SKIing on Simplices

Kernel Interpolation on the Permutohedral Lattice for Scalable Gaussian Processes

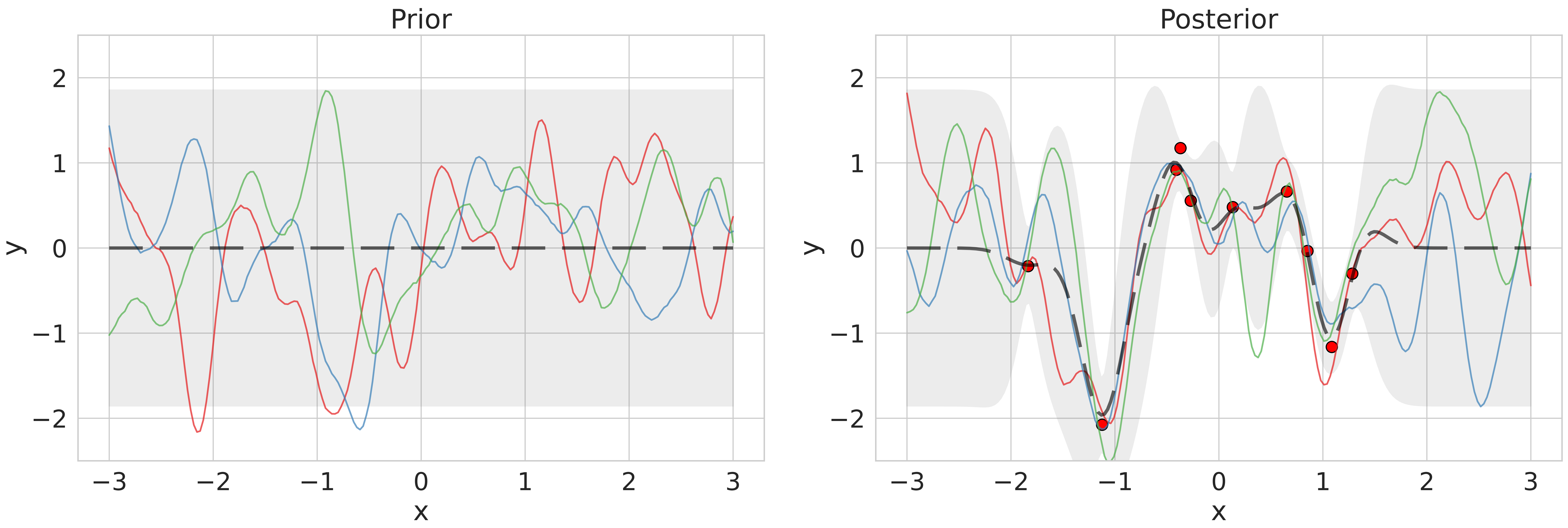
Sanyam Kapoor*, Marc Finzi*, Ke Alexander Wang, Andrew Gordon Wilson

NEW YORK UNIVERSITY

Stanford University

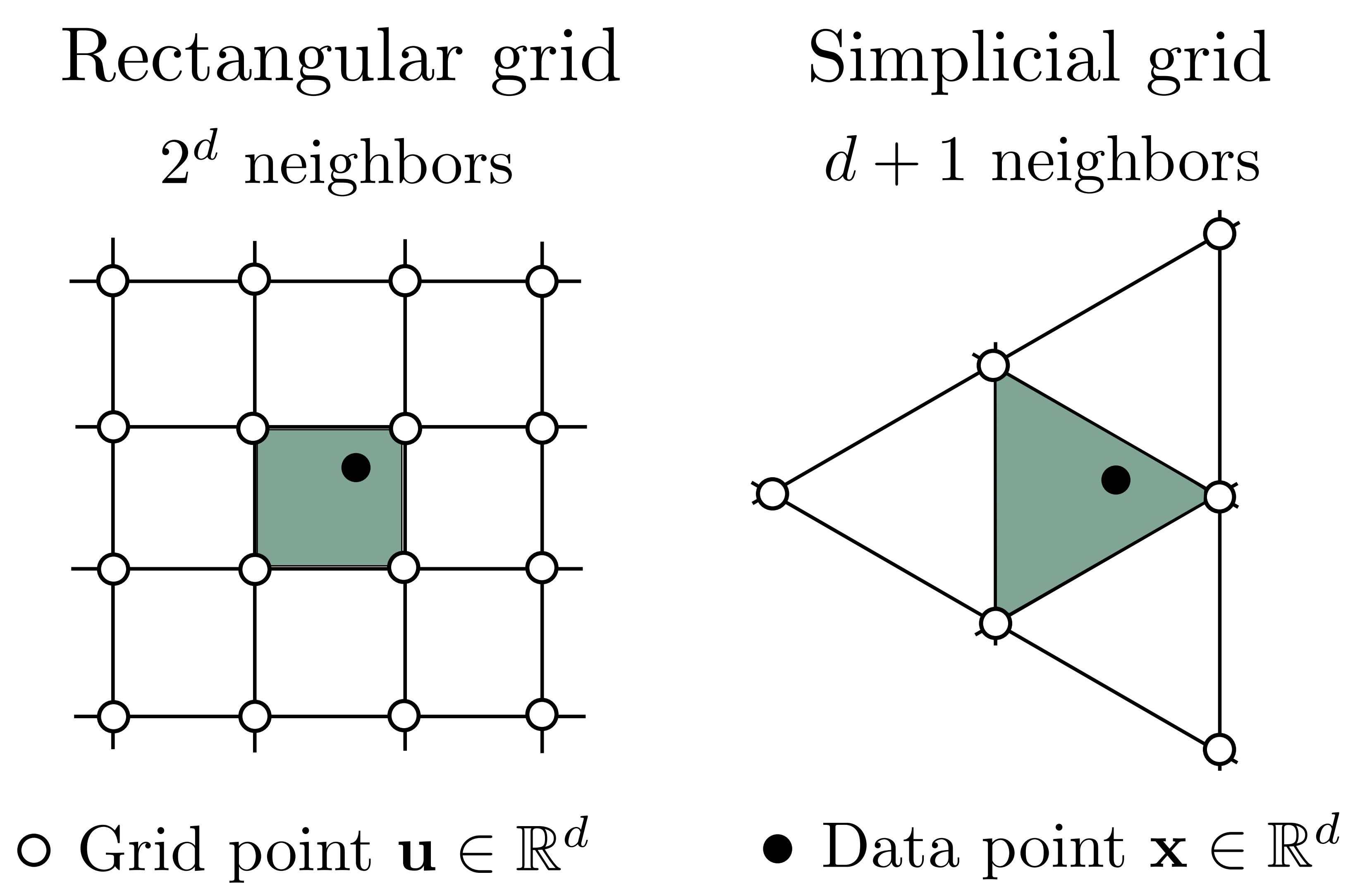
Gaussian Processes

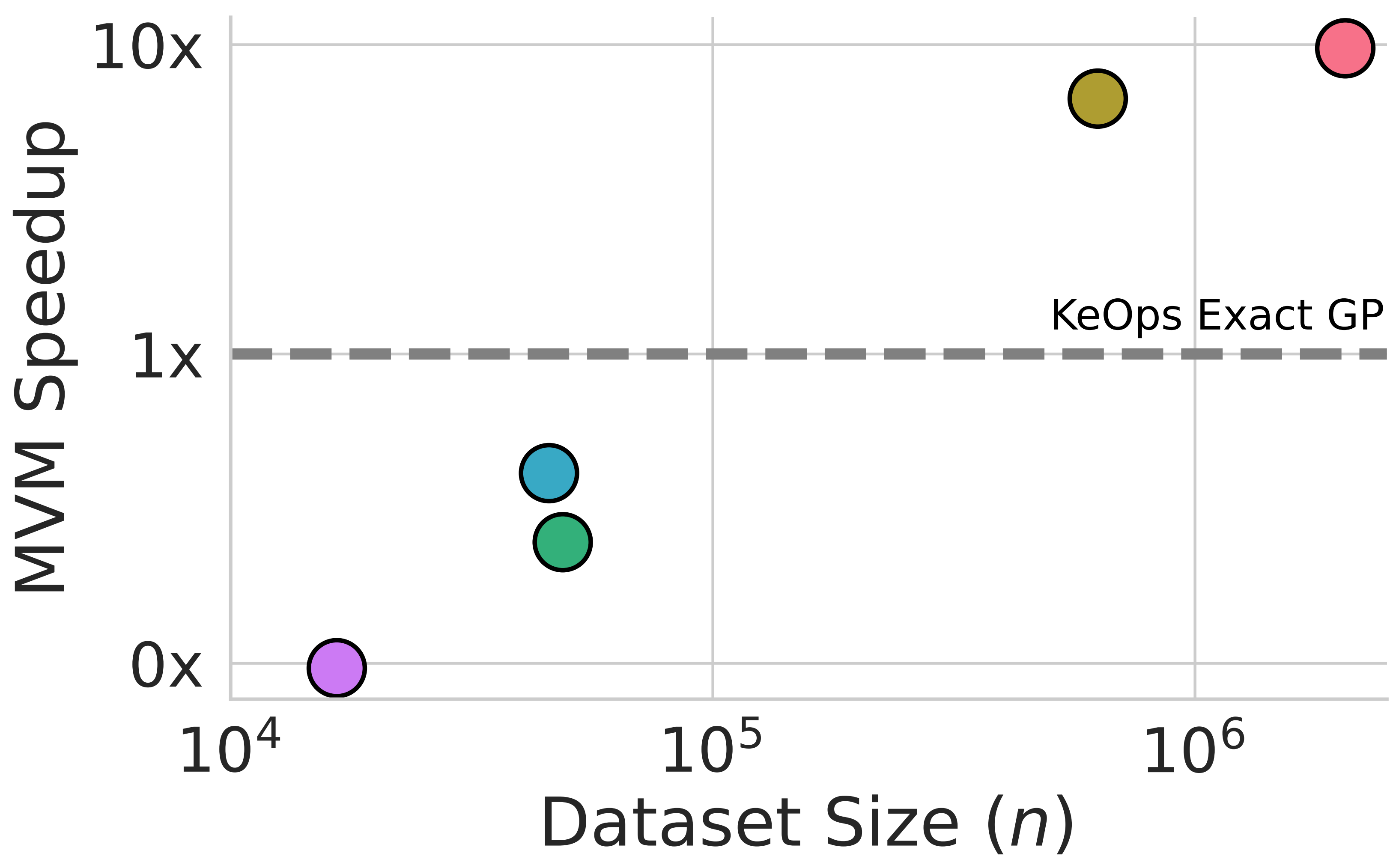
- Gaussian processes (GPs) are a popular non-parametric method that model priors over functions.
- GPs are very flexible, and provide well-calibrated uncertainties.



The Big Picture

- Modern scalable Gaussian processes rely on fast matrix-vector multiplies (MVMs).
- Structured Kernel Interpolation (SKI) uses sparse interpolation of inducing points on a rectangular grid, costing $\mathcal{O}(n4^d)$.
- We propose **Simplex-GPs**, that leverage equivalence between GP inference and bilateral filtering to instead use sparse interpolation on a simplicial grid, *exponentially* accelerating MVMs to $\mathcal{O}(nd^2)$.





Gaussian Processes for Regression

- For a typical regression problem with n inputs $\mathbf{X}$ and outputs $\mathbf{y}$, we model a Gaussian process prior, with a Gaussian observation likelihood as,

$$f(\cdot) \sim \mathrm{GP}(\mu(\cdot), k(\cdot, \cdot)) \ ,$$
$$\mathbf{y} \mid \mathbf{X} \sim \mathcal{N}(f(\mathbf{X}), \sigma^2\mathbf{I}) \ .$$

- The mean function μ is taken to be a constant function (often zero), and the kernel function k is defined by parameters θ.

- The objective of GP inference now is to find the posterior over functions - $p(f \mid \mathbf{X}, \mathbf{y}, \theta, \sigma^2)$.

Gaussian Process Inference

- Using Gaussian conditioning identities, for $n_\star$ novel inputs $\mathbf{X}_\star$, the posterior is fully specified by mean and covariance,

$$\mathbb{E}[f(\mathbf{X}_\star)] = \overbrace{\mu_{\mathbf{X}_\star}}^{n_\star \times 1} + K_{\mathbf{X}_\star,\mathbf{X}}[K_{\mathbf{X},\mathbf{X}} + \sigma^2\mathbf{I}]^{-1} \overbrace{\tilde{\mathbf{y}}}^{n \times 1},$$

$$\mathrm{cov}(\mathbf{X}_\star) = \underbrace{K_{\mathbf{X}_\star,\mathbf{X}_\star}}_{n_\star \times n_\star} - K_{\mathbf{X}_\star,\mathbf{X}}[\underbrace{K_{\mathbf{X},\mathbf{X}}}_{n \times n} + \sigma^2\mathbf{I}]^{-1}\underbrace{K^\top_{\mathbf{X}_\star,\mathbf{X}}}_{n \times n_\star}.$$

- All we need now is model selection, i.e. selecting values of kernel parameters θ and likelihood noise σ^2.

Gaussian Process Model Selection

- This is achieved by maximizing the marginal $\log$-likelihood (MLL) of data:

$$\log p(\mathbf{y} \mid \mathbf{X}) \propto -\frac{1}{2}\mathbf{y}^\top \left(K_{\mathbf{X},\mathbf{X}} + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{y} - \frac{1}{2} \log \left| K_{\mathbf{X},\mathbf{X}} + \sigma^2 \mathbf{I} \right| \, .$$

- Exact inverse and determinant computations are $\mathcal{O}(n^3)$, making the optimization prohibitively expensive even for moderately-sized datasets.

- Modern scalable GP methods instead rely on iterative methods like *conjugate gradients* (CG), often using $p \ll n$ MVMs involving only the data covariance matrix as $\mathbf{v} \mapsto K_{\mathbf{X},\mathbf{X}}\mathbf{v}$.

- Our work on accelerating MVMs, is therefore crucial for fast inference.

Inducing Point Methods

- Inducing point methods aim to reduce the computational burden by introducing a set of $m \ll n$ pseudo-points $\mathbf{U}$. We then have,

$$K_{\mathbf{X},\mathbf{X}} \approx \underbrace{K_{\mathbf{X},\mathbf{U}}}_{n \times m} \underbrace{K_{\mathbf{U},\mathbf{U}}^{-1}}_{m \times m} \underbrace{K_{\mathbf{X},\mathbf{U}}^{\top}}_{m \times n} .$$

- The computation now reduces to $\mathcal{O}(m^2 n + m^3)$.

- Structured Kernel Interpolation (SKI) argues that the cross-covariance matrix $K_{\mathbf{X},\mathbf{U}}$ incurs a significant cost for large-scale data.

Structured Kernel Interpolation

- Structured Kernel Interpolation (SKI) provides a general framework for approximating covariance matrices, even allowing $m \gg n$.

- A *sparse* interpolation $K_{\mathbf{X},\mathbf{U}} \approx W_{\mathbf{X}} K_{\mathbf{U},\mathbf{U}}$ is posited such that,

$$K_{\mathbf{X},\mathbf{X}} \approx W_{\mathbf{X}} K_{\mathbf{U},\mathbf{U}} W_{\mathbf{X}}^{\top} \,.$$

- By exploiting geometric structures on $\mathbf{U}$ like Kronecker factorization, the computational time is reduced to $\mathcal{O}(n4^d + g(m))$, but still suffers from the *curse of dimensionality*.

Wilson and Nickisch. Kernel Interpolation for Scalable Structured Gaussian Processes. In ICML 2015

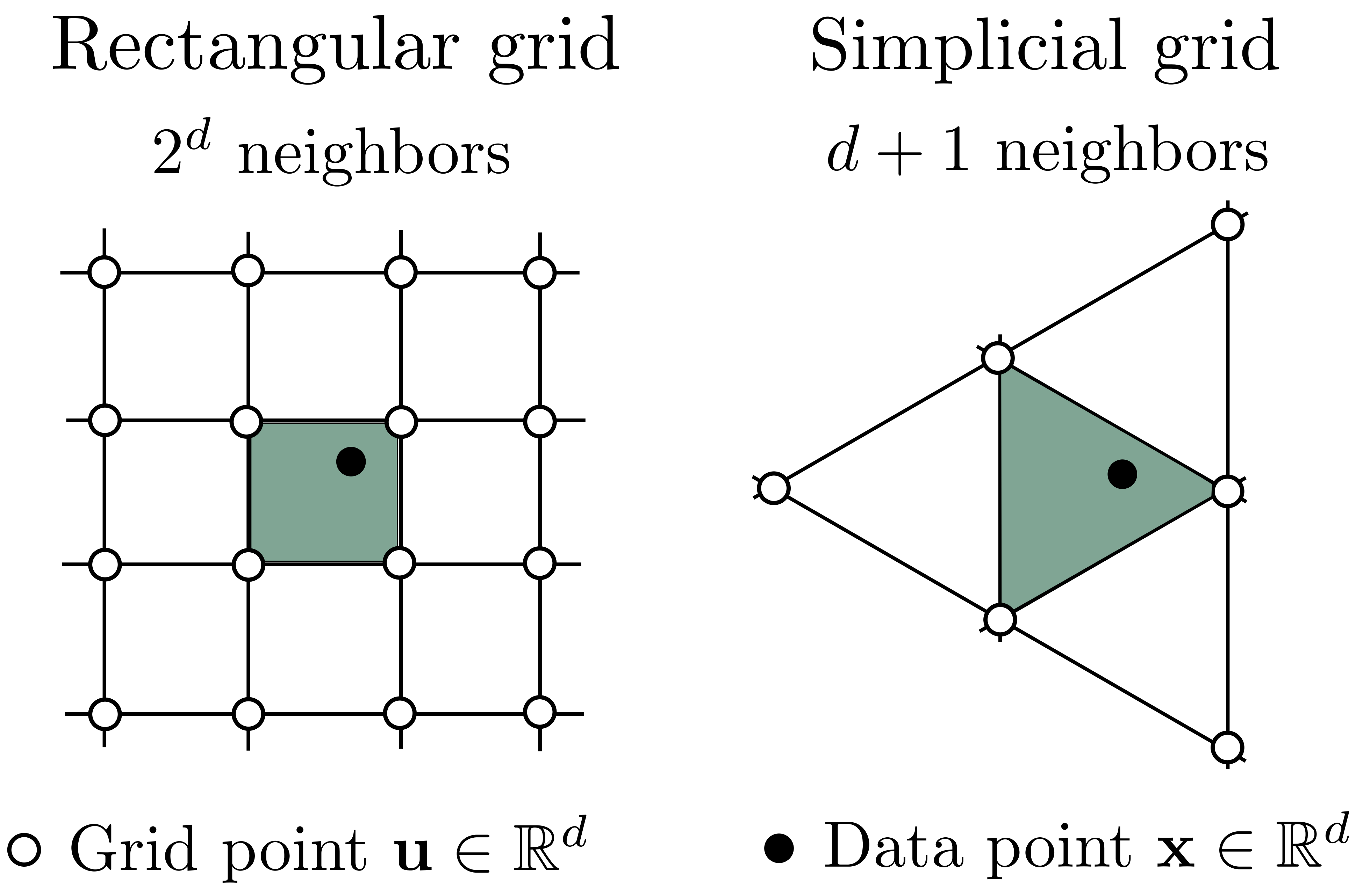
Bilateral Filtering

- High-dimensional Gaussian filtering can be described in general as a local interpolation,
$$\mathbf{y}_i' = \sum_{j=1}^{n} e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2} \mathbf{y}_j.$$
- In a *color bilateral filter*, location $\mathbf{x}$ represents the 2-D pixel locations and RGB color; values $\mathbf{y}$ represents the RGB color.
- Notably, the filtering operation is an MVM; a brute force computation would require $\mathcal{O}(n^2 d)$ time.

A bilateral filter is a non-linear version of the Gaussian filter that preserves sharp edges.

The Permutohedral Lattice

- Bilateral filtering can be accelerated.
- Instead of a rectangular grid, we tile the space with a *simplicial* grid.
- The number of neighbors are now *linear*, instead of exponential, in the dimension.
- Each input can be encoded as a barycentric interpolation of its enclosing simplex; filtering can be done in $\mathcal{O}(nd^2)$ time.



Adams, et. al. Fast High-dimensional Filtering using the Permutohedral Lattice. Computer Graphics Forum, 2010

Bilateral Filtering & MVM-Based GP Inference

- Under the SKI framework, MVM-based GP inference with the RBF kernel is *exactly* equivalent to bilateral filtering.

- Bilateral filtering is accelerated by embedding the locations onto a *sparse* permutohedral lattice.

- **Simplex-GPs** exploit this connection to help accelerate SKI inference, slashing the complexity to $\mathcal{O}(nd^2)$, and alleviating the curse of dimensionality.

Simplex Gaussian Processes

- For a vector $\mathbf{v}$, the key MVM approximation we care about, as developed by SKI, is

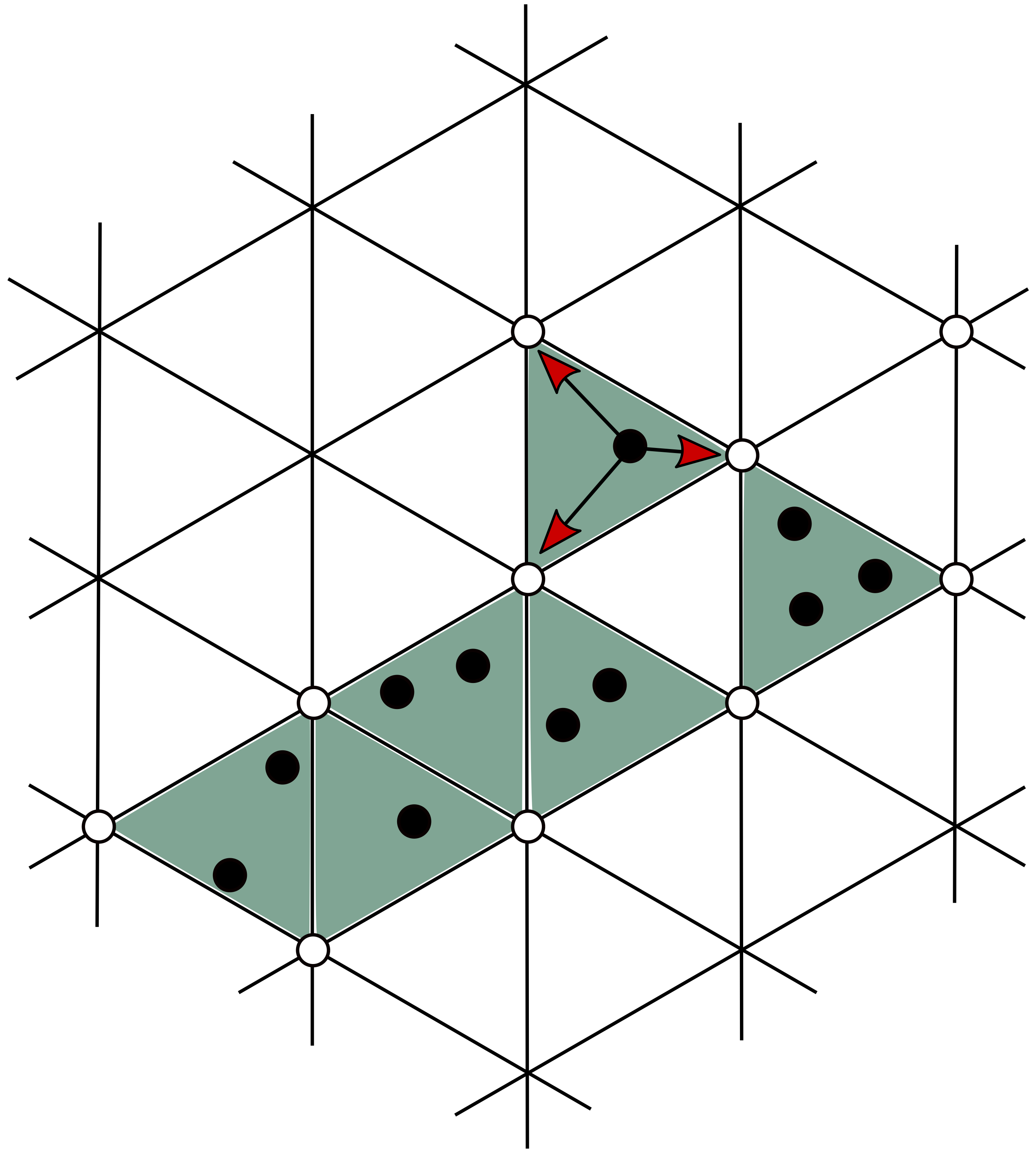
$$K_{\mathbf{X},\mathbf{X}}\mathbf{v} \approx \underbrace{W_{\mathbf{X}}}_{\text{Slice}} \; \underbrace{K_{\mathbf{U},\mathbf{U}}}_{\text{Blur}} \; \underbrace{W_{\mathbf{X}}^{\top}}_{\text{Splat}} \mathbf{v} \; .$$

- Executing this MVM using the permutohedral lattice is a three-stage operation — *splat*, *blur*, and *slice*.

- Consequently, a fast MVM directly impacts GP inference with conjugate gradients, which rely *exclusively* on MVMs.

Simplex GPs — Splat

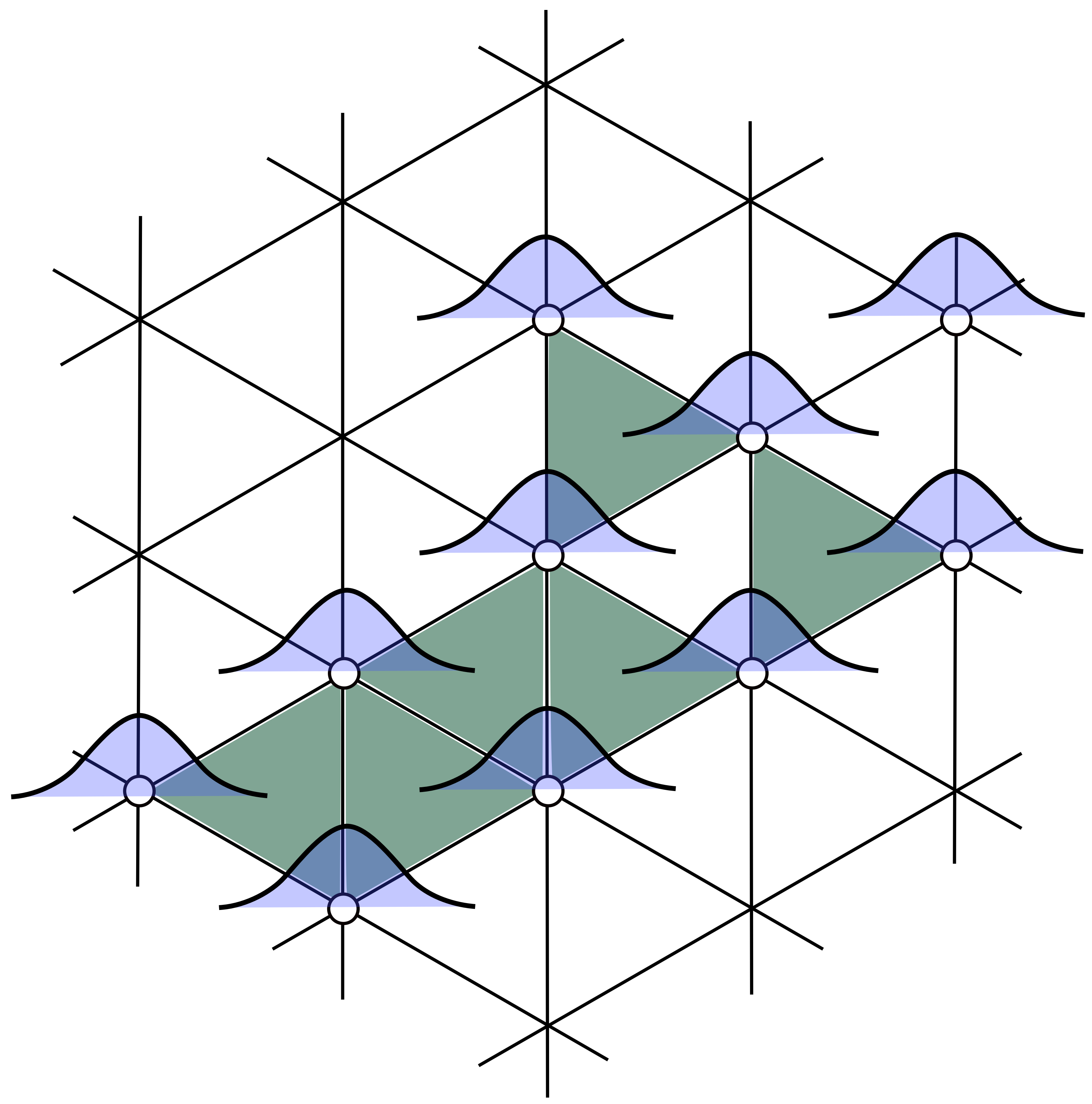
- Consider a set of d-dimensional locations $\mathbf{X}$, with corresponding values $\mathbf{v}$.
- Splat projects locations $\mathbf{X}$ on to the lattice by finding each enclosing simplex in $\mathcal{O}(d^2)$, storing sparsely.
- Each lattice vertex stores the barycentric weights for interpolation of both locations and corresponding values, using $W_{\mathbf{X}}$ implicitly.
- Each of the m generated vertices now acts as an inducing point $\mathbf{U}$.



Multiply by $W_{\mathbf{X}}^{\top}$ (Splat)

Simplex GPs — Blur

- We can now apply convolutions using discretized filters on each vertex, e.g. $[1,2,1]$ binomial stencil for Gaussian blur.
- In a simplicial grid, *all* neighbors can be looked up in $\mathcal{O}(d^2)$.
- The weights for filtering, i.e. the blur stencil, implicitly correspond to the entries of the matrix $K_{\mathbf{U},\mathbf{U}}$.
- We also provide a general scheme to discretize any stationary kernel.



Multiply by $K_{\mathbf{U},\mathbf{U}}$
(Blur)

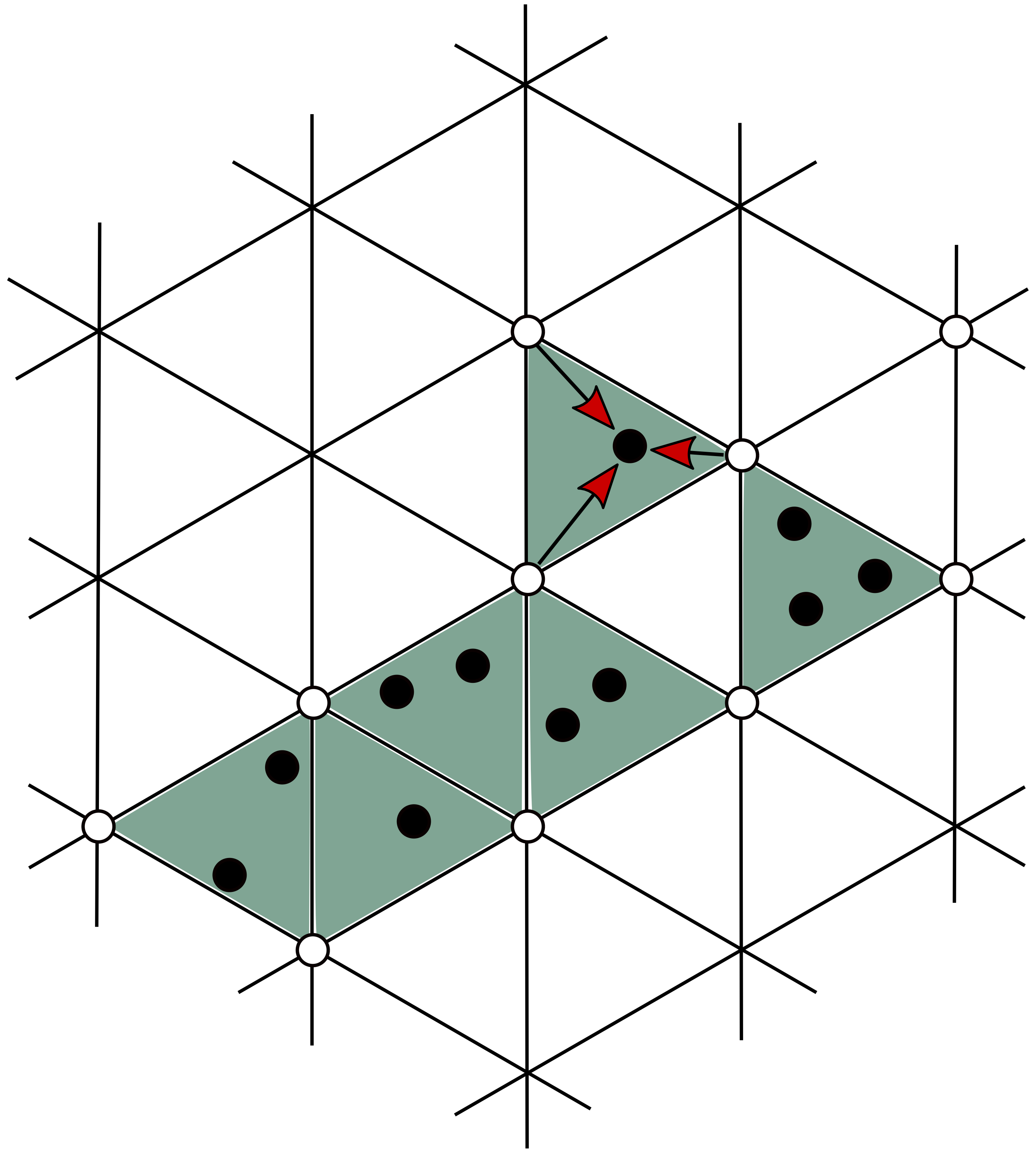
Simplex GPs — Slice

- *Slice* is the reversal of the *splat* operation.

- We project the locations back into original space, using the same barycentric weights for each lattice vertex computed during *splat*.

- The entire filtering, or the implied MVM, completes in $\mathcal{O}((n+m)d^2)$.

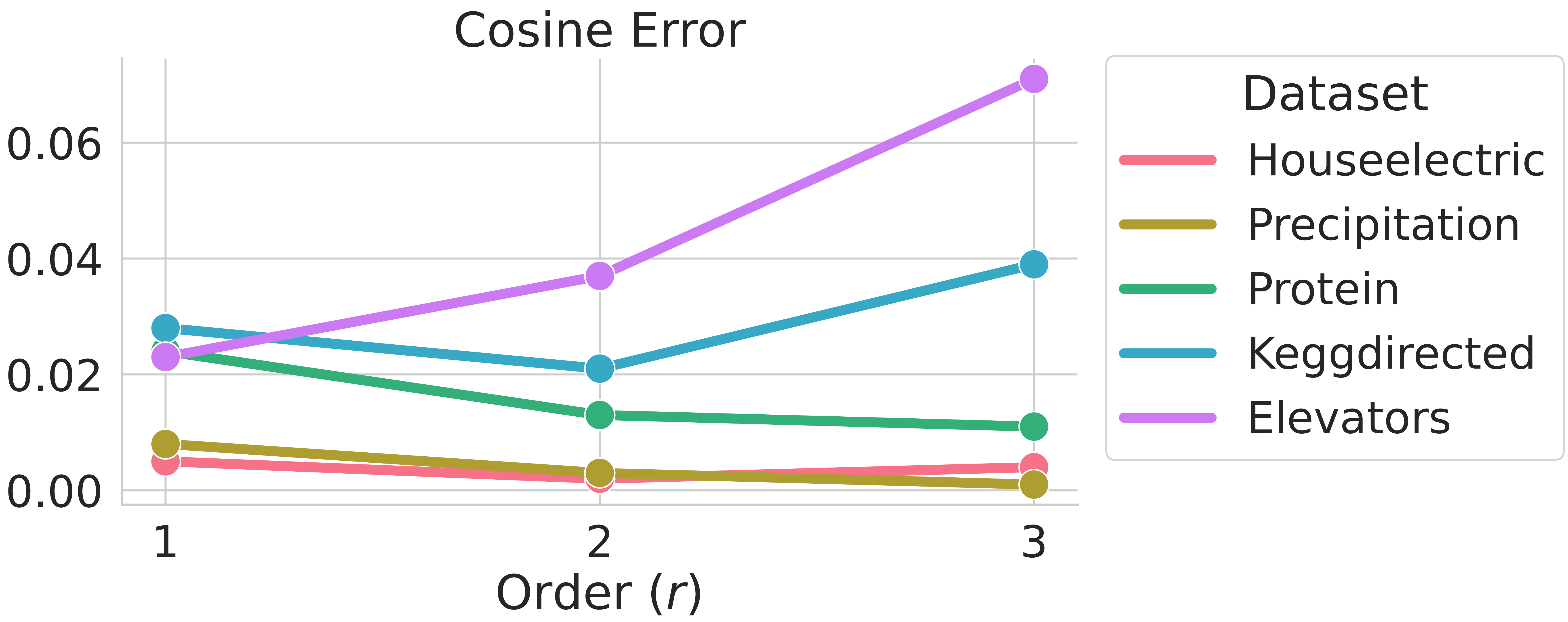
- We also show that derivatives can be approximated as a filtering operation too!



Multiply by $W_{\mathbf{x}}$
(Slice)

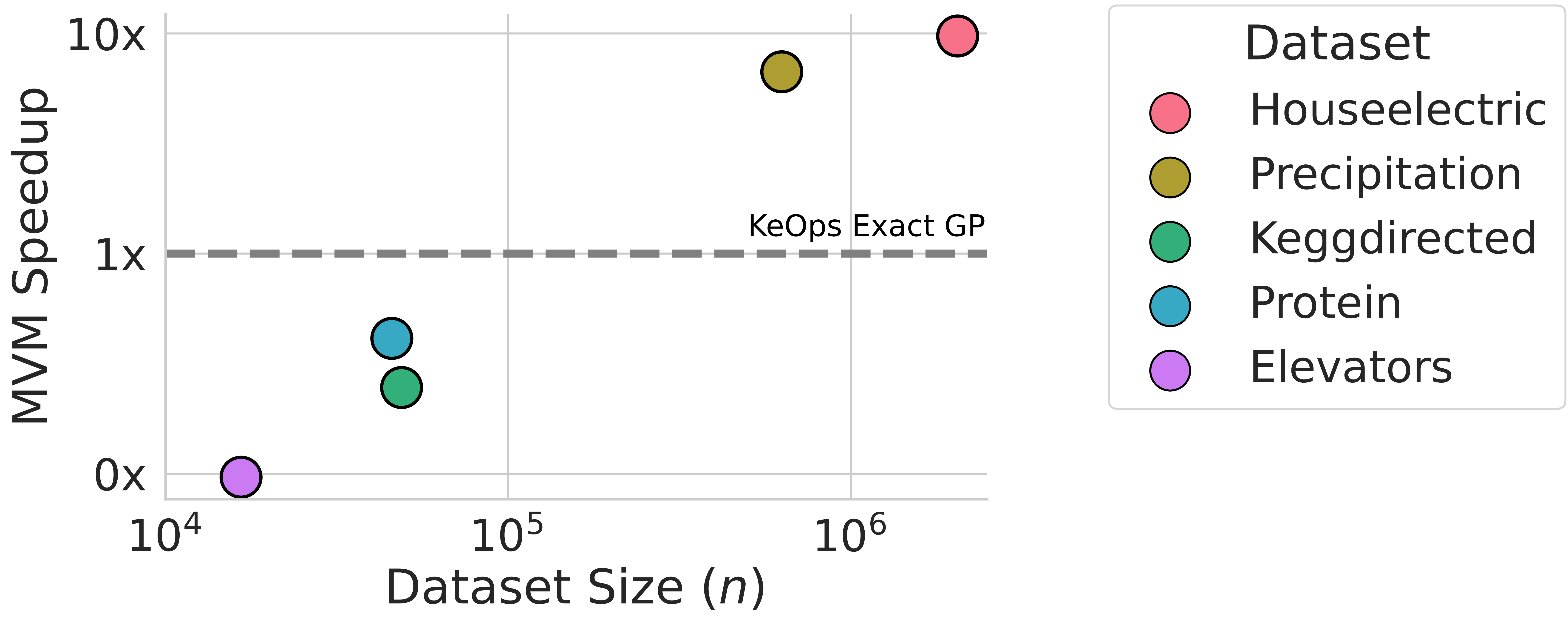
Small MVM Approximation Error

- We compute the cosine error incurred for an MVM w.r.t. exact computation with a KeOps RBF kernel for benchmark datasets.



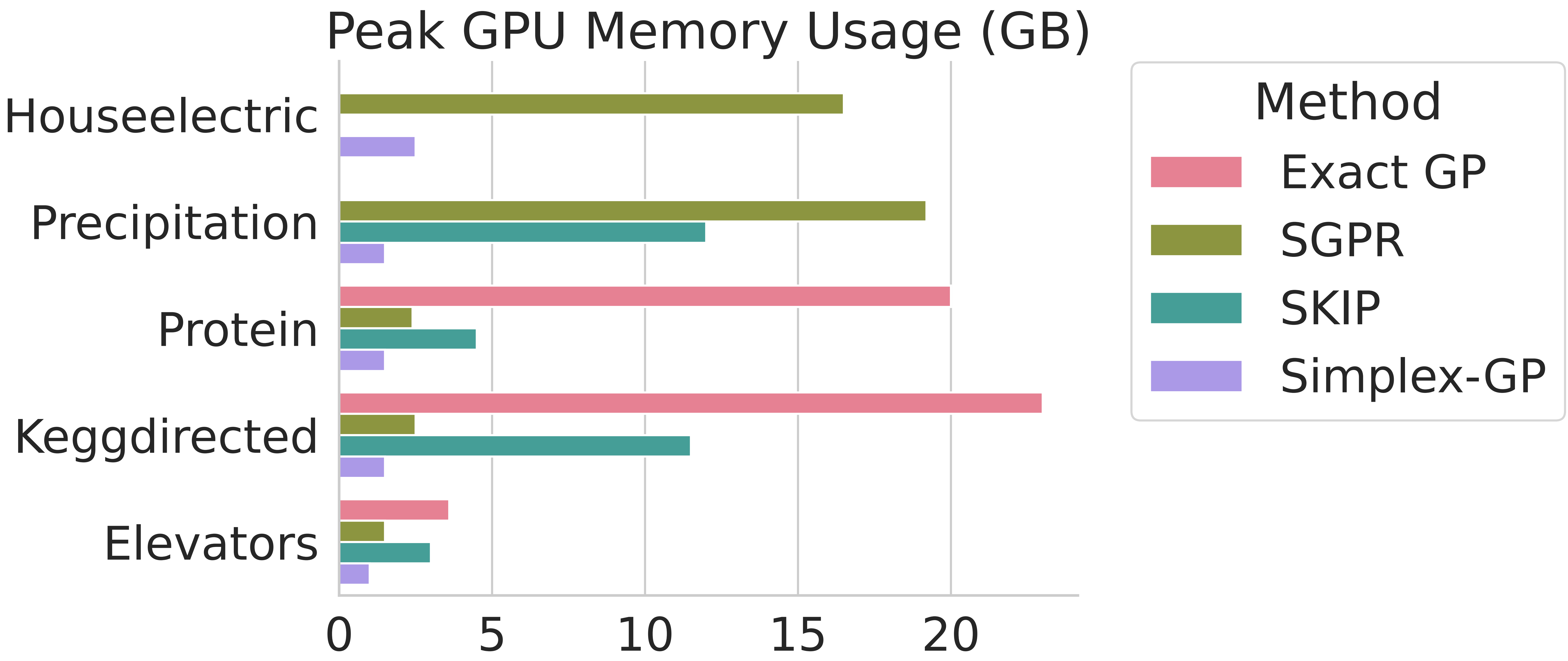
Fast Matrix-Vector Multiplies

- For large-scale datasets, Simplex-GPs can be up to 10 times faster as compared to a single MVM computation with KeOps.



Economical Memory Usage

- Simplex-GPs significantly reduce the peak memory footprint, by up to 10 times against competing approximate methods, and even more in comparison to exact GPs.



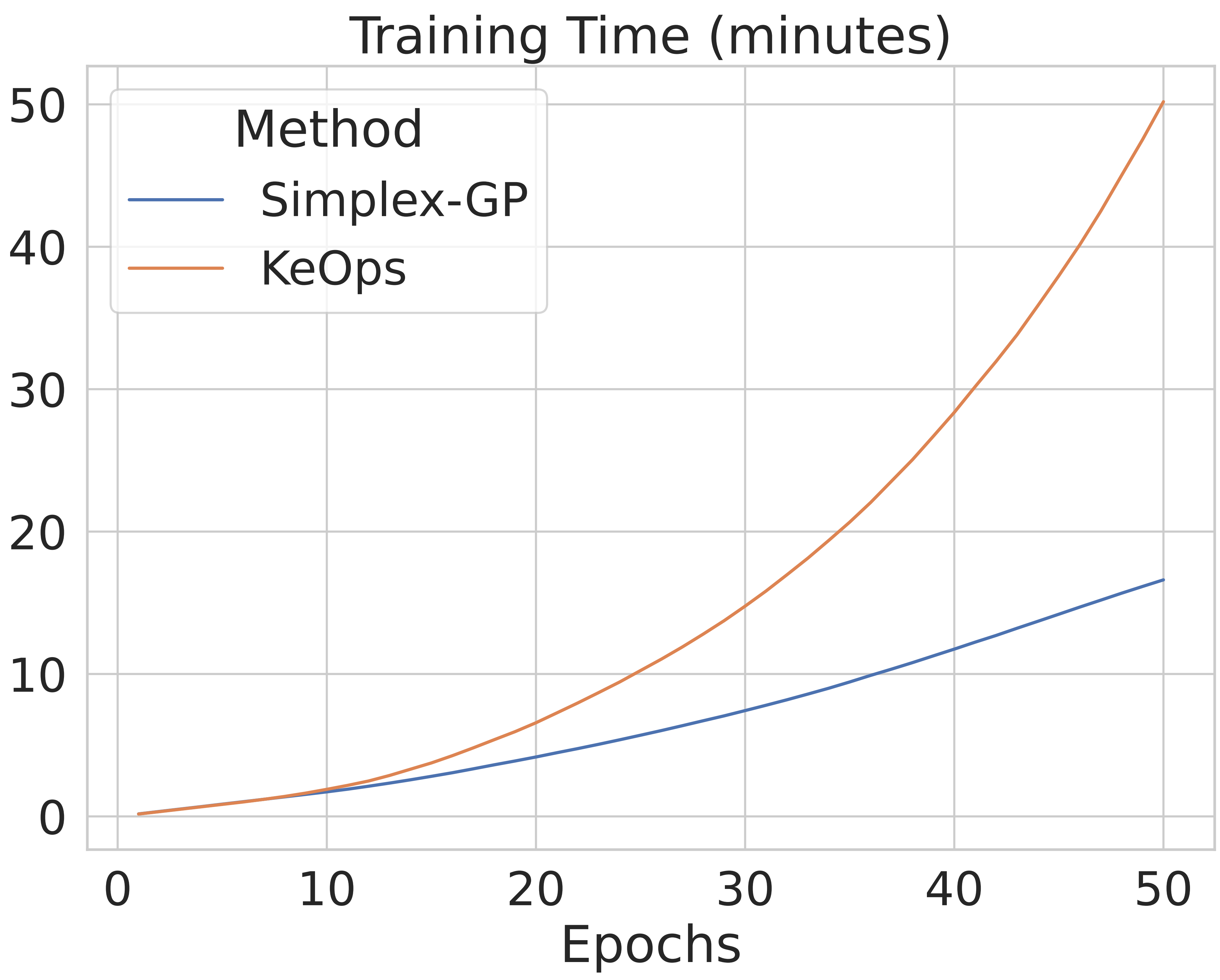
Minimal Performance Loss

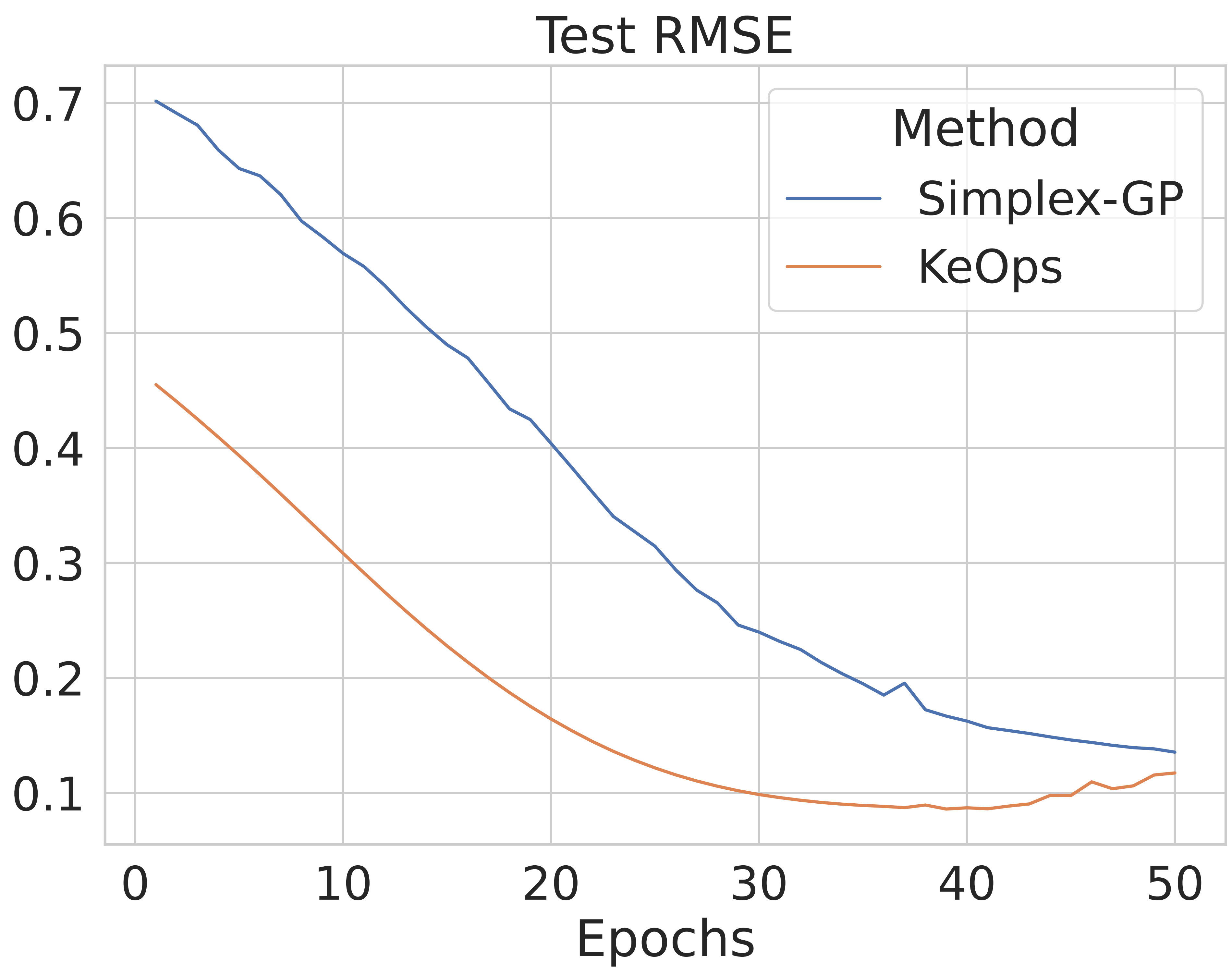
Test RMSE Performance

Dataset / Method	Exact GP (KeOps)	Simplex-GP	SGPR
Houseelectric	0.054	0.079	0.067
Precipitation	0.937	0.939	1.033
Keggdirected	0.083	0.095	0.380
Protein	0.511	0.571	0.579
Elevators	0.399	0.510	0.356

Faster Training

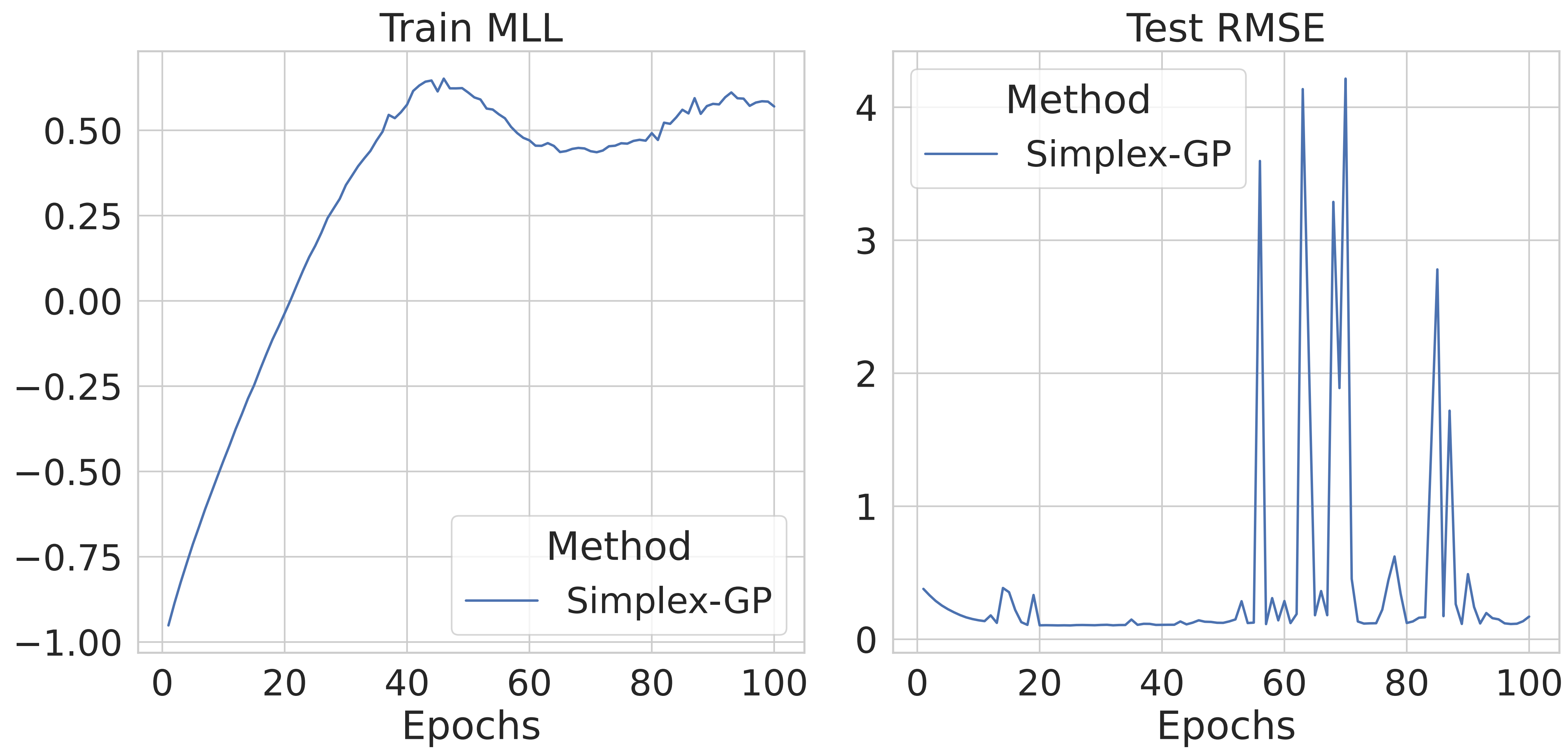
- Simplex-GPs can train faster at little loss in test performance, even when compared to highly scalable exact GP implementation using KeOps.





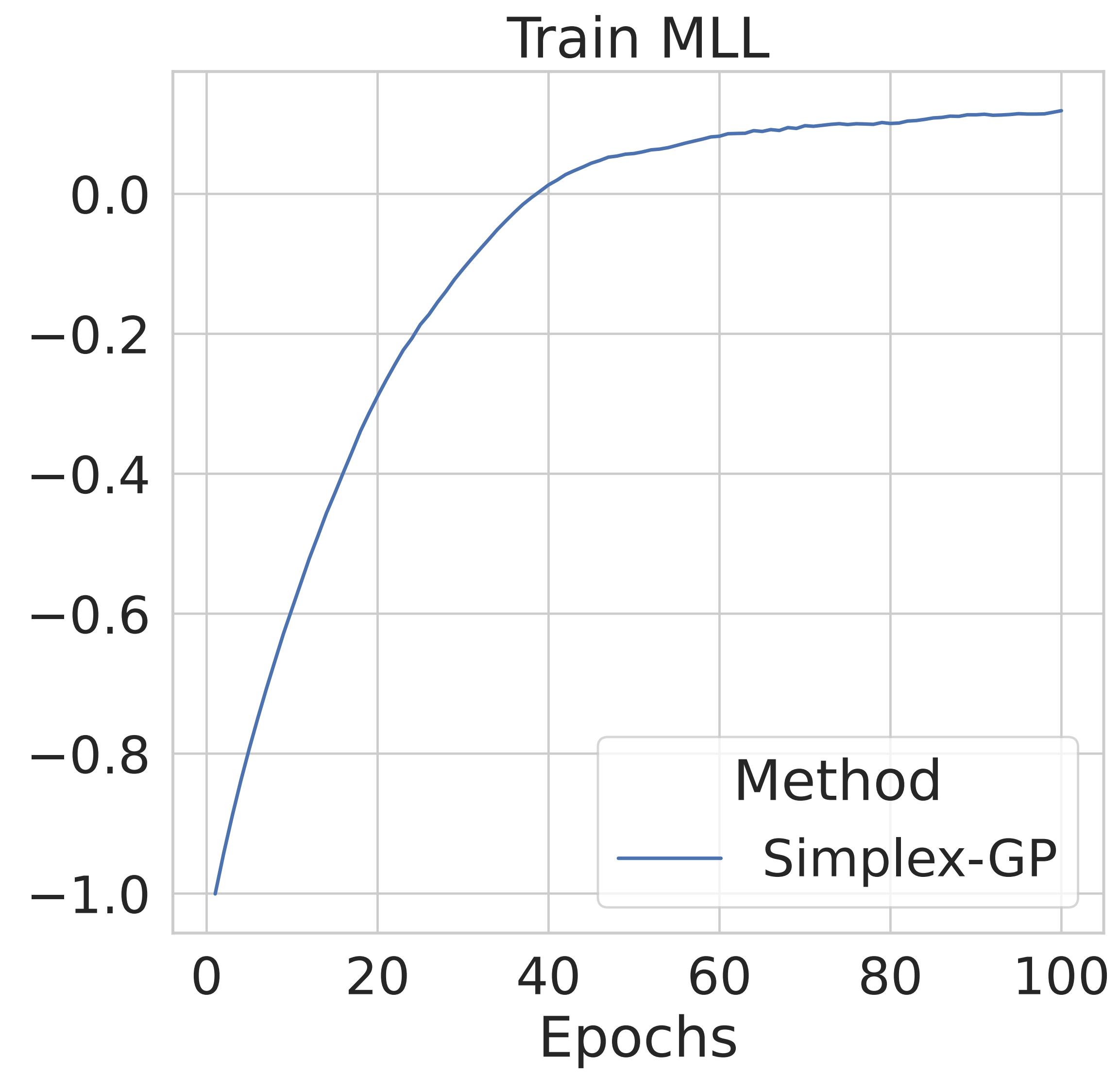
Practical Considerations

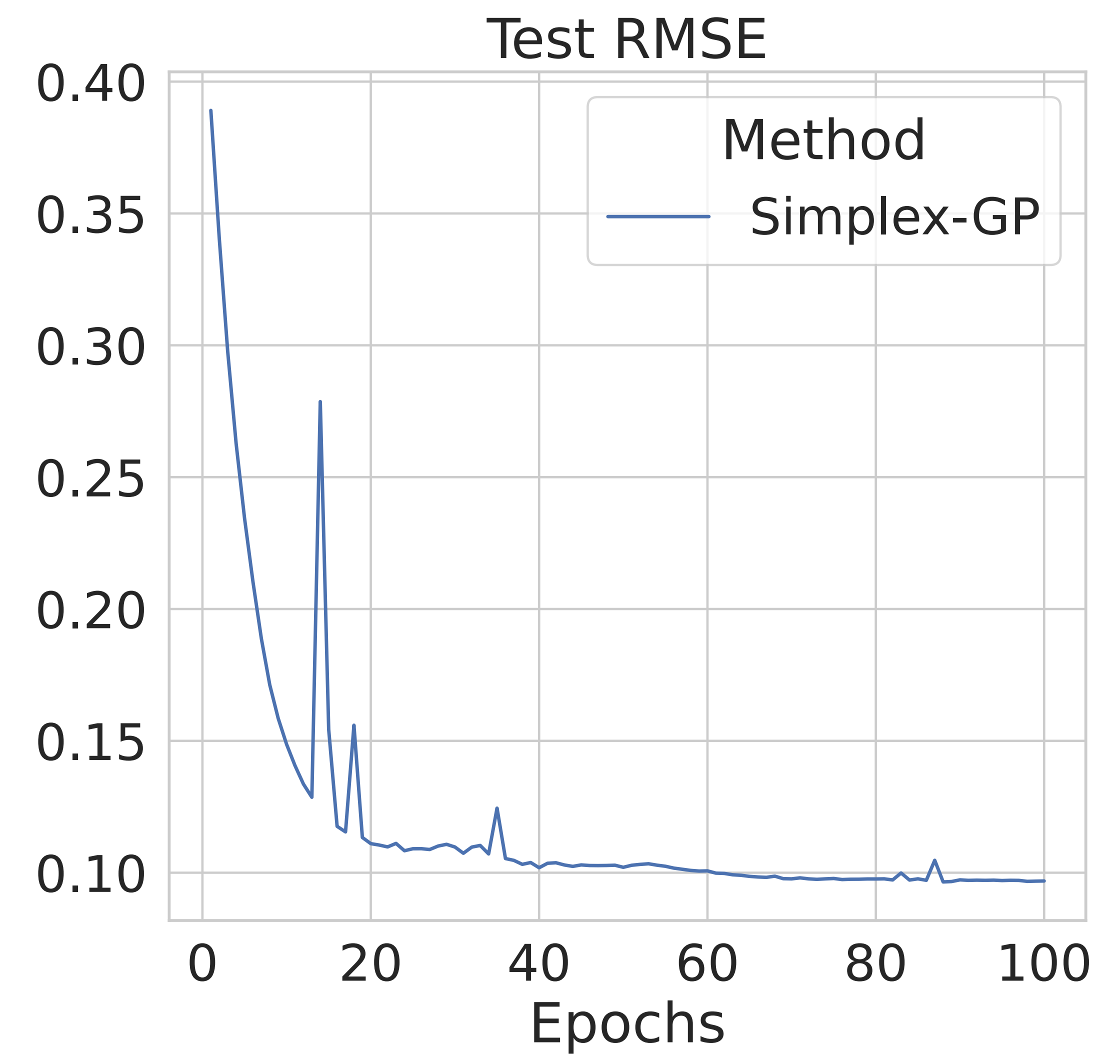
- Stability of iterative solvers for GP inference can be sensitive to the error tolerance in conjugate gradients (CG).
- This consideration is not limited to Simplex-GPs, but applies to CG based methods in general.



Improving Training Stability

- Training can stabilized by using:
  - early stopping while monitoring performance on a held-out validation dataset.
  - a lower CG error tolerance, typically of the order of 10^{-4}, but can significantly slowdown inference.
  - randomized truncations for bias-free CG, which can alleviate slowdown.





Potapczynski, Wu*, Biderman*, et. al. Bias-free Scalable Gaussian Processes via Randomized Truncations. ICML 2021*

When to use Simplex-GPs?

- Simplex-GPs can better exploit scenarios where we have,
  - large-scale datasets with more than $100k$ training points.
  - datasets that generate a sparse lattice, owing to
    - moderate data variance, or
    - moderately large kernel lengthscales.

Getting Started with Lattice Kernels

```
pip install gpytorch-lattice-kernel
```

A One Line Replacement in GPyTorch

- The lattice kernels can be simply dropped into existing GPyTorch models!

```
import gpytorch as gp
from gpytorch_lattice_kernel import RBFLattice

class SimplexGPModel(gp.models.ExactGP):
  def __init__(self, train_x, train_y):
    likelihood = gp.likelihoods.GaussianLikelihood()
    super().__init__(train_x, train_y, likelihood)

    self.mean_module = gp.means.ConstantMean()
    self.covar_module = gp.kernels.ScaleKernel(
-       gp.kernels.RBFKernel(ard_num_dims=train_x.size(-1))
+       RBFLattice(ard_num_dims=train_x.size(-1), order=1)
    )

  def forward(self, x):
    mean_x = self.mean_module(x)
    covar_x = self.covar_module(x)
    return gp.distributions.MultivariateNormal(mean_x, covar_x)
```

Challenges & Outlook

- Simplex-GPs provide a favorable trade-off between computation and performance for large-scale datasets.
- The runtime constants, however, are large such that the asymptotic gains are only realized for large datasets.
- We hope this cross-pollination of ideas stimulates both communities:
    - scalable GP inference for even higher-dimensional data,
    - and fast high-dimensional image filtering using scalable GP inference.

Resources

perhapsbay.es/simplex-gp

perhapsbay.es/simplex-gp-code