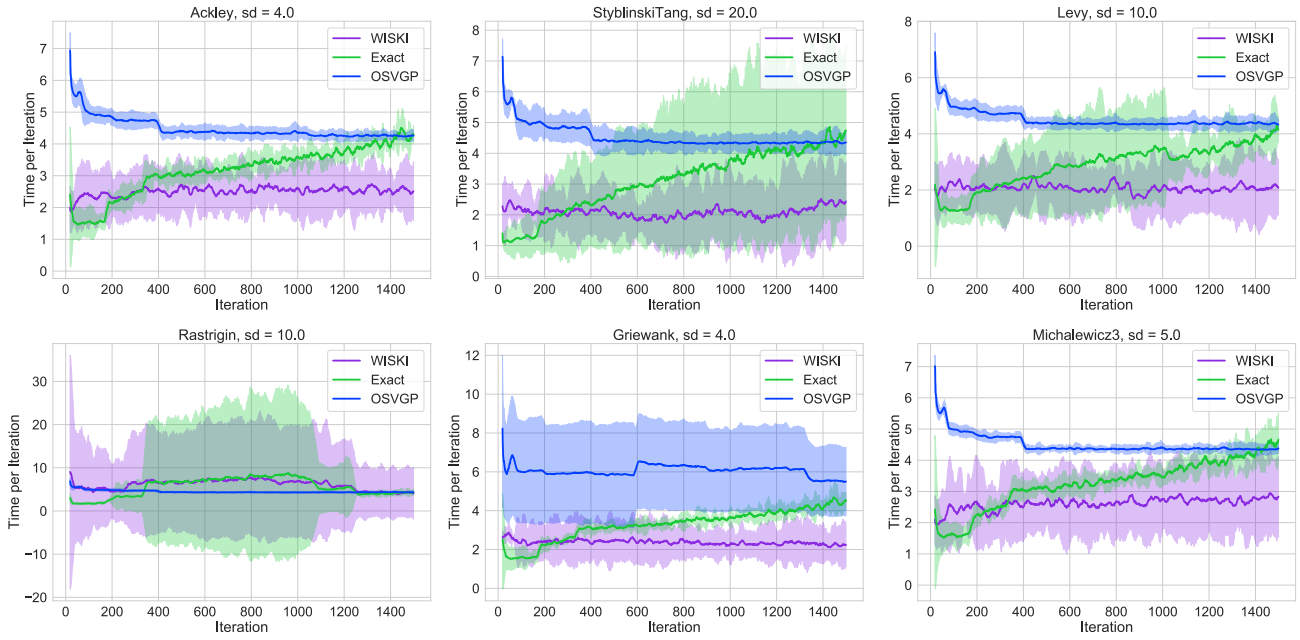


Figure A.8: Bayesian optimization results in terms of time complexity for the noisy $3D$ test functions. WISKI is the fastest method on the problems, while the exact GPs increase time per iteration at least linearly (they use LOVE predictive variances internally). While OSVGP is constant time, it typically is somewhat slower due to larger constants with respect to n, m^3 versus m^2 for WISKI.