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 $\mathbf{w}_t$,

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

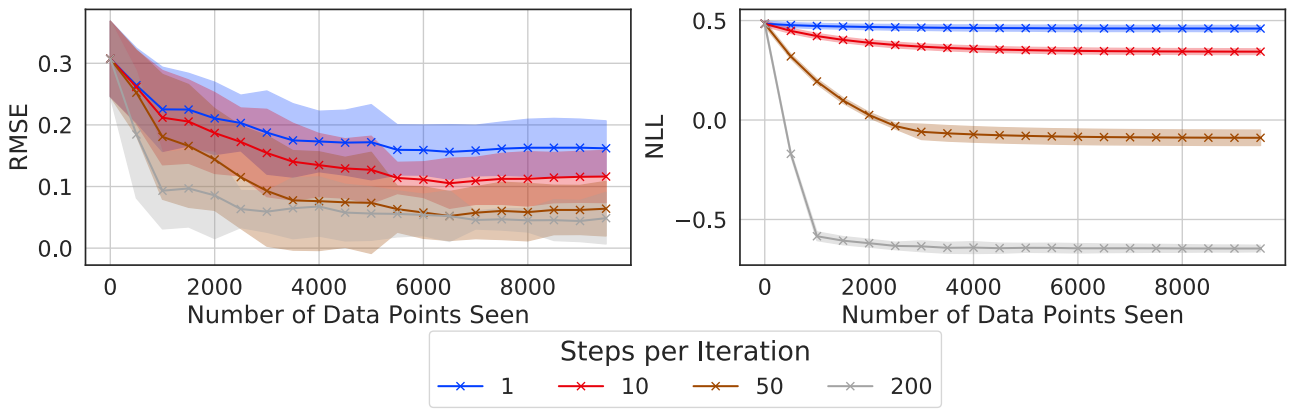


Figure A.1: (**Left:**) MSEs through the course of the dataset stream for up to 10,000 data points coming in batches of 500 data points for online SVGP. We varied the number of optimization steps per batch, finding that at least 10 steps were required to achieve good performance. The data points are 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

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

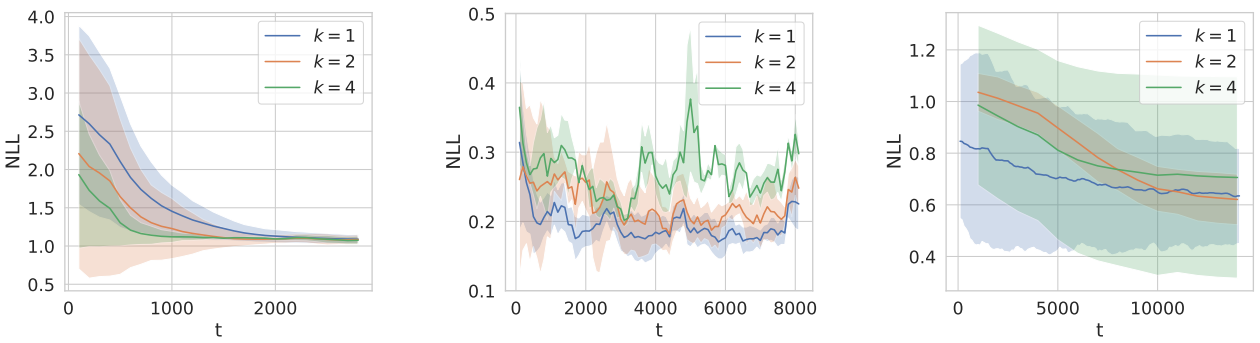
where $\alpha_i = I_{y_i=1} + \alpha_\epsilon$, where α_ϵ is a tuning parameter. We use $\alpha_\epsilon = 0.01$ in our classification experiments. As there are C classes, we must model each class regression target; Milios et al. (2018) use independent GPs to model each class as we do. The likelihood 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.

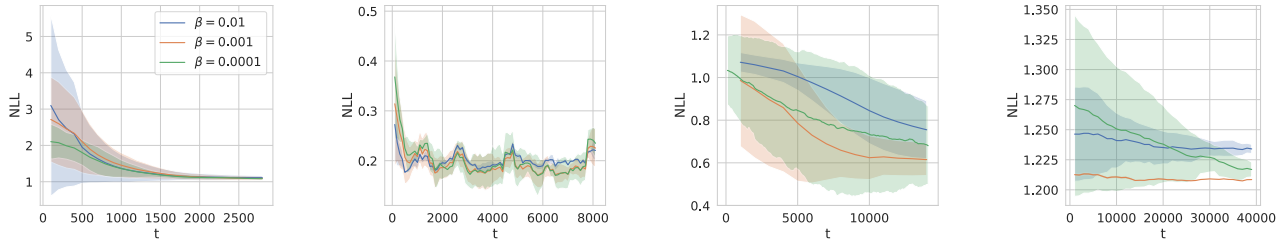
Since the bound in Eq. A.7 is uncollapsed, it must be optimized to a global maximum at every timestep to ensure both the GP hyperparameters and the variational parameters are at their optimal values. As noted in Bui et al. (2017), this optimization is extremely difficult in the streaming setting for 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**

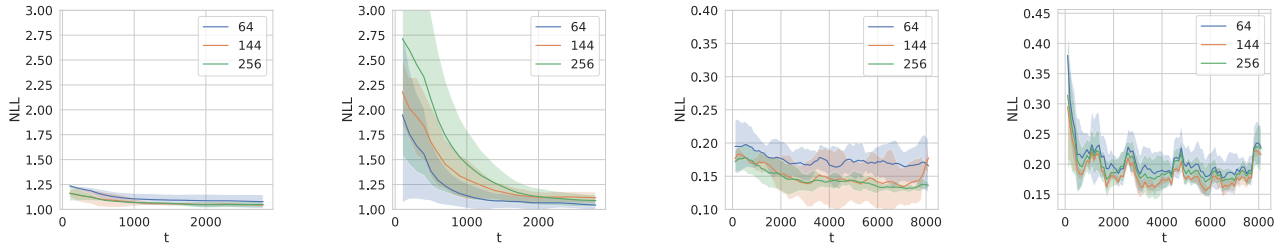
- Receive $\mathbf{x}_t$.
- Predict $\hat{p}(y_t | \mathbf{x}_{1:t}, \mathbf{y}_{t-1}, \theta, \phi)$ (Eq. 14).
- Observe y_t, compute $\mathbf{w}_t$.
- Update caches $L_t, W_t^\top \mathbf{y}_t, \mathbf{y}_t^\top \mathbf{y}_t$.
- $\phi \leftarrow \phi - \eta \nabla_\phi \mathcal{L}(\phi)$ (Eq. 18).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

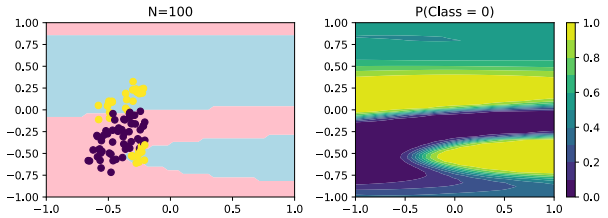
Table 1: Root rank (r) ablation by NLL across both $m = 256$ and $m = 1024$ inducing points on skillcraft. Too small of a rank fails to converge; however, once r is large enough (about $m/2$), the performance is unchanged.

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

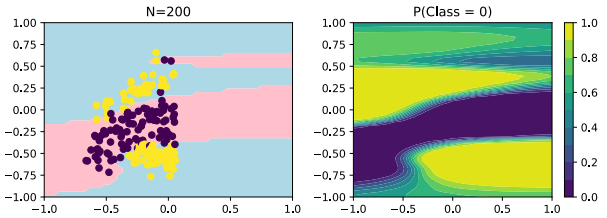
C.2 Bayesian Optimization Experimental Details and Further Results

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

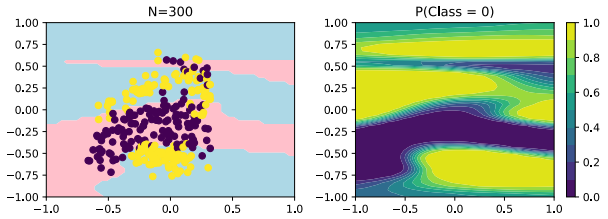
[10] `https://botorch.org/api/test_functions.html`



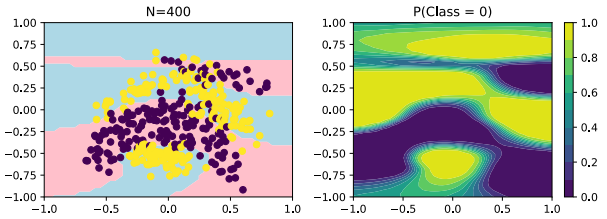
(a) Test Accuracy - 70%



(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`.

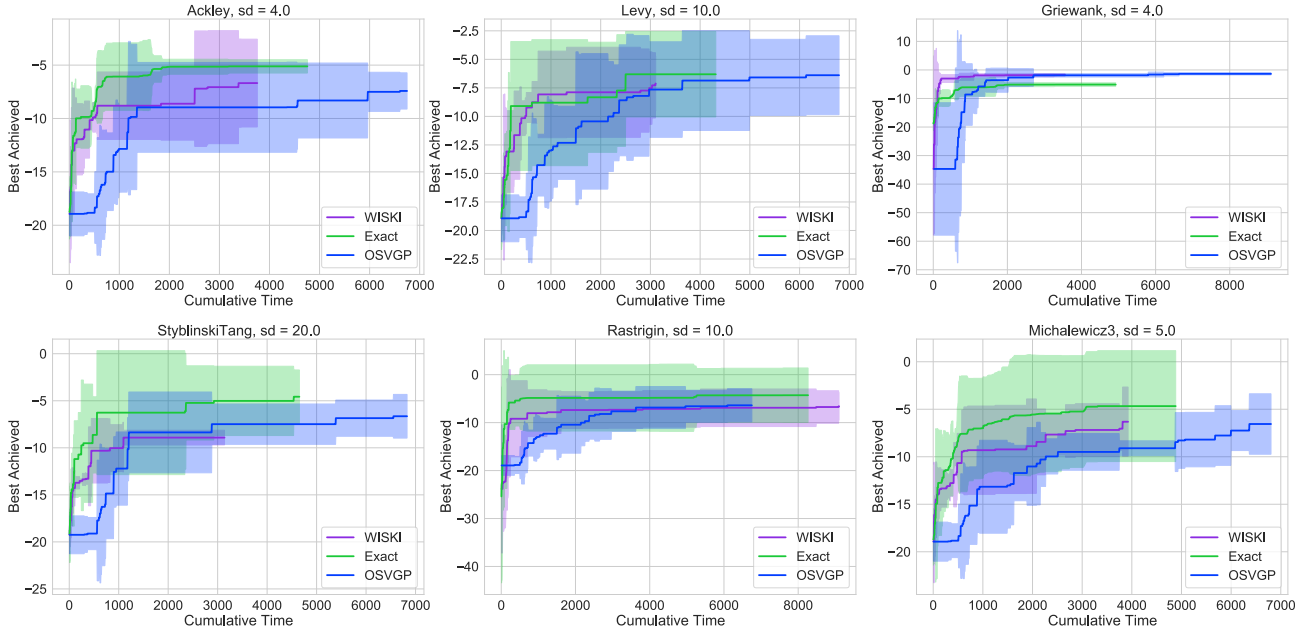


Figure A.6: Bayesian optimization results in terms of total optimization time. Throughout, WISKI is generally the fastest, except on Griewank, while reaching similar optimization performance to the exact GPs across the board. WISKI is somewhat better but significantly faster than the other methods on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

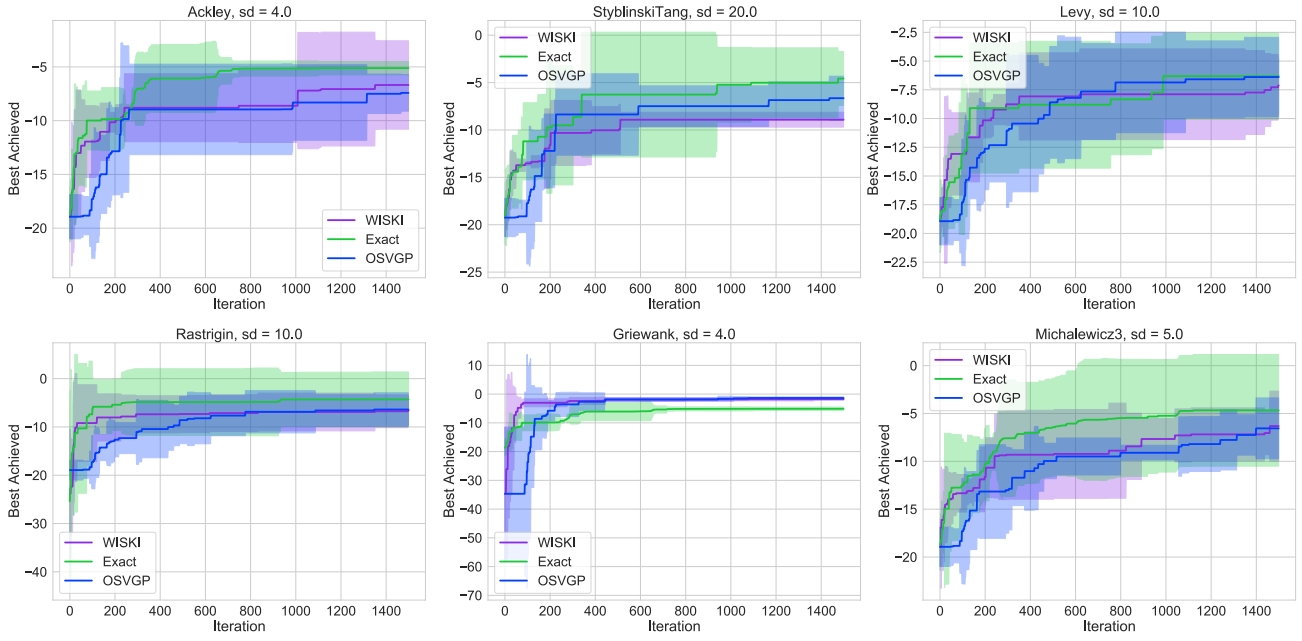


Figure A.7: Bayesian optimization results in terms of iteration complexity for noisy $3D$ test functions. Throughout, WISKI performs comparably to the exact GP.

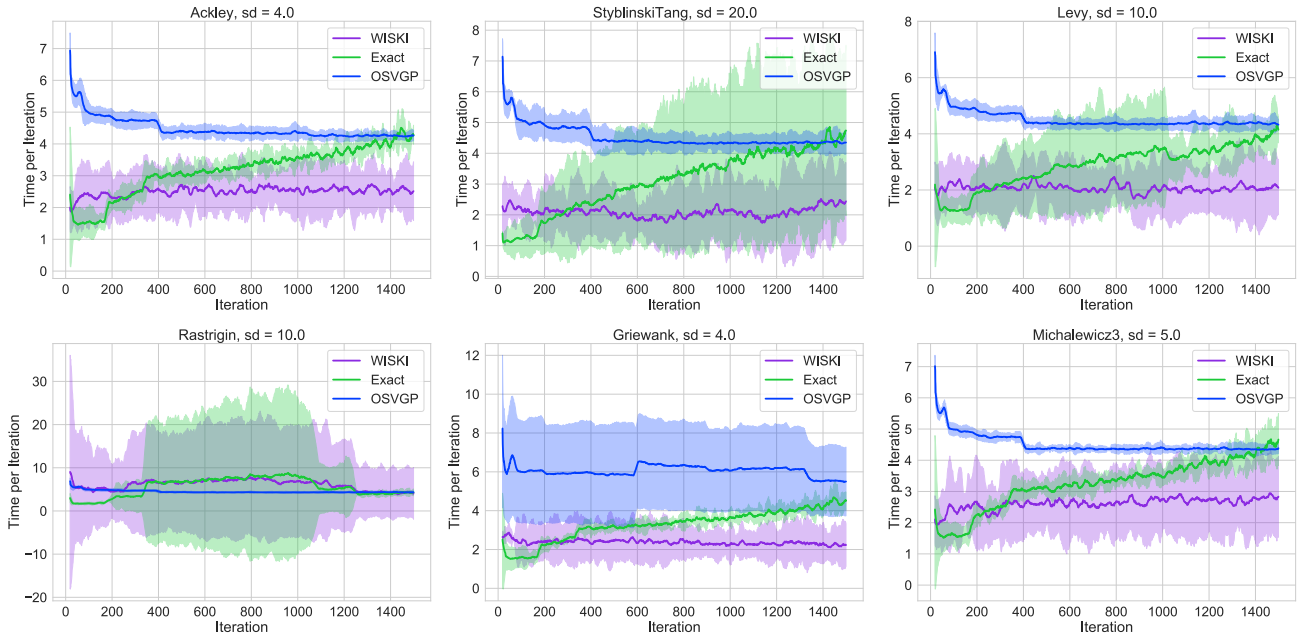


Figure A.8: Bayesian optimization results in terms of time complexity for the noisy $3D$ test functions. WISKI is the fastest method on the problems, while the exact GPs increase time per iteration at least linearly (they use LOVE predictive variances internally). While OSVGP is constant time, it typically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