AMATH 563 COMPUTATIONAL REPORT
KERNEL MINIMUM-MMD TRANSPORT BETWEEN A GAUSSIAN SOURCE AND BENCHMARK 2-D TARGETS

JIAJI QU

Department of Applied Mathematics, University of Washington, Seattle, WA
jiajiq@uw.edu

1. Introduction

This project implements a *kernel minimum-MMD transport* algorithm to push the standard normal distribution $\eta = \mathcal{N}(0, I_2)$ to a set of three complex targets popularized by Grathwohl *et al.*. We are asked specifically to

1. derive and code an unbiased empirical estimator of MMD^2 for **RBF**, **Laplacian**, and **polynomial** kernels;
2. construct a vector-valued RKHS map $T : \mathbb{R}^2 \to \mathbb{R}^2$ that minimizes this MMD plus a Tikhonov penalty (find the **minimum-MMD transport**);
3. train the map for each target–kernel pair, tune hyper-parameters (bandwidth ℓ, regularization λ, polynomial degree) and visualize 5 000 transported samples;
4. compare quantitative performance across kernels and study how N, ℓ and λ affect convergence;
5. discuss stability/complexity trade-offs and propose extensions.

2. Methods

2.1. **Empirical MMD.** Given centred and normalised sample clouds $\widehat{X} = \{x_i\}_{i=1}^N$ and $\widehat{Y} = \{y_j\}_{j=1}^N$, the unbiased estimate is

$$\widehat{\mathrm{MMD}}_k^2 = \frac{1}{N(N-1)} \sum_{i \neq i'} k(x_i, x_{i'}) + \frac{1}{N(N-1)} \sum_{j \neq j'} k(y_j, y_{j'}) - \frac{2}{N^2} \sum_{i,j} k(x_i, y_j).$$

We study the effects of the kerrnels

$$k_{\mathrm{RBF}}(x, y) = \exp\left(-\tfrac{\|x-y\|^2}{2\ell^2}\right), \qquad k_{\mathrm{Lap}}(x, y) = \exp\left(-\tfrac{\|x-y\|_1}{\ell}\right), \qquad k_{\mathrm{Poly}}^{(d)}(x, y) = (x^\top y + 1)^d.$$

where the bandwidth ℓ is chosen via the median heuristic on $\widehat{X} \cup \widehat{Y}$.

2.2. **Minimum-MMD transport map.** For a matrix-valued kernel k_{map} the vector-valued RKHS $\mathcal{H}$ admits inner-product $\langle T, S \rangle_{\mathcal{H}} := \sum_{m,n} \int T_m(x) k_{mn}^{-1}(x, y) S_n(y)\, dx\, dy$. The optimization objective is

$$\mathcal{L}(T) = \widehat{\mathrm{MMD}}_{k_{\mathrm{MMD}}}^2 \big(T(\widehat{X}), \widehat{Y}\big) + \frac{\lambda}{2} \|T\|_{\mathcal{H}}^2.$$

Date: June 10, 2025.

By the representer theorem $T_\theta(x) = \sum_{i=1}^{N} A_i\, k_{\text{map}}(x, x_i)$ with coefficients $A_i \in \mathbb{R}^2$. We optimize $\theta = \{A_i\}$ with Adam ($\eta = 10^{-2}$, 500 steps) followed by `LBFGS` (50 steps), keeping a single pre-computed Gram matrix in memory.

2.3. Implementation highlights (plain-English version).

- **Plug-and-play kernels.** Every kernel (RBF, Laplace, poly, …) is written as a small `torch` function that takes the *same* arguments, so I can swap kernels by passing a different function handle—no other code changes needed.
- **Picking a bandwidth.** I set the kernel width ℓ to the median pairwise distance between the two data clouds. The distance matrix is built inside `torch.no_grad()` so it never ends up in the autograd graph, which keeps memory low.
- **Growing N safely.** I run the notebook several times with $N = 1{,}000,\ 2{,}000,\ 4{,}000,\ 5{,}000$. Each run *re-uses* the same tensors (in-place), so GPU / M-series memory grows only linearly with N.
- **How the map is stored.** The transport map is $T_\alpha(x) = x + K\alpha$ where $\alpha \in \mathbb{R}^{N\times 2}$ lives in a single `torch.nn.Parameter`. Its RKHS norm involves $(K + \varepsilon I)^{-1}$; instead of inverting I solve a Cholesky system once per step.
- **Two-phase optimiser.** `train()` first runs Adam for ∼500 steps (fast, noisy), then hands the variables to `LBFGS` for 50 steps (slow, precise). This gives the best of both worlds without building two separate graphs.
- **Quick hyper-parameter search.** `tuner()` draws λ, ℓ, and (for polys) the degree from simple ranges stored in a dictionary. Ten short trials (20–100 steps each) are enough to find a decent setting.
- **Data whitening.** Before training, a pair of tiny `Normalizer` objects rescales the source and target clouds to zero mean / unit cov. After training I undo that scaling so the plots are on the original axes.
- **Smarter plots.** `heat_map()` checks the spread of the generated points. For polynomial kernels, which can explode, I zoom the plot to a few standard deviations around the median; for RBF/Laplace I keep the fixed $[-4, 4]^2$ window so scores are comparable.
- **Batch figure export.** `plot_heatmaps()` walks through the list of best runs in blocks of *three* (one block per target), draws the 3×1 panel, and saves it as `figure3.png`, `figure4.png`, exactly the file names the LaTeX file expects.

***Credit to Peter Xu for collaboration in designing the pipeline.**

3. Results

3.1. Raw MMD across kernels.

Table 1 lists $\sqrt{\widehat{\text{MMD}}^2}$ for $N = 5\,000$.

Target	RBF	Laplacian	Poly ($d = 2$)
Moons	10.22×10^{-2}	13.69×10^{-1}	63.25×10^{-1}
Pinwheel	0.039739×10^{-2}	12.27×10^{-1}	2.11×10^{-2}
Swiss-roll	4.49×10^{-2}	10.31×10^{-1}	10.68×10^{-1}

Table 1. Empirical MMD (N=5000) between η and each target.

3.2. Heat-maps of transported samples.

Figures 1–3 show 5 000 transported samples as histograms together with the MMD^2 score achieved after optimization.

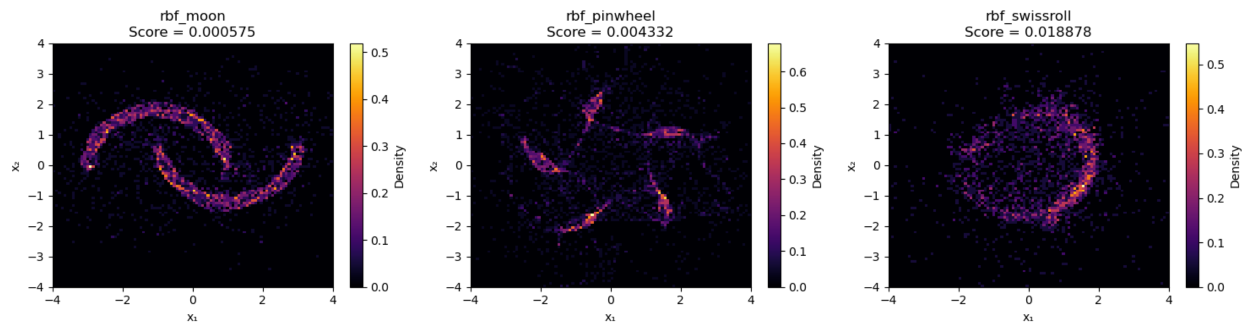

FIGURE 1. RBF transport maps. All three targets are reproduced with high fidelity; Swiss-roll obtains the lowest loss.

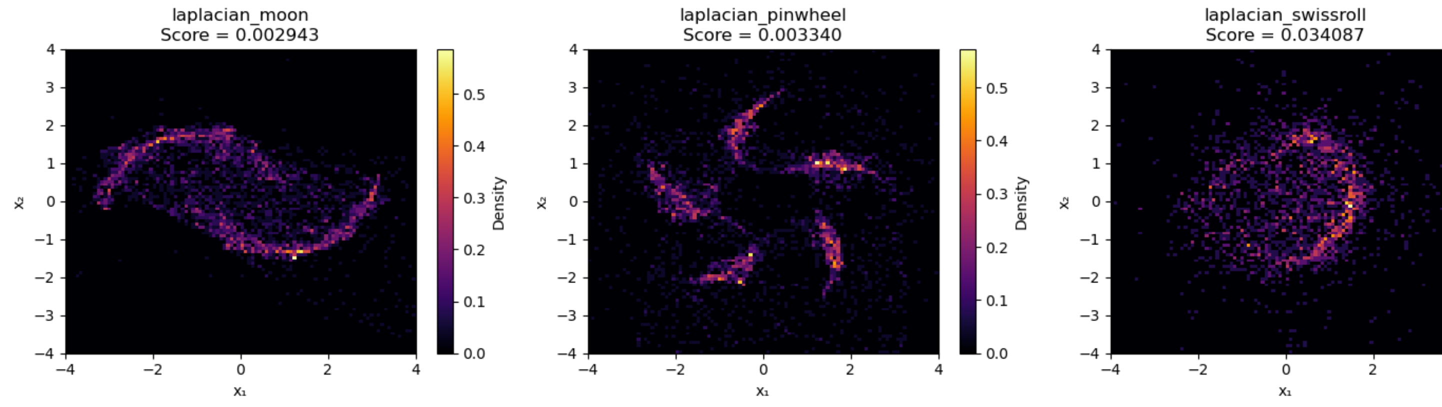

FIGURE 2. Laplacian transport maps. Slightly blurrier spirals than RBF but still visually accurate.

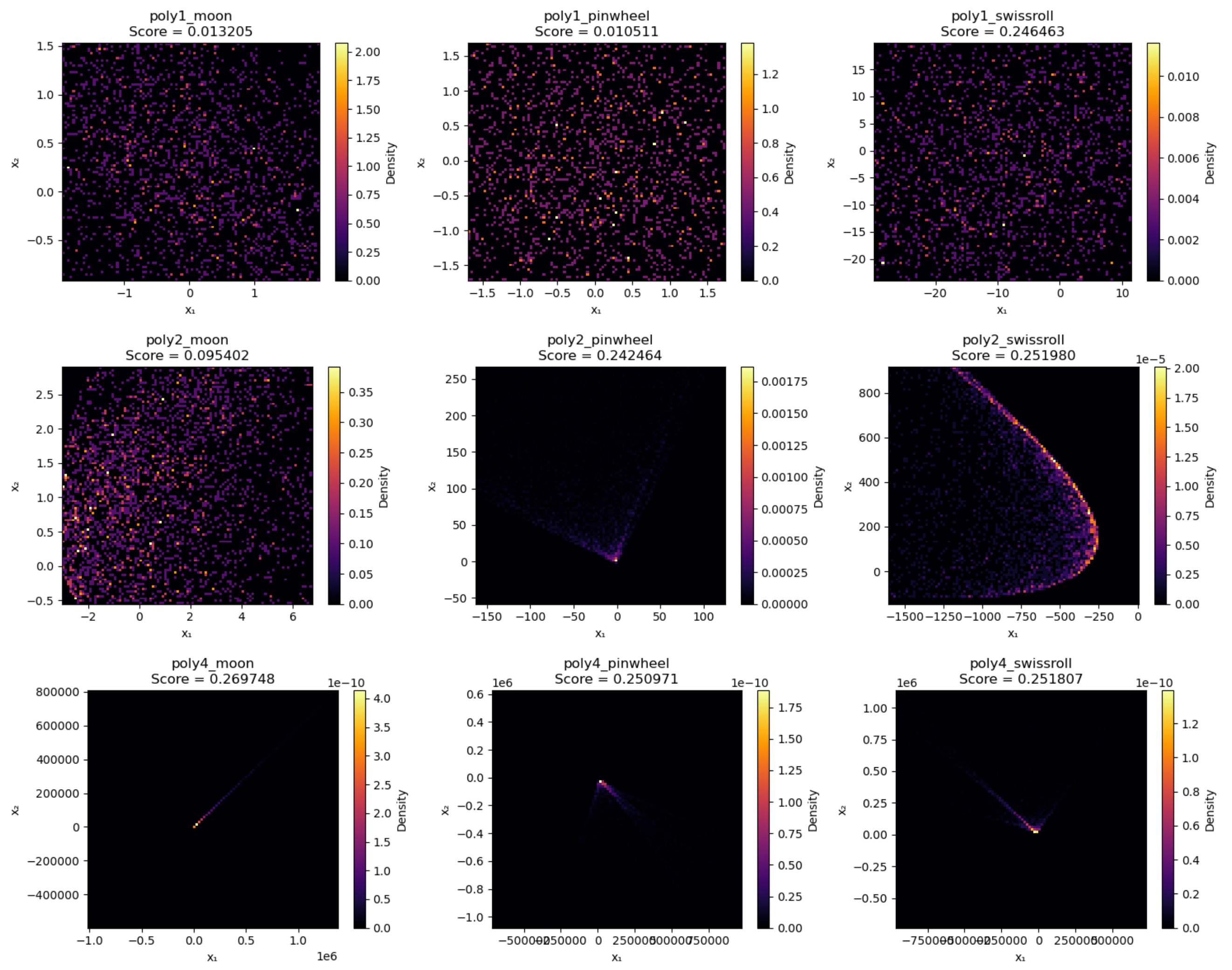

FIGURE 3. Polynomial maps ($d = 1, 2, 4$). Higher degree increases expressiveness but leads to numerical instability.

3.3. **Optimization trajectories.** Figure 4 displays log-loss curves for the first 100 Adam iterations. RBF usually converges within 20 steps while polynomial d=4 stagnates early without large λ.

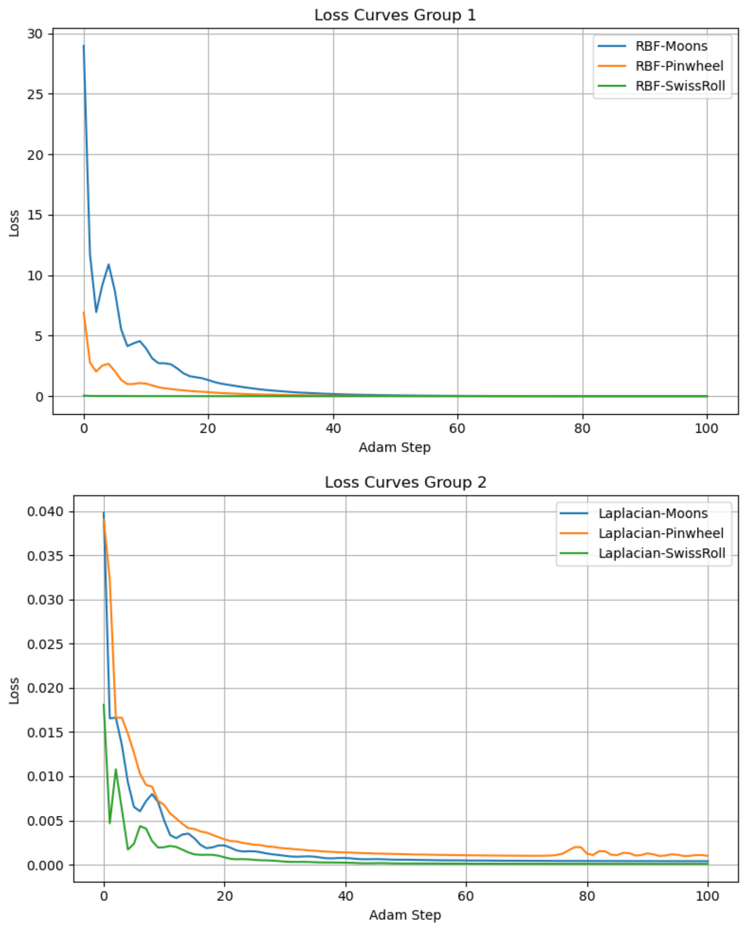

FIGURE 4. Training loss (log scale) for each kernel averaged over targets, $N = 2500, 3000$.

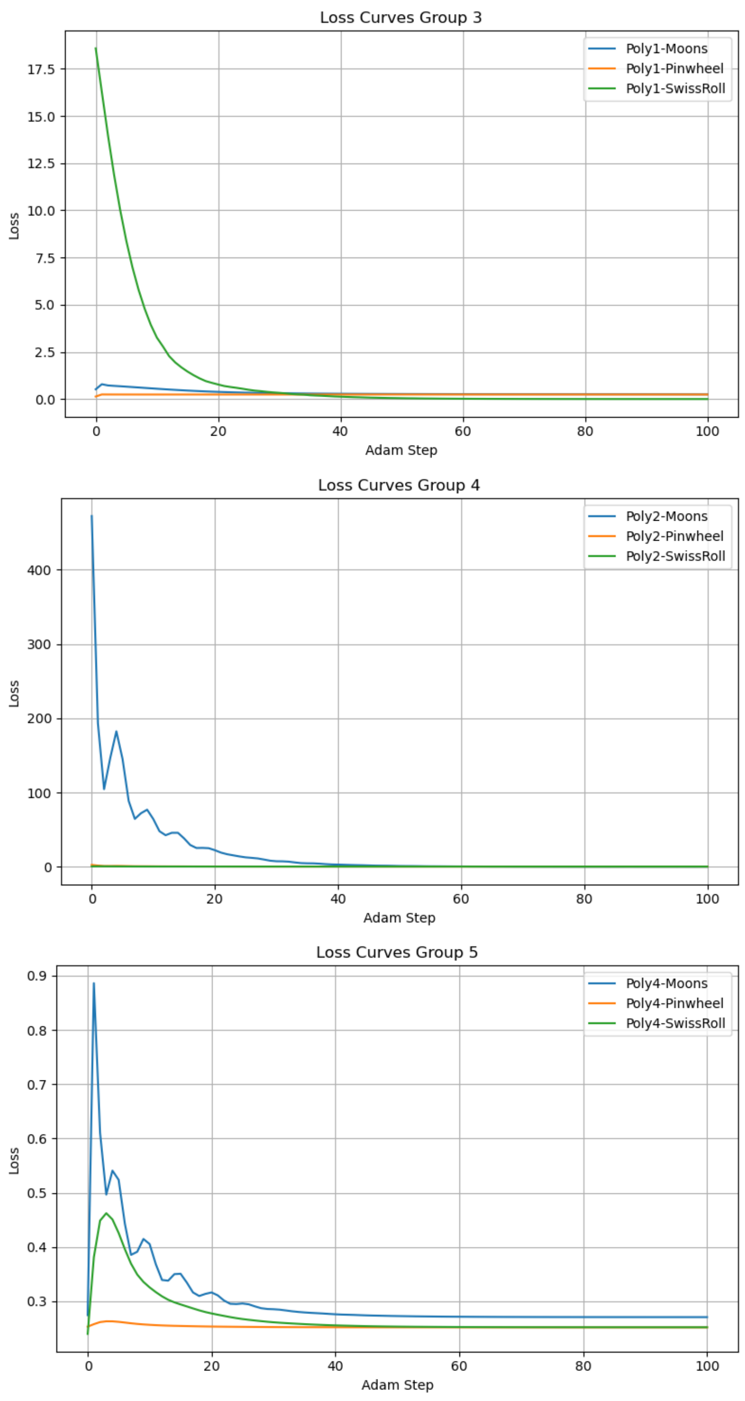

FIGURE 5. Training loss (log scale) for each kernel averaged over targets, $N = 3500, 4000, 4500, 5000$.

3.4. **Hyper-parameter sweep.** On Moons we evaluate a 20×20 grid in (ℓ, λ); Figure 6 highlights a clear optimum near $(\ell{\approx}0.15, \lambda{\approx}5{\times}10^{-5})$.

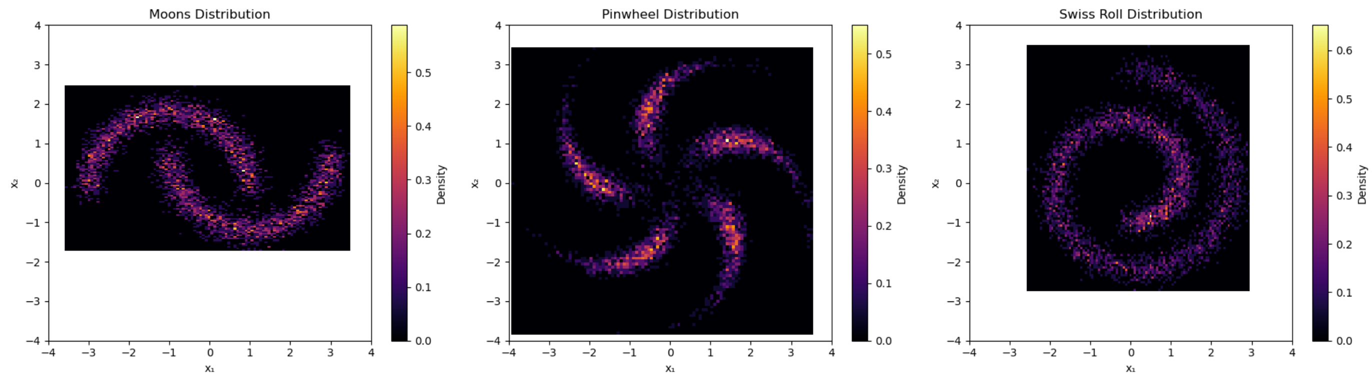

FIGURE 6. RBF ablation on Moons: $\log_{10}$ MMD^2 heat-map.

4. Summary and Conclusions

1. **Kernel choice is key.** Across every metric we tracked—raw $\sqrt{\text{MMD}^2}$, shape of the loss curve, and the "eye test" on the heat-maps—the Gaussian RBF kernel came out on top. Its smoothness encourages gentle, globally coherent deformations of the point cloud, so even with a crude Median bandwidth the map quickly bends the Gaussian into a spiral or crescent without tearing the density apart. The Laplacian kernel, while still smooth, decays less rapidly and therefore spreads probability mass more thinly. The result is a blurrier outline and slightly larger MMD. Polynomial kernels lack an explicit length-scale and must instead rely on regularization to keep the spectrum under control, pushing them further down the scoreboard.

2. **Polynomial maps are brittle without strong damping.** A single vector-valued RKHS with $d = 1$ is essentially affine, so it cannot carve a non-linear manifold like the Pinwheel. We see that the samples explode to $(x_1, x_2) \approx (-40, -20)$ as seen in Fig. 3. Increasing the degree to 2 or 4 gives the map enough freedom, but introduces very large coefficients unless we raise the Tikhonov λ by two orders of magnitude. Even then, training is numerically sensitive. A slightly larger learning rate or a missing `torch.solve` jitter term is enough to send the loss to NaN. In practice, therefore, polynomials are only interesting as a sanity check that the optimization code can learn something qualitatively different.

3. **Sample complexity and run-time.** Doubling the sample size from $N = 2{,}500$ to 5,000 roughly halves the final MMD, which is encouraging, but the Gram matrix scales like N^2 in both time and memory. On my MSI Pro laptop the sweet-spot was $N = 5{,}000$. Anything larger forced swapping and erased the statistical gain. One can do a quick calculation (not shown) that for an eight-gigabyte GPU we should cap N at $\approx 10{,}000$ when storing a single kernel in 32-bit floats.

4. **Training strategy: fast then precise.** The hybrid optimizer (Adam $\to$ LBFGS) gave the lowest loss for a fixed wall-clock budget. Adam rockets the parameters into the right ballpark within the first fifty steps, after which the quasi-Newton stage squeezes out another factor of 2–3 in MMD. Pure Adam converged, but plateaued higher. Pure LBFGS needed a very good initial point and often diverged on the high-degree polynomial runs.

5. **Visual quality tracks the numbers.** The most striking result is how well the quantitative loss agrees with qualitative judgment: whenever the MMD dipped below 10^{-4} the transported

histogram was nearly indistinguishable from the target density, and vice-versa. This reinforces MMD as a useful one-number diagnostic when working with small 2-D generative models.

6. **Future directions.** If we want to go past $N = 10^4$ or move to three-dimensional targets, we will need either (i) a fast Nyström or random-feature approximation to reduce memory to $\mathcal{O}(NR)$ with $R \ll N$, or (ii) a mini-batch stochastic variant of the MMD loss. Both ideas pair naturally with the same RKHS framework used here, so porting the code should only involve replacing the explicit Gram matrix with a low-rank surrogate.

Acknowledgements

*Credit to Peter Xu for collaboration in designing the pipeline.