Giacomo Meanti

Curriculum Vitae

Education

November 2022–Present **PostDoc**, *IIT*, Genova.
Working on physics-informed machine learning, as well as kernel-based algorithms for natural sciences (study of dynamical systems, force estimation) at the Italian Institute of Technology.

November 2019–2022 **PhD**, *Università di Genova*, Genova.
Supervised by Prof. Lorenzo Rosasco at the LCSL lab, I focused on the computational and statistical aspects of kernel methods, making them more scalable, efficient and user-friendly. My research interests include scalable and efficient shallow learning models, as well as applying ML to the natural sciences.

2022 **Visiting at University of California, Berkeley**.
I spent 4 months at UCB, supervised by Ben Recht, working on large scale kernel methods.

2016–2018 **MSc in Computer Science**, *ETHZ*, Zürich, 5.69/6.
I specialized in statistics and machine learning, focussing on their applications to biology through courses on computational systems biology and statistical models in computational biology.

2018 **Dissertation**, *Protein Contact Network Analysis with Graph Neural Networks*.
Supervisors: Prof. Joachim Buhmann, Dr. Stefan Bauer
This work explored a novel application of graph neural networks to protein structures, with the goal of learning general characteristics of different protein families. An attention-based algorithm allowed to identify the most important amino acids for classification.

2013–2016 **BSc in Computer Science**, *The University of Southampton*, Southampton (UK), *Grade: First Class Honours (average 90/100)*.

2016 **Dissertation**, *Modelling Capacitor Patterns To Detect Alignment*.
Supervisor: Dr. Klaus-Peter Zauner
I worked on the optimization of capacitor plate conformation, with the goal of facilitating alignment detection between two objects, constrained by low precision measurements.

Publications

2023 G. Meanti*, A. Chatalic, V. R. Kostic, P. Novelli, M. Pontil, L. Rosasco, "Estimating Koopman operators with sketching to provably learn large scale dynamical systems", arXiv:2306.04520.

2023 S. F. Keil*, G. Meanti*, F. Warburg, A. Kanazawa, B. Recht, "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance", CVPR.

2022 S. Vigogna, G. Meanti, E. De Vito, and L. Rosasco, "Multiclass learning with margin: exponential rates with no bias-variance trade-off", ICML.

2022 D. Lagomarsion-Oneto, G. Meanti, N. Pagliana, A. Verri, A. Mazzino, L. Rosasco, and A. Seminara, "Physics Informed Shallow Machine Learning for Wind Speed Prediction", Energy.

2022 G. Meanti, L. Carratino, L. Rosasco, E. De Vito, "Efficient Hyperparameter Tuning for Large Scale Kernel Ridge Regression", AISTATS.

2021 F. Ceola, E. Maiettini, G. Pasquale, G. Meanti, L. Rosasco, and L. Natale, "Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot", IEEE Transactions on Robotics.

2020 G. Meanti, L. Carratino, L. Rosasco, and A. Rudi, "Kernel Methods Through the Roof: Handling Billions of Points Efficiently", NeurIPS (**Oral**).

2018 G. Meanti, S. Bauer, X. Deupi, T. Flock, and J. Buhmann, "Protein Structure Analysis with Graph Neural Nets", Joint ICML and IJCAI 2018 Workshop on Computational Biology.

Teaching

2021–2022 **Teaching Assistant for *Machine Learning* course**, *UniGe*.
Prepared python labs for an introductory ML course, helping students complete their tasks.

2020–2021 **Introduction to Machine Learning – MAIA course**.
Prepared and taught several introductory machine learning lessons for corporate students through the MAIA program.

Talks

11/2022 *Conference*: Matematica per l'Intelligenza Artificiale e il Machine Learning - Giovani ricercatori @ Politecnico di Torino

2020–2022 Seminars in Computer Science @ UniGE

2021 Research oriented talk at G-Research

2020 Oral presentation at NeurIPS

Awards

2017 **ETH Zürich Scolarship**.
Full scolarship for the MSc in Computer Science at ETH Zürich.

July 2016 **Andy Cranny Memorial Prize**.
£100 award for best dissertation at the *University of Southampton*.

Work Experience

January – October 2019 **Modeling Expert**, *Learn to Forecast (l2f)*, Lausanne.
I worked in a team devising time-series models for financial data. Developing forecasting models for prices of different assets and deploying them at scale using cloud technologies. The startup environment led me to quickly learn different roles: putting ML into production, working with financial data, as well as the unique problems posed by time-series.

October 2017 – December 2018 **Research Assistant**, *ETHZ – Institute for Machine Learning*, Zürich.
I worked at the Institute for Machine Learning on a number of projects also in collaboration with external departments such as chemistry and neuroscience. I analyzed different datasets applying existing statistical techniques and exploring novel approaches to exploit data-dependent priors. We also worked on sharing our results with the wider research community.

Kernel methods through the roof: handling billions of points efficiently

Giacomo Meanti[1], Luigi Carratino[1], Lorenzo Rosasco[1,2,3], and Alessandro Rudi[4]

[1]*MaLGa, DIBRIS, Università degli Studi di Genova, Genova, Italy*
[2]*Center for Brains, Minds and Machines, MIT, Cambridge, MA, USA*
[3]*Istituto Italiano di Tecnologia, Genova, Italy*
[4]*INRIA - Département d'Informatique de l'École Normale Supérieure - PSL Research University, Paris, France*

giacomo.meanti@edu.unige.it luigi.carratino@dibris.unige.it lorenzo.rosasco@unige.it alessandro.rudi@inria.fr

Abstract

Kernel methods provide an elegant and principled approach to nonparametric learning, but so far could hardly be used in large scale problems, since naïve implementations scale poorly with data size. Recent advances have shown the benefits of a number of algorithmic ideas, for example combining optimization, numerical linear algebra and random projections. Here, we push these efforts further to develop and test a solver that takes full advantage of GPU hardware. Towards this end, we designed a preconditioned gradient solver for kernel methods exploiting both GPU acceleration and parallelization with multiple GPUs, implementing out-of-core variants of common linear algebra operations to guarantee optimal hardware utilization. Further, we optimize the numerical precision of different operations and maximize efficiency of matrix-vector multiplications. As a result we can experimentally show dramatic speedups on datasets with billions of points, while still guaranteeing state of the art performance. Additionally, we make our software available as an easy to use library[1].

1 Introduction

Kernel methods provide non-linear/non-parametric extensions of many classical linear models in machine learning and statistics [44, 48]. The data are embedded via a non-linear map into a high dimensional feature space, so that linear models in such a space effectively define non-linear models in the original space. This approach is appealing, since it naturally extends to models with infinitely many features, as long as the inner product in the feature space can

[1] https://github.com/FalkonML/falkon

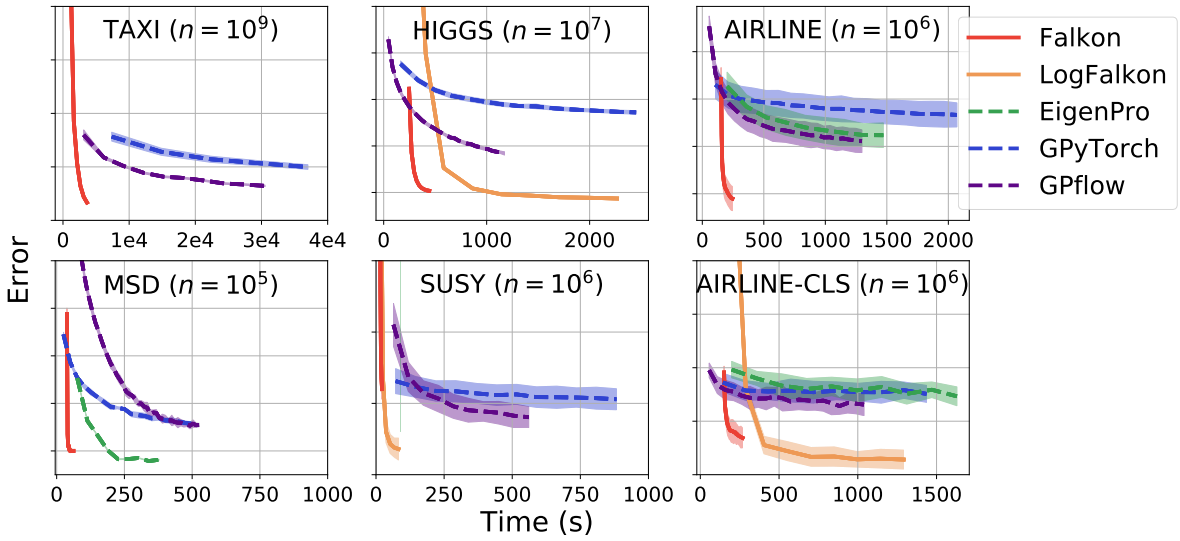


Figure 1: Benchmarks of kernel solvers on large scale datasets with millions and billions points (see Section 4). Our approach (red and yellow lines) consistently achieves state of the art accuracy in minutes.

be computed. In this case, the inner product is replaced by a positive definite kernel, and infinite dimensional models are reduced to finite dimensional problems. The mathematics of kernel methods has its foundation in the rich theory of reproducing kernel Hilbert spaces [46], and the connection to linear models provides a gateway to deriving sharp statistical results [52, 10, 53, 6, 4, 55]. Further, kernel methods are tightly connected to Gaussian processes [39], and have recently being used to understand the properties of deep learning models [22, 28]. It is not a surprise that kernel methods are among the most theoretically studied models. From a numerical point of view, they reduce to convex optimization problems that can be solved with strong guarantees. The corresponding algorithms provide excellent results on a variety of data-sets, but most implementations are limited to problems of small/medium size, see discussion in [51], Chapter 11. Most methods require handling a kernel matrix quadratic in the sample size. Hence, dealing with datasets of size 10^4 to 10^5 is challenging, while larger datasets are typically out of reach. A number of approaches have been considered to alleviate these computational bottlenecks. Among others, random features [37, 38, 65, 25, 12, 11] and the Nyström method are often used [60, 49], see also [14, 24, 17, 3, 66, 9]. While different, both these approaches consider random projections to reduce the problem size and hence computational costs. Renewed interest in approximate kernel methods was also spurred by recent theoretical results proving that computational gains can possibly be achieved with no loss of accuracy, see e.g. [26, 54, 40, 4, 41, 5].

In this paper, we investigate the practical consequences of this line of work, developing and testing large scale kernel methods that can run efficiently on billions of points. Following [42]

we use a Nyström approach to reduce the problem size and also to derive a preconditioned gradient solver for kernel methods. Indeed, we focus on smooth loss functions where such approaches are natural. Making these algorithmic ideas practical and capable of exploiting the GPU, requires developing a number of computational solutions, borrowing ideas not only from optimization and numerical analysis but also from scientific and high performance computing [27, 2, 7]. Indeed, we design preconditioned conjugate gradient solvers that take full advantage of both GPU acceleration and parallelization with multiple GPUs, implementing out-of-core variants of common linear algebra operations to guarantee optimal hardware utilization. We further optimize the numerical precision of different operations and investigate ways to perform matrix-vector multiplications most efficiently. The corresponding implementation is then tested extensively on a number of datasets ranging from millions to billions of points. For comparison, we focused on other available large scale kernel implementations that do not require data splitting, or multiple machines. In particular, we consider Eigenpro [29] which is an approach similar to the one we propose, and GPyTorch [15] and GPflow [57] which come from the Gaussian process literature. While these latter solutions allow also for uncertainty quantification, we limit the comparison to prediction. We perform a systematic empirical evaluation running an extensive series of tests. Empirical results show that indeed our approach can process huge datasets in minutes and obtain state of the art performances, comparing favorably to other solutions, both in terms of efficiency and accuracy. More broadly, these results confirm and extend the observations made in [28, 29], that kernel methods can now be seamlessly and effectively deployed on large scale problems. To make these new solutions readily available, the corresponding code is distributed as an easy to use library developed on top of PyTorch [35].

The rest of the paper is organized as follows. In Section 2, we provide some background on the considered approaches. In Section 3, we detail the main algorithmic solutions in our implementation, whereas the last section is devoted to assessing the practical advantages.

2 Background

Supervised learning is the problem of inferring an input-output function, given finitely many input-output pairs. In statistical learning theory the data $(x_i, y_i)_{i=1}^n$ are assumed to be sampled independently from a probability distribution ρ, and a loss function $\ell(y, f(x))$ is fixed measuring the cost of predicting $f(x)$ in place of y. The examples we consider are the squared $(y - f(x))^2$ and the logistic loss $\log(1 + e^{-yf(x)})$. Then, a good function f should minimize the expected loss

$$L(f) = \int \ell\big(f(x), y\big) d\rho(x, y). \tag{1}$$

A basic approach to solve the problem is empirical risk minimization, based on the idea of replacing the above expectation with an empirical average. Further, the search of a solution needs to be restricted to a suitable space of hypothesis, a simple example being linear functions $f(x) = w^\top x$. Kernel methods extend this idea by considering a non linear feature map

$x \mapsto \Phi(x) \in \mathcal{F}$ and functions of the form $f(x) = w^\top \Phi(x)$. Here $\Phi(x) \in \mathcal{F}$ can be seen as a feature representation in some space of features. The function space $\mathcal{H}$ thus defined is called reproducing kernel Hilbert space [45]. If we denote by $\|f\|_{\mathcal{H}}$ its norm then regularized empirical risk minimization is given by

$$\hat{f}_\lambda = \operatorname*{arg\,min}_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell\big(f(x_i), y_i\big) + \lambda \|f\|_{\mathcal{H}}^2, \tag{2}$$

where the penalty term $\|f\|_{\mathcal{H}}$ is meant to prevent possible instabilities and $\lambda \geq 0$ is a hyperparameter. From a statistical point of view the properties of the estimator $\hat{f}_\lambda$ are well studied, see e.g. [52, 6, 47]. Under basic assumptions, for $\lambda = \mathcal{O}(1/\sqrt{n})$, it holds with high probability that

$$L(\hat{f}_\lambda) - \inf_{f \in \mathcal{H}} L(f) = \mathcal{O}\Big(n^{-1/2}\Big). \tag{3}$$

This bound is sharp, but can be improved under further assumptions [6, 52]. Here, we use it for reference. From a computational point of view, the key fact is that it is possible to compute a solution also if $\Phi(x)$ is an infinite feature vector, as long as the kernel $k(x, x') = \Phi(x)^\top \Phi(x')$ can be computed [44]. The Gaussian kernel $\exp(-\|x - x'\|^2/2\sigma^2)$ is a basic example. Indeed, by the representer theorem [23, 45], $\hat{f}_\lambda(x) = \sum_{i=1}^n \alpha_i k(x, x_i)$, so Problem (2) can be replaced with a finite dimensional problem on the coefficients. Its solution depends on the considered loss, but typically involves handling the kernel matrix $K_{nn} \in \mathbb{R}^{n \times n}$ with entries $k(x_i, x_j)$, which becomes prohibitive as soon as $n \sim 10^5$ (although multi-GPU approaches [58] have been recently shown to scale to 10^6 points). In the following, we focus on Nyström approximation, considering functions of the form

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x, \tilde{x}_i), \tag{4}$$

where $\{\tilde{x}_1, \ldots, \tilde{x}_m\} \subset \{x_1, \ldots, x_n\}$ are inducing points sampled uniformly at random. As we discuss next, this approach immediately yields computational gains. Moreover, recent theoretical results show that the basic bound in (3) still holds taking as few as $m = \mathcal{O}(\sqrt{n})$ inducing points [41, 30]. With these observations in mind, we next illustrate how these algorithmic ideas can be developed considering first the square loss and than the logistic loss.

Squared loss. This choice corresponds to kernel ridge regression (KRR). Since both the loss and penalty are quadratic, solving KRR reduces to solving a linear system. In particular, letting $\boldsymbol{y} = (y_1, \ldots, y_n)$, we obtain $(K_{nn} + \lambda n I)\boldsymbol{\alpha} = \boldsymbol{y}$, for the coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$ in the solution of the problem in Eq. (2), while using the Nyström approximation (4) we get

$$(K_{nm}^\top K_{nm} + \lambda n K_{mm})\boldsymbol{\alpha} = K_{nm}^\top \boldsymbol{y}, \tag{5}$$

for $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$. The first linear system can be solved directly in $\mathcal{O}\big(n^3\big)$ time and $\mathcal{O}\big(n^2\big)$ space. In turn, Eq. (5) can be solved directly in $\mathcal{O}\big(nm^2 + m^3\big)$ time and $\mathcal{O}\big(m^2\big)$ space (if the K_{nm} matrix is computed in blocks). It is well known, that for large linear

Algorithm 1 Pseudocode for the Falkon algorithm.

function FALKON($X \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n, \lambda, m, t$)
  $X_m \leftarrow$ RANDOMSUBSAMPLE(X, m)
  $T, A \leftarrow$ PRECONDITIONER(X_m, λ)
  function LINOP($\boldsymbol{\beta}$)
    $\boldsymbol{v} \leftarrow A^{-1}\boldsymbol{\beta}$
    $\boldsymbol{c} \leftarrow k(X_m, X)k(X, X_m)T^{-1}\boldsymbol{v}$
    return $A^{-\top}T^{-\top}\boldsymbol{c} + \lambda n\boldsymbol{v}$
  end function
  $R \leftarrow A^{-\top}T^{-\top}k(X, X_m)\boldsymbol{y}$
  $\boldsymbol{\beta} \leftarrow$ CONJUGATEGRADIENT(LINOP, R, t)
  return $T^{-1}A^{-1}\boldsymbol{\beta}$
end function

function PRECONDITIONER($X_m \in \mathbb{R}^{m\times d}, \lambda$)
  $K_{mm} \leftarrow k(X_m, X_m)$
  $T \leftarrow \text{chol}(K_{mm})$
  $K_{mm} \leftarrow {}^1\!/_m TT^\top + \lambda \boldsymbol{I}$
  $A \leftarrow \text{chol}(K_{mm})$
  return T, A
end function

Note: LinOp performs the multiplication $\tilde{P}^\top H\tilde{P}\beta$ as in Eq. (8), via matrix-vector products.

systems iterative solvers are preferable [43]. Further, the convergence of the latter can be greatly improved by considering preconditioning. The naïve preconditioner P for problem (5) is such that $PP^\top = (K_{nm}^\top K_{nm} + \lambda n K_{mm})^{-1}$, and as costly to compute as the original problem. Following [42] it can be approximated using once again the Nyström method to obtain

$$\tilde{P}\tilde{P}^\top = (\tfrac{n}{m}K_{mm}^2 + \lambda n K_{mm})^{-1} \tag{6}$$

since $K_{mm}^2 \approx K_{nm}^\top K_{nm}$. Next, we follow again [42] and combine the above preconditioning with conjugate gradient (CG). The pseudocode of the full procedure is given in Algorithm 1. Indeed, as shown in [42] $\mathcal{O}(\log n)$ CG steps are sufficient to achieve the bound in (3). Then with this approach, the total computational cost to achieve optimal statistical bounds is $\mathcal{O}(n\sqrt{n}\log n)$ in time, and in $\mathcal{O}(n)$ in memory, making it ideal for large scale scenarios. The bulk of our paper is devoted to developing solutions to efficiently implement and deploy Algorithm 1.

Logistic loss. The above ideas extend to the logistic loss and more generally to self-concordant loss functions, including the softmax loss [31]. For reasons of space, we detail this case in Appendix B and sketch here the main ideas. In this case, iterative solvers are the default option since there is no closed form solution. Nyström method can be used a first time to reduce the size of the problem, and then a second time to derive an approximate Newton step [30]. More precisely, at every step preconditioned conjugate gradient descent is run for a limited number of iterations with a decreasing value of λ, down to the desired regularization level. In practice, this requires running Algorithm 1 multiple times with small number of iterations t and with decreasing λ. Making these ideas practical requires efficiently implementing and deploying Algoritm 1, making full use of the available computational architectures. This the core of our contribution that we detail in the next section.

3 Reformulating kernel solvers for multi-core/multi-GPU architectures

GPU machines have a peculiar architecture with rather different properties than the standard von Neumann computer, in particular they are characterized by highly parallel computational

power, relatively small local accelerator memory and slow memory transfer to/from the accelerator compared to their computational speed [63]. In their standard definition, kernel methods require large amounts of memory with a low density of operations per byte of memory used. This opens the question of how to adapt methods with low operation density to platforms designed to be extremely efficient with very high density of operations per byte. With this in mind, we started considering the state of the art kernel solver with minimal computational requirements for optimal guarantees (described at a high level in Algorithm 1), with the goal to reformulate its computational structure to dramatically increase the density of operations per byte, and reduce as much as possible the required memory use / transfers. To achieve this goal, we use a number of carefully designed computational solutions which systematically reduce the impact of the inherent bottlenecks of multi-core/multi-GPU architectures, while leveraging their intrinsic potential. In particular in the rest of this section we will focus on (a) minimizing the memory footprint of the solver, which has long been the main bottleneck for kernel methods, and is the main limitation encountered by current kernel solvers, (b) dealing with limited memory on the GPU, (c) reaching the highest possible accelerator utilization, parallelizing memory transfers and computation, (d) using the enhanced capabilities of GPUs with reduced-precision floating point data.

3.1 Overcoming RAM memory bottleneck

Kernel solvers that use the Nyström method need the matrices K_{mm} and K_{nm}. Since K_{nm} is used only in matrix-vector products, we can avoid constructing it explicitly (as we shall see in the following paragraphs) which leaves us to deal with the K_{mm} matrix. When m is large, it is crucial to carefully manage the memory needed for this task: in our implementation we only ever allocate one $m \times m$ matrix, and overwrite it in different steps to calculate the preconditioner. Indeed, choosing an appropriate form of the preconditioner, the matrix K_{mm} itself is not needed in the conjugate gradient iteration. Figure 2 shows the total memory usage, which consists of the preconditioner occupying approximately 90% of the memory (see last paragraph of Sect. 3.1), the weight vector $\boldsymbol{\beta}$ and two buffers holding (part of) the m inducing points and a data batch needed to compute K_{nm}.

In-place computation and storage of the preconditioner. The preconditioner $\tilde{P}$ of Eq. (6) is used to solve a linear system of the form $\tilde{P}^\top H \tilde{P} \boldsymbol{\beta} = \tilde{P}^\top K_{mn} \boldsymbol{y}$ with $H = K_{mn} K_{nm} +$

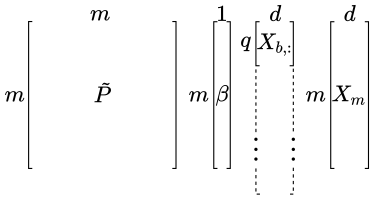


Figure 2: Structure of RAM allocation.

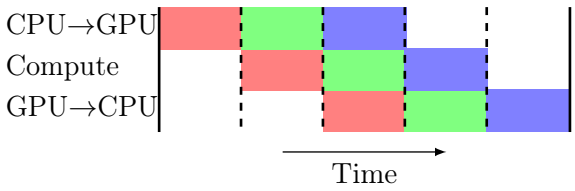


Figure 3: Overlapping memory transfers and computation.

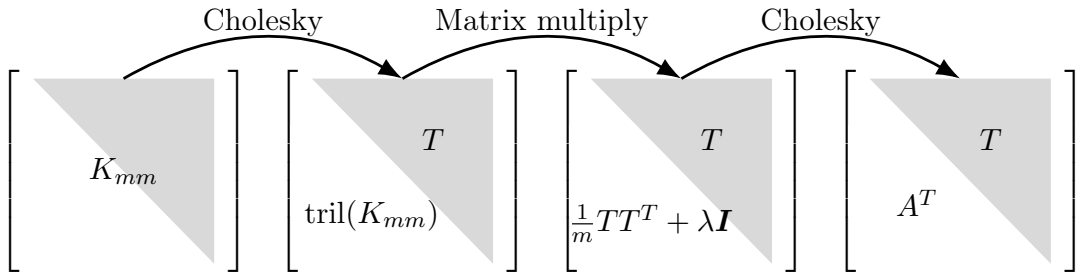


Figure 4: Evolution of the preconditioner matrix in memory.

$\lambda n K_{mm}$ and $\boldsymbol{\beta} = \tilde{P}^{-1}\boldsymbol{\alpha}$. $\tilde{P}$ can be decomposed into two triangular matrices obtained via Cholesky decomposition of K_{mm},

$$\tilde{P} = \tfrac{1}{\sqrt{n}} T^{-1} A^{-1}, \qquad T = \text{chol}(K_{mm}), \qquad A = \text{chol}(\tfrac{1}{m} T T^\top + \lambda \boldsymbol{I}_m). \tag{7}$$

All operations are performed in-place allocating a single $m \times m$ matrix as shown in Figure 4 and as described next: (a) a matrix of dimension $m \times m$ is allocated in memory; (b) the K_{mm} kernel is computed in blocks on the GPU and copied to the matrix; (c) *in-place* Cholesky decomposition of the upper triangle of K_{mm} is performed on the GPU (if the kernel does not fit GPU memory an out-of-core algorithm is used, see later sections); (d) the product $TT^\top$ is computed in blocks via GPU and stored in the lower part; (e) out-of-core in-place Cholesky decomposition is performed on the lower triangle to get $A^\top$. Additional care is needed to take into account the matrix diagonal, not described here for brevity.

Elimination of the storage of K_{mm}. Considering more carefully the matrix $\tilde{P}(K_{nm}^\top K_{nm} + \lambda n K_{mm})\tilde{P}$ with $\tilde{P}$ as in Eq. (7), we observe that the occurrences of K_{mm} cancel out. Indeed $(T^{-1})^\top K_{mm} T^{-1} = \boldsymbol{I}$ since $K_{mm} = T^\top T$ by Eq. 7. Then, the following characterization allows to overwrite K_{mm} when calculating the preconditioner.

$$\begin{aligned} \tilde{P}^\top H \tilde{P}\beta = \quad & (A^{-1})^\top (T^{-1})^\top (K_{nm}^\top K_{nm} + \lambda n K_{mm}) T^{-1} A^{-1} \boldsymbol{\beta} && (8) \\ = \quad & (A^{-1})^\top [(T^{-1})^\top K_{nm}^\top K_{nm} T^{-1} + \lambda n I] A^{-1} \boldsymbol{\beta}. && (9) \end{aligned}$$

Blockwise K_{nm}-vector product on GPU. The conjugate gradient algorithm will repeatedly execute Eq. (9) for different $\boldsymbol{\beta}$. The most expensive operations are the matrix-vector products $K_{nm}^\top (K_{nm} \boldsymbol{v})$ for an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^{m \times 1}$ which – if computed explicitly – would require $n \times m$ memory. However, it is possible to split the input data $X \in \mathbb{R}^{n \times d}$ in B batches of q rows each $\{X_{b,:} \in \mathbb{R}^{q \times d}\}_{b=1}^{B}$, so that matrix-vector products can be accumulated between batches using the formula $\sum_{b=1}^{B} k(X_{b,:}, X_m)^\top (k(X_{b,:}, X_m)\boldsymbol{v})$. The matrix blocks to be held in memory are summarized in Figure 2 for a total size of $m \times (m + d + 1) + q \times d$ where q can be small under memory pressure, or large for greater performance. It is important to note that $k(X_{b,:}, X_m)$ is never stored in main memory, as all operations on it are done on the GPU.

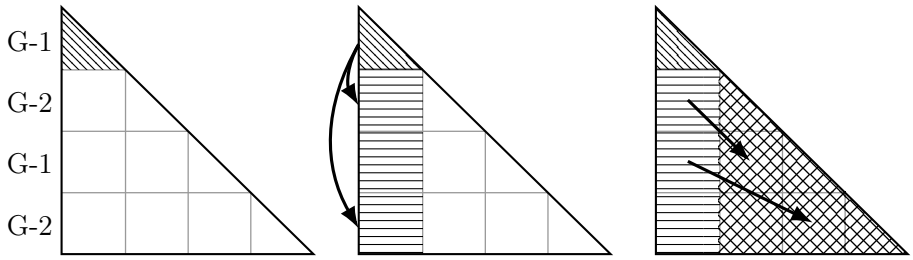


Figure 5: Three phases of the block Cholesky decomposition for updating the first column. Arrows indicate inter-GPU memory transfers between accelerators G-1 and G-2.

3.2 Fitting in GPU memory and dealing with multiple GPUs

While the main RAM might be a bottleneck, GPUs have an even smaller amount of memory, and another level of splitting is needed to exploit their speed. For example, a typical architecture has 256GB of RAM and 4 GPUs with 16GB ram each; a preconditioner with $m = 2 \times 10^5$ occupies 150 GB and K_{nm} with $n = 10^7$ would need 2000 GB of memory if stored. So we need to deal with both efficient computation of K_{nm}-vector product in chunks that fit a GPU, and with the computation of the preconditioner that usually does not fit in GPU memory. Operations based on a large storage layer (main RAM) and a small but fast layer (GPU) are called out-of-core (OOC) operations. However, common machine learning libraries such as Tensorflow [1] or PyTorch [35] do not implement OOC versions of the required matrix operations, leaving potentially complex implementations to the users. Hence, in our library, we provide these implementations in easily reusable form. It is important to note that splitting our workload to fit in GPU also provides an easy path to parallelization in a multi-GPU system: new chunks of computation are assigned to the first free GPU, effectively redistributing the workload between multiple accelerators when available.

Optimized block decomposition for out-of-core K_{nm}-vector multiplication. As seen in the previous section, matrix-vector products can be split along the dimension n, resulting in independent chunks of work that need to be summed up at the end. The OOC product between a kernel matrix and a vector proceeds by: (a) transferring a block of data onto the device, (b) computing the kernel on device and multiplying it by the vector, (c) copying the result back to the host. This sequence of operations minimizes expensive data-transfers between host and device since the kernel matrix is never moved. In particular, the computation is also split along dimensions m and d, to maximize the ratio between computational complexity and transfer time: i.e., maximizing $\frac{qrs}{qs+ds}$ subject to $qs + ds \leq G$, where q, r and s are the batch dimensions along n, m and d respectively, and G is the available GPU memory.

Out-of-core multi-GPU Cholesky decomposition. Other operations, such as Cholesky decomposition and triangular matrix multiplication (lines 15, 16, 17 of Algorithm 1), can also benefit from GPU execution. Here we describe, at a high level, our algorithm for multi-GPU OOC Cholesky decomposition inspired by [27, 64]. We leave further details to Appendix C.

Consider a symmetric matrix A, split into $B \times B$ tiles $A_{ij} \in \mathbb{R}^{t \times t}, i \in [B], j \in [B]$, assumed of equal size for brevity. We want a factorization $A = LL^\top$, where L is lower triangular, with the formula $A_{i,j} = \sum_{k=1}^{j} L_{i,k} L_{j,k}^\top$. The algorithm runs in-place, updating one column of A at a time. Each column update proceeds in three steps, illustrated in Figure 5. Clearly $A_{1,1} = L_{1,1} L_{1,1}^\top$ so we compute $L_{1,1}$ by a Cholesky decomposition on tile $A_{1,1}$ which is small and can be done entirely on the GPU (e.g. with cuSOLVER [33]). Then we consider the other tiles of the first block column of L for which $A_{j,1} = L_{j,1} L_{1,1}^\top$ with $j > 1$. Since we know $L_{1,1}$ from the first step, we obtain $L_{j,1} = A_{j,1} L_{1,1}^{-\top}$ for all $j > 1$ by solving a triangular system (on the GPU). Finally the first block column of L is used to update the trailing submatrix of A. Note that $A_{i,j} = \sum_{k=1}^{j} L_{i,k} L_{j,k}^\top = L_{i,1} L_{j,1}^\top + \sum_{k=2}^{j} L_{i,k} L_{j,k}^\top$ for $2 \leq j \leq i$, so we can update the trailing submatrix as $A_{i,j} = A_{i,j} - L_{i,1} L_{j,1}^\top$. We implemented a parallel version of the above algorithm which distributes block-rows between the available processors in a 1D block-cyclic way (e.g. Figure 5 (left): rows 1 and 3 are assigned to GPU-1, rows 2 and 4 are assigned to GPU-2). For each column update, one processor executes the first step and transfers the result to the others (the arrows in Figure 5), which can then execute step 2 in parallel. To update the trailing matrix, further data transfer between devices may be necessary. The tile-size is chosen as a function of GPU memory: each device needs to hold one block column plus a single block at any given time. An analysis of the scalability of our implementation is in Appendix C.

3.3 Optimizing data transfers and other improvements.

The speed of computations on GPUs is such that data transfers to and from the devices become significant bottlenecks. We have described earlier how, for matrix-vector products, the computed blocks of K_{nm} never leave the device. Further, optimization is possible by parallelizing computations and data transfers. Indeed, modern GPUs have an independent and parallel control on the following activities: loading from RAM, saving to RAM, performing computations. By running three parallel threads for the same GPU and assuming equal duration of each piece of work, we can run t GPU computations in $t + 2$ time units instead of $3t$ time units for a serial implementation (see Figure 3, where $t = 3$). This guarantees near optimal usage of the GPU and in practice corresponds to a considerable speed up of matrix-vector products.

Leveraging the trade-off numerical precision / computational power. GPUs are designed to achieve peak performance with low precision floating point numbers, so much that going from 64 to 32-bit floats can correspond (depending on the exact architecture) to $\approx 10\times$ throughput improvement. However, changing precision can lead to unexpected problems. For example, computing the Gaussian kernel is commonly done by expanding the norm $\|\boldsymbol{x} - \boldsymbol{x}'\|^2 = \boldsymbol{x}^\top \boldsymbol{x} - 2\boldsymbol{x}^\top \boldsymbol{x}' + \boldsymbol{x}'^\top \boldsymbol{x}'$, but in high dimensions $\|\boldsymbol{x}\|, \|\boldsymbol{x}'\|$ can be very large and the cross-term very negative, so their sum has fewer significant digits. Loss of precision can lead to non positive-definite kernels causing Cholesky decomposition to fail. To avoid this, we compute K_{mm} in blocks, converting each block to 64-bit precision for the sum, and then back to 32-bits.

Dealing with thin submatrices. As a result of our block division strategies, it may happen

that blocks become thin (i.e. one dimension is small). In this case, matrix operations, e.g. using cuBLAS [32], cannot leverage the full computational power. In turn this can reduce performance, breaking the inherent computational symmetry among GPUs which is crucial for the effectiveness of a parallel system like the one proposed in this paper. To guarantee good performance for this case, instead of using standard GPU operations, we perform matrix-vector products using KeOps [8]: a specialized library to compute kernel matrices very efficiently when one dimension is small, see Table 1.

Dealing with sparse datasets. On the other side of the spectrum, sparse datasets with high dimensionality are common in some areas of machine learning. While the kernel computed on such datasets will be dense, and thus can be handled normally, it is inefficient and in some cases impossible (e.g. with $d \sim 10^6$ as is the case for the YELP dataset we used) to convert the inputs to a dense representation. We therefore wrapped specialized sparse linear algebra routines to perform sparse matrix multiplication [34], and adapted other operations such as the row-wise norm to sparse matrices. Thus our library handles sparse matrices with no special configuration, both on the GPU and – if a GPU is not available – on the CPU.

4 Large-scale experiments

We ran a series of tests to evaluate the relative importance of the computational solutions we introduced, and then performed extensive comparisons on real-world datasets. The outcome of the first tests is given in Table 1 and is discussed in Appendix A.1 for brevity. In summary, it shows a 20× improvement over the base implementation of [42] which runs only partially on the GPU. Such improvement is visible in equal parts for the preconditioner computations, and for the iterative CG algorithm. For the second series of experiments we compared our implementation against three other software packages for GPU-accelerated kernel methods on several large scale datasets. All experiments were run on the same hardware, with comparable amounts of hyperparameter tuning. Finally we compared the results of our library against a comprehensive list of competing kernel methods found in the literature. We will denote our implementation by **Falkon** for squared loss and by **LogFalkon** for logistic loss. Next we present the algorithms we will compare with, then shortly describe the datasets used and the experimental setting, and finally show the benchmark results. More details are in Appendix A.

Algorithms under test. We compare against the following software packages: EigenPro [29], GPflow [57] and GPyTorch [15]. The first library implements a KRR solver based on preconditioned block-coordinate gradient descent where the preconditioner is based on a truncated eigendecomposition of a data subsample. EigenPro provides a fully in-core implementation and therefore does not scale to the largest datasets we tried. On some datasets EigenPro required the training data to be subsampled to avoid GPU memory issues. The other two packages implement several GP approximations and exact solvers, and we had to choose the model which would give a more appropriate comparison: we decided to avoid deep GPs [13, 62, 11] since they share more similarities to deep nets than to kernel methods; on the other hand the exact GP – even when implemented on GPU [15, 58] – as well as structured kernel interpolation [61, 16]

Table 1: Relative performance improvement of the implemented optimizations w.r.t. [42]. The experiment was run with the HIGGS dataset, 1×10^5 centers and 10 conjugate gradient iterations.

Experiment	Preconditioner		Iterations	
	Time	Improvement	Time	Improvement
Falkon from [42]	2337 s	–	4565 s	–
Float32 precision	1306 s	1.8×	1496 s	3×
GPU preconditioner	179 s	7.3×	1344 s	1.1×
2 GPUs	118 s	1.5×	693 s	1.9×
KeOps	119 s	1×	232 s	3×
Overall improvement		19.7×		18.8×

approximations do not scale to the size of datasets we are interested in. The only GP models which would scale up to tens of millions of points are stochastic variational GPs (SVGP). The SVGP is trained in minibatches by maximizing the ELBO objective with respect to the variational parameters and the model hyperparameters. Stochastic training effectively constrains GPU memory usage with the minibatch size. Hyperparameters include kernel parameters (such as the length-scale of the RBF kernel) as well as the inducing points which – unlike in Falkon – are modified throughout training using gradient descent. For this reason SVGP works well even with very few inducing points, and all operations can run in-core. While GP solvers are capable of estimating the full predictive covariance, we ensured that the software did not compute it, and further we did not consider prediction times in our benchmarks. Furthermore we always considered the Gaussian kernel with a single length-scale, due to the high effort of tuning multiple length-scales for Falkon, although for GPs tuning would have been automatic. Both GPyTorch and GPflow implement the same SVGP model, but we found the best settings on the two libraries to be different; the discrepancies in running time and accuracy between the two GP libraries come from implementation and tuning differences. We ran all algorithms under as similar conditions as possible: same hardware, consistent software versions, equal floating-point precision and equal kernels (we always considered the Gaussian kernel with a single length-scale). Hyperparameters were optimized manually by training on a small data subset, to provide a sensible trade off between performance and accuracy: we increased the *complexity* of the different algorithms until they reached high GPU utilization since this is often the knee in the time-accuracy curve. Details on the GP likelihoods, optimization details and other settings used to run and tune the algorithms are in Appendix A.4.

Datasets. We used eight datasets which we believe represent a broad set of possible scenarios for kernel learning in terms of data size, data type and task ranging from MSD with 5×10^5 points up to TAXI with 10^9 points and YELP with 10^7 sparse features. The characteristics of

the datasets are shown in table 2 while a full description, along with details about preprocessing and relevant data splits, is available in appendix A.3.

Experimental setting. All experiments were run on a Dell PowerEdge server with 2 Intel Xeon 4116 CPUs, 2 Titan Xp GPUs and 256GB of RAM. Since out of the analyzed implementations only Falkon could use both GPUs effectively, we ran it both in a 2-GPU configuration (see Table 2) and in a single-GPU configuration (see in appendix Table 4) where Falkon was on average 1.6× slower. Each experiment was run 5 times, varying the random train/test data split and the inducing points. Out of all possible experiments, we failed to run GPyTorch on TIMIT due to difficulties in setting up a multi-class benchmark (this is not a limitation of the software). Other experiments, such as EigenPro on several larger datasets, failed due to memory errors and others yet due to software limitations in handling sparse inputs (none of the examined implementations could run the sparse YELP dataset). Finally, LogFalkon only makes sense on binary classification datasets.

Results. We show the results in Table 2. In all cases, our library converges in less time than the other implementations: with an average speedup ranging from 6× when compared to EigenPro to $> 10\times$ when compared to GPyTorch. Only on very few datasets such as AIRLINE-CLS, GPflow gets closer to Falkon's running time. Both models had worse accuracy than Falkon. EigenPro has generally high accuracy but can not handle large datasets at all. Finally, LogFalkon provides a small but consistent accuracy boost on binary classification problems, at the expense of higher running time. Compared with the original Falkon library [42] we report slightly higher error on HIGGS; this is attributable to the use of low-precision floating point numbers. We did not find significant performance differences for other datasets. We defer comparisons with results from the literature to Appendix A.6; suffice it to note that a distributed GP applied to the TAXI dataset resulted in a running-time of 6000 s using a system with 28 000 CPUs [36] while we achieved similar accuracy in less time, with a much smaller computational budget.

5 Conclusions

Making flexible and easy to use machine learning libraries available is one of the keys of the recent success of machine learning. Here, we contribute to this effort by developing a library for large scale kernel methods. We translate algorithmic ideas into practical solutions, using a number of carefully design computational approaches specifically adapted to the GPU. The resulting library achieves excellent performance both in terms of accuracy and computational costs. A number of further developments are possible building on our work. For example, considering other loss functions or optimization approaches, and especially more structured kernels [9] that could further improve efficiency.

Table 2: Accuracy and running-time comparisons on large scale datasets.

	TAXI $n \approx 10^9$		HIGGS $n \approx 10^7$		YELP $n \approx 10^6, d \approx 10^7$	
	RMSE	time	$1-$AUC	time	rel. RMSE	time
Falkon	**311.7±0.1**	**3628±2 s**	0.1804±0.0003	**443±2 s**	**0.810±0.001**	**1008±2 s**
LogFalkon	—		**0.1787±0.0002**	2267±5 s	—	
EigenPro	FAIL		FAIL		FAIL	
GPyTorch	315.0±0.2	37 009±42 s	0.1997±0.0004	2451±13 s	FAIL	
GPflow	313.2±0.1	30 536±63 s	0.1884±0.0003	1174±2 s	FAIL	
	TIMIT $n \approx 10^6$		AIRLINE $n \approx 10^6$		MSD $n \approx 10^5$	
	c-error	time	rel. MSE	time	rel. error	time
Falkon	32.27±0.08 %	**288±3 s**	**0.758±0.005**	**245±5 s**	$(4.4834\pm0.0008)\times10^{-3}$	**62±1 s**
EigenPro	**31.91±0.01 %**	1737±8 s	0.785±0.005	1471±11 s[1]	$\mathbf{(4.4778\pm0.0004)\times10^{-3}}$	378±8 s
GPyTorch	—		0.793±0.005	2069±50 s	$(4.5004\pm0.0010)\times10^{-3}$	502±2 s
GPflow	33.78±0.14 %	2672±10 s	0.782±0.005	1297±2 s	$(4.4986\pm0.0005)\times10^{-3}$	525±5 s
	AIRLINE-CLS $n \approx 10^6$		SUSY $n \approx 10^6$			
	c-error	time	c-error	time		
Falkon	31.5±0.2 %	**186±1 s**	19.67±0.02 %	**22±0 s**		
LogFalkon	**31.3±0.2 %**	1291±3 s	**19.58±0.03 %**	83±1 s		
EigenPro	32.5±0.2 %	1629±1 s[1]	20.08±0.55 %	90±0 s[2]		
GPyTorch	32.5±0.2 %	1436±2 s	19.69±0.03 %	882±9 s		
GPflow	32.3±0.2 %	1039±1 s	19.65±0.03 %	560±11 s		

[1]Using a random subset of 1×10^6 points for training. [2]Using a random subset of 6×10^5 points for training.

Acknowledgments

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216, and the Italian Institute of Technology. We gratefully acknowledge the support of NVIDIA Corporation for the donation of the Titan Xp GPUs and the Tesla k40 GPU used for this research. Part of this work has been carried out at the Machine Learning Genoa (MaLGa) center, Università di Genova (IT) L. R. acknowledges the financial support of the European Research Council (grant SLING 819789), the AFOSR projects FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), and the EU H2020-MSCA-RISE project NoMADS - DLV-777826. This work was funded in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry

Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

[2] Hartwig Anzt, Stanimire Tomov, Piotr Luszczek, William Sawyer, and Jack Dongarra. Acceleration of GPU-based Krylov solvers via data transfer reduction. *International Journal of High Performance Computing Applications*, 29(3):366–383, 2015. doi: 10.1177/1094342015580139.

[3] Haim Avron, Kenneth L. Clarkson, and David P. Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1116–1138, 2017. doi: 10.1137/16M1105396.

[4] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Proceedings of the Annual Conference on Computational Learning Theory*, 2013.

[5] Daniele Calandriello and Lorenzo Rosasco. Statistical and computational trade-offs in kernel k-means. In *Advances in Neural Information Processing Systems 31*, 2018.

[6] A. Caponnetto and Ernesto De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7:331–368, 2007. doi: 10.1007/s10208-006-0196-8.

[7] Bryan Catanzaro, Narayanan Sundaram, and Kurt Keutzer. Fast support vector machine training and classification on graphics processors. In *Proceedings of 25th the Conference on Machine Learning*, 2008.

[8] Benjamin Charlier, Jean Feydy, Joan Alexis Glaunès, and Ghislain Durif. KeOps, 2020. URL `https://github.com/getkeops/keops`.

[9] Jie Chen, Haim Avron, and Vikas Sindhwani. Hierarchically compositional kernels for scalable nonparametric learning. *Journal of Machine Learning Research*, 18(1):2214–2255, 2017.

[10] Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39, 2001. doi: 10.1090/S0273-0979-01-00923-5.

[11] Kurt Cutajar, Edwin V. Bonilla, Pietro Michiardi, and Maurizio Filippone. Random feature expansions for deep Gaussian processes. In *Proceedings of the 34th Conference on Machine Learning*, 2017.

[12] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina F. Balcan, and Le Song. Scalable kernel methods via doubly stochastic gradients. In *Advances in Neural Information Processing Systems 27*, 2014.

[13] Andreas Damianou and Neil Lawrence. Deep Gaussian processes. In *Proceedings of the 16th Conference on Artificial Intelligence and Statistics*, 2013.

[14] Petros Drineas and Michael W. Mahoney. On the nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

[15] Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems 31*, 2018.

[16] Jacob R. Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q. Weinberger, and Andrew G. Wilson. Product kernel interpolation for scalable gaussian processes. In *Proceedings of the 21st Conference on Artificial Intelligence and Statistics*, pages 1407–1416, 2018.

[17] Alex Gittens and Michael W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. *Journal of Machine Learning Research*, 17:3977–4041, 2016.

[18] James Hensman, Nicolò Fusi, and Neil D. Lawrence. Gaussian processes for big data. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013.

[19] James Hensman, Alexander G. Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. In *Proceedings of the 18th Conference on Artificial Intelligence and Statistics*, 2015.

[20] James Hensman, Nicolas Durrande, and Arno Solin. Variational fourier features for Gaussian processes. *Journal of Machine Learning Research*, 18(1):5537–5588, 2017.

[21] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. In *Proceedings of the 21st Conference on Artificial Intelligence and Statistics*, 2018.

[22] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*, 2018.

[23] George S. Kimeldorf and Grace Wahba. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *Annals of Mathematical Statistics*, 41 (2):495–502, 1970. doi: 10.1214/aoms/1177697089.

[24] Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. Sampling methods for the nyström method. *Journal of Machine Learning Research*, 13:981–1006, 2012.

[25] Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood: Approximating kernel expansions in loglinear time. In *Proceedings of the 30th Conference on Machine Learning*, 2013.

[26] Zhu Li, Jean-Francois Ton, Dino Oglic, and Dino Sejdinovic. Towards a unified analysis of random Fourier features. In *Proceedings of the 36th Conference on Machine Learning*, 2019.

[27] Hatem Ltaief, Stanimire Tomov, Rajib Nath, Peng Du, and Jack Dongarra. A scalable high performant Cholesky factorization for multicore with GPU accelerators. In *High Performance Computing for Computational Science*, 2011. doi: 10.1007/978-3-642-19328-6_11.

[28] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems 30*, 2017.

[29] Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to GPUs for effective large batch training. In *Proceedings of the 2nd Conference on Machine Learning and Systems*, 2019.

[30] Ulysse Marteau-Ferey, Francis Bach, and Alessandro Rudi. Globally convergent newton methods for ill-conditioned generalized self-concordant losses. In *Advances in Neural Information Processing Systems 32*, 2019.

[31] Ulysse Marteau-Ferey, Dmitrii Ostrovskii, Francis Bach, and Alessandro Rudi. Beyond least-squares: Fast rates for regularized empirical risk minimization through self-concordance. In *Proceedings of the Thirty-Second Conference on Learning Theory*, 2019.

[32] NVIDIA Corporation. cuBLAS, 2020. URL `https://developer.nvidia.com/cublas`.

[33] NVIDIA Corporation. cuSOLVER, 2020. URL `https://developer.nvidia.com/cusolver`.

[34] NVIDIA Corporation. cuSPARSE, 2020. URL `https://developer.nvidia.com/cusparse`.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.

[36] Hao Peng, Shandian Zhe, Xiao Zhang, and Yuan Qi. Asynchronous distributed variational gaussian process for regression. In *Proceedings of the 34th Conference on Machine Learning*, 2017.

[37] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, 2008.

[38] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21*, 2009.

[39] Carl Edwards Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[40] Alessandro Rudi and Lorenzo Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems 30*, 2017.

[41] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. In *Advances in Neural Information Processing Systems 28*, 2015.

[42] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. FALKON: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems 29*, 2017.

[43] Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. SIAM, 2003.

[44] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001. doi: 10.7551/mitpress/4175.001.0001.

[45] Bernhard Schölkopf, Ralf Herbrich, and Alexander J. Smola. A generalized representer theorem. In *Proceedings of the Annual Conference on Computational Learning Theory*, 2001.

[46] Laurent Schwartz. Sous-espaces hilbertiens d'espaces vectoriels topologiques et noyaux associés (noyaux reproduisants). *Journal d'Analyse Mathématique*, 13:115–256, 1964. doi: 10.1007/BF02786620.

[47] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[48] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[49] Alex J. Smola and Bernhard Schökopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the 17th Conference on Machine Learning*, 2000.

[50] Ingo Steinward and P. Thomann. liquidSVM: A fast and versatile SVM package, 2017.

[51] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[52] Ingo Steinwart, Don Hush, and Clint Scovel. Optimal rates for regularized least squares regression. In *Proceedings of the Annual Conference on Computational Learning Theory*, 2009.

[53] Nicholas Sterge, Bharath Sriperumbudur, Lorenzo Rosasco, and Alessandro Rudi. Gain with no pain: Efficient kernel-PCA by nyström sampling. In *Proceedings of the 23rd Conference on Artificial Intelligence and Statistics*, 2020.

[54] Yitong Sun, Anna Gilbert, and Ambuj Tewari. But how does it work in theory? linear SVM with random features. In *Advances in Neural Information Processing Systems 31*, 2018.

[55] Dougal Sutherland, Heiko Strathmann, Michael Arbel, and Arthur Gretton. Efficient and principled score estimation with nyström kernel exponential families. In *Proceedings of the 21nd Conference on Artificial Intelligence and Statistics*, 2018.

[56] Stephen Tu, Rebecca Roelofs, Shivaram Venkataraman, and Benjamin Recht. Large scale kernel learning using block coordinate descent, 2016.

[57] Mark van der Wilk, Vincent Dutordoir, S. T. John, Artem Artemev, Vincent Adam, and James Hensman. A framework for interdomain and multioutput Gaussian processes, 2020.

[58] Ke Wang, Geoff Pleiss, Jacob Gardner, Stephen Tyree, Kilian Q Weinberger, and Andrew Gordon Wilson. Exact Gaussian processes on a million data points. In *Advances in Neural Information Processing Systems 32*, 2019.

[59] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801, 2018.

[60] Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, 2001.

[61] Andrew G. Wilson and Hannes Nickisch. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the 32nd Conference on Machine Learning*, 2015.

[62] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. Stochastic variational deep kernel learning. In *Advances in Neural Information Processing Systems 30*, 2016.

[63] Nicholas Wilt. *The CUDA handbook: A comprehensive guide to GPU programming*. Pearson Education, 2013.

[64] R. Wu. A heterogeneous parallel cholesky block factorization algorithm. *IEEE Access*, 6, 2018. doi: 10.1109/ACCESS.2018.2803794.

[65] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems 24*, 2012.

[66] Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 16(102):3299–3340, 2015.

A Further experiment details and results

A.1 Relative impact of performance optimizations

We performed an experiment to analyze how much improvement was due to the different performance optimization steps. We ran Falkon on the HIGGS dataset several times with the same hyperparameters ($m = 1{\times}10^5$ and 10 epochs), but with different *features* enabled. Each feature roughly corresponds to one of the performance optimizations discussed in Section 3. Our baseline model is very similar to the original Falkon implementation [42], where the preconditioner ran on the CPU, float64 precision was being used, but matrix-vector multiplications for the CG algorithm were GPU accelerated. As a first optimization we used float32 precision for all computations, with care taken to avoid errors in the Cholesky decomposition as discussed in Section 3. This immediately resulted in a 2× speedup for the CPU part, and 3× for the GPU part. Switching to a GPU preconditioner (using the algorithms described in Appendix C) gave a huge boost to the preconditioner running time which went from more than 20 min to just under 3 min. Adding a second GPU produced a perfect 2× speedup for the CG iterations, and a more modest 1.5× speedup for the preconditioner which a) involves operations which are not perfectly parallelizable and b) incurs in some fixed startup costs. Finally, since the HIGGS dataset has only 9 features (thus the data matrix is thin), we can use KeOps [8] with great benefits to the speed of matrix-vector multiplications. Overall our implementation provides a nearly 20× improvement over the baseline, which makes learning on several huge datasets doable in a matter of minutes.

A.2 Multi-GPU scalability

In this section we look into the scalability of our implementation across multiple GPUs. Scalability results for the full Falkon algorithm on the TAXI dataset are shown in Figure 6. This result depends on scaling both the preconditioner and the conjugate gradient iterations. The preconditioner itself is computed with three main operations: two Cholesky decompositions and one triangular matrix multiplication (this is called the LAUUM operation in LAPACK terms), see Figure 4 for more details. Each CG iteration instead consists of two multiplications between the kernel matrix and an arbitrary vector. First we look at the scalability of the preconditioner operations with multiple GPUs. Then we examine our out-of-core matrix-vector product implementation and compare it to KeOps for different settings of n and d.

Preconditioner scalability. Figure 7 shows the results from running both triangular matrix multiplication and the Cholesky decomposition with one and two GPUs. At low matrix sizes the speedup with two GPUs is negligible, especially for the Cholesky decomposition. In such cases it is best to use a single GPU (especially since for $n = 40000$ the whole matrix fits in GPU memory, so an in-place decomposition can be used). With higher matrix sizes, having more than one GPU starts bringing real benefits, with a peak speedup around 1.8× for preconditioners of size 140 000. The factors blocking such speedup from reaching a perfect 2×

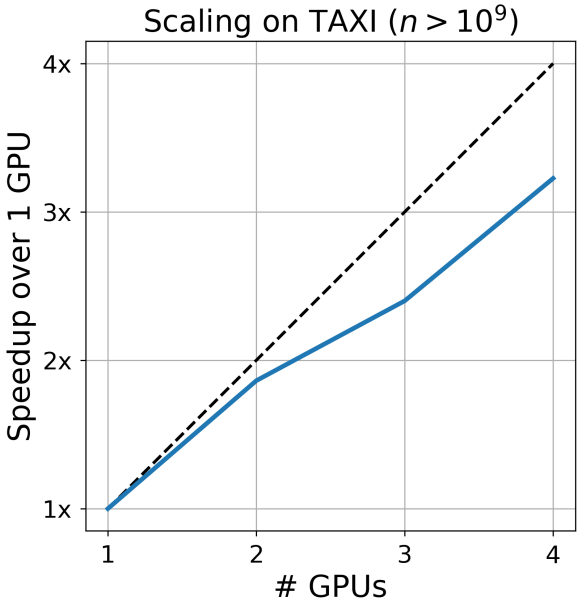
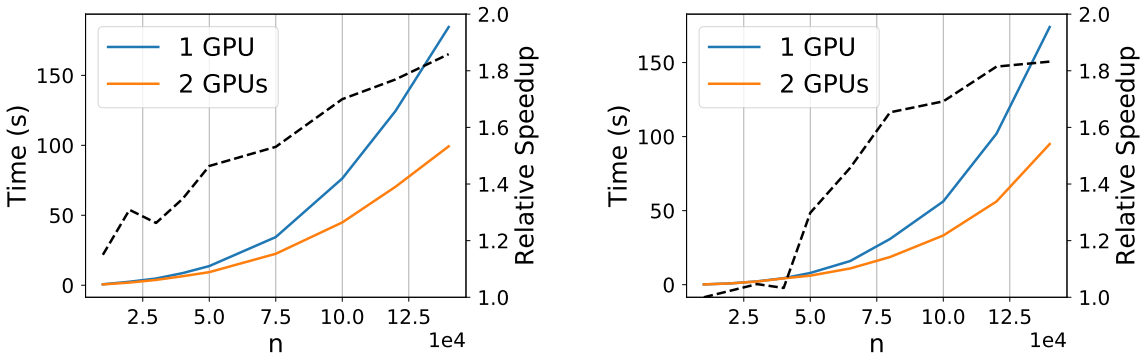


Figure 6: Multi-GPU scalability of Falkon on the TAXI dataset (settings are the same as per Table 3). Falkon scales remarkably well, with even 4 GPUs.

are different for the two operations. Since the LAUUM operation was run out-of-place (see Appendix C for more details), it does not need any synchronization – and should therefore be able to scale well across multiple GPUs. The main blocking factor is the operation at Line 7 of Algorithm 3 which is executed on the CPU (since an equivalent implementation does not exist in cuSOLVER), thus both GPU threads must share the same CPU resources. We left porting the LAUUM operation to the GPU as future work, but it has the potential to speed up the LAUUM operation considerably. For the Cholesky decomposition the limiting factors are the data-dependencies intrinsic to the algorithm which cannot be easily solved.

Comparing different MVM implementations. We compare our specialized routine for the kernel-vector multiplication $k(X^{(1)}, X^{(2)})\boldsymbol{v}$ implemented in Python, leveraging PyTorch for GPU computations, against the native CUDA implementation from KeOps [8]. Using a similar notation for the dimensions as in the main text we have $X^{(1)} \in \mathbb{R}^{n\times d}, X^{(2)} \in \mathbb{R}^{m\times d}, \boldsymbol{v} \in \mathbb{R}^{m\times 1}$ and $k(\cdot,\cdot)$ is a kernel function. Two distinct scenarios arise in different settings: increasing the number of data points n produces linear scaling for both implementations, with KeOps being approximately 10 times faster than our implementation (see Figure 8(a)). Increasing the data dimensionality d our implementation scales linearly, but KeOps scales polynomially, so as it is obvious from Figure 8(b) KeOps can not be used when the data is high-dimensional. A caveat of this plot is that KeOps is continuously evolving, and is likely to improve performance with large d in the future. In our final algorithm we set a threshold on the data dimensionality and



(a) Parallel LAUUM.

(b) Parallel Cholesky decomposition.

Figure 7: Running time of two preconditioner operations with one and two GPUs. The relative speed-up with 2 GPUs is shown in the black dashed line. The LAUUM operation (triangular matrix multiplication) was run out-of-place, which is theoretically easier to parallelize, while the Cholesky decomposition was run in-place.

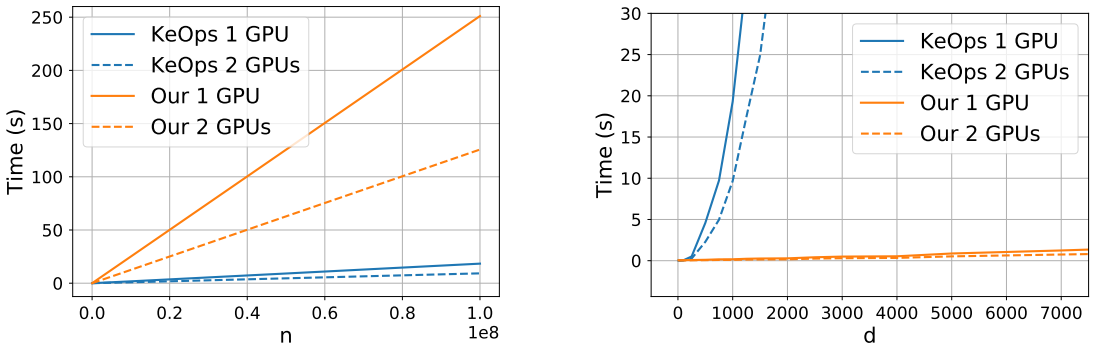
switch implementation based on this. Finally note that this operation scales almost perfectly with multiple GPUs.

A.3 Additional information on the datasets

We used several datasets which we believe represent a broad set of scenarios for kernel learning, in terms of data size, data type, and learning task. We normally used a standard random split with 80% training, 20% testing data unless predefined splits existed (as noted below). Preprocessing mostly consisted in basic data cleaning and data standardization to zero mean and unit standard deviation; we comment in more detail below on specific preprocessing steps applied to the individual datasets.

HIGGS has dimensions $n = 1.1 \times 10^7, d = 28$ and a binary target. It was preprocessed to 0 mean and unit variance. Results are reported on a 80-20 split with 1 minus the AUC metric in Table 2 and with the binary classification error in Table 6. It is available for download at `https://archive.ics.uci.edu/ml/datasets/HIGGS`.

TIMIT has dimensions $n = 1.2 \times 10^6, d = 440$ and a multiclass target with 144 classes. TIMIT comes from audio data, and our dataset uses the 10 ms resampling rate as in [28, 29]. It was preprocessed to 0 mean and unit standard deviation. The error metric is classification error on a subset of classes (as used in [28]), and is calculated over a standardized subset of 57 242 samples. It is available for download at `https://catalog.ldc.upenn.edu/LDC93S1`.



(a) (b)

Figure 8: Scaling of matrix-vector implementations where the matrix is the Gaussian kernel. In (a) we have set $m = 20\,000$, $d = 10$ and n is variable; in (b) we set $m = n = 20\,000$ and we vary d. All experiments are run on 1 and 2 GPUs on single precision random data.

YELP has dimensions $n = 1.5 \times 10^6, d = 6.52 \times 10^7$ and a continuous target. This dataset consists of text reviews, labeled with their star rating. We used the same data as [56] (Yelp round 9 dataset), processed by extracting all 3-grams and encoding each review by a count vector which tells us which 3-grams are present. Such encoding produces a large number of sparse features which is reflected in the huge dimensionality of this dataset. Since the data is sparse we did not normalize it. The error metric is RMSE, calculated on random 20% of the samples. The dataset can be provided on request.

TAXI has dimensions $n = 1.1 \times 10^9, d = 9$ with a continuous target. Data are normalized to have zero-mean and unit standard deviation; reported error is RMSE on a 20% random sub-sample. The data can be downloaded by following instructions at `https://github.com/toddwschneider/nyc-taxi-data`. Consistently with other users of this dataset [36] we took the data from January 2009 to December 2015, excluding outliers (taxi trips more than 5 hours long) and trips where the pickup or drop off location is outside of NYC.

AIRLINE has dimensions $n = 5.93 \times 10^6, d = 8$ and a continuous target. Data are normalized to zero-mean and unit standard deviation, and the error is the MSE over normalized targets calculated on random test-sets of size 33 % of the full data (consistently with the literature [20, 18]). The same dataset is also used for binary classification by thresholding the target at 0, which results in the **AIRLINE-CLS** dataset. For this latter variation we used 100 000 random points for testing, reporting classification error in Table 2 and 1 minus the AUC in Table 6 to facilitate comparisons with the literature. The data can be downloaded from `https://www.transtats.bts.gov/Fields.asp?Table_ID=236` and `http://stat-computing.org/dataexpo/2009/supplemental-data.html`.

MSD has dimensions $n = 5.1\times10^5, d = 90$ with continuous target. Data are normalized to zero-mean and unit standard deviation, and we report the relative error over a standard test-set of size 51 630. The dataset can be downloaded from `https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD`.

SUSY has dimensions $n = 5\times10^6, d = 18$ with binary target. Data are normalized to zero-mean and unit standard deviation. We report the classification error on 20% of the data. Data is available from the UCI repositories `https://archive.ics.uci.edu/ml/datasets/SUSY`.

A.4 Additional information on the experimental settings

1. EigenPro2. Its only hyperparameters – other than the kernel parameters – are the ones governing the preconditioner's complexity making EigenPro easy to tune. It is however limited to datasets which fit entirely in GPU memory, so can not easily scale to larger datasets; to alleviate this problem, consistently with the original paper, some experiments were run on sub-sampled datasets. Furthermore, on some experiments we found it necessary to manually tune the learning rate (we divided the automatically inferred learning rate by a fixed integer, denoted by $\eta\div$ in Table 3).

2. GPFlow (v2.1.3). We used the SVGP model with Gaussian likelihood for regression, Bernoulli for binary classification and Softmax for multi-class problems. We used Adam for optimization and tuned the learning rate, the number of inducing points, and the constraints on the variational distribution covariance (i.e. diagonal or full covariance matrix). We found that using a full covariance matrix was rarely beneficial and increased training times slightly, so all final experiments used a diagonal covariance matrix. The number of parameters was $m \times d + m \times 2 + 3$ which includes the inducing points, the variational parameters, two parameters for the Gaussian kernel (lengthscale and variance) and the variance of the likelihood. For multi-class problems separate variational parameters were trained for each class. Since we wished to use single-precision floating point numbers in order to make GPU training more efficient, we found that natural gradient optimization was unstable. It remains to be seen whether the tradeoff between double-precision data and natural gradient optimization could further improve results. We further tested the benefits of using whitening of the inducing points, and found that it decreased per-epoch running times by about $2\times$, while at the same time slowing down convergence by around the same amount. In practice this meant that the difference in global running time was not strongly affected by whitening, and we ended up using it only for the HIGGS data.

3. GPyTorch (v.1.2.0). We used the SVGP model with Gaussian and Bernoulli likelihoods. We were unable to run GPyTorch's SVGP model on the TIMIT dataset due to problems in dealing with multiple outputs. We used the natural gradient optimizer to learn the variational parameters, and Adam to learn the other hyperparameters. The learning

rate of the two optimizers was kept equal and tuned for best performance. We further optimized the number of inducing points, and variational distribution constraints. In practice we found that we had to use the natural gradient variational distribution for regression problem, and the lower-triangular parametrization for classification problems (which are non-conjugate). We additionally tested whether whitening the inducing points was beneficial: in practice we found that using the unwhitened strategy was around 3× faster and did not hamper convergence, so we selected it for all experiments. While GPyTorch is theoretically able to run on multi-GPU systems, we noticed that this feature was not available for the SVGP model thus we always used a single GPU; furthermore, while a KeOps integration into GPyTorch is available, we found that for the SVGP model it would increase the running time, so we did not use it. The trained parameters were the same as for GPFlow plus another scalar for the GP mean.

4. Falkon. We tuned the kernel length-scale, number of inducing points and regularization amount. We used a coarse to fine approach to tune the length-scale which gives good results with a limited number of validation runs.

5. Logistic Falkon. Here we tuned the kernel length-scale, number of inducing points and regularization path. We found that the algorithm is not very sensitive to the exact regularization path: it is sufficient to set the final λ, and many different paths which lead to such value will work in the same way.

A.5 Additional benchmarks

In Table 4 we show the performance of the Falkon algorithm on all considered datasets for 1 and 2 GPUs side by side. It is clear that larger datasets scale better with more GPUs since the startup cost (mostly taken up by CUDA initialization) and the lower scaling ratio of the preconditioner are amortized.

In Table 5 we compare the running times of Falkon and ThunderSVM [59] on three popular image datasets. ThunderSVM was chosen among several SVM implementations as it runs entirely on the GPU, and can thus solve the hinge-loss problem quickly for problems of moderate size. Smaller datasets than the ones used for previous experiments were considered, since ThunderSVM solves the full SVM problem and thus suffers from cubic time scaling. The results obtained show that Falkon can work efficiently even on smaller datasets, resulting between 2 and 10 times faster than ThunderSVM (depending on problem size), with comparable accuracy. To further shave off some time, we implemented a version of Falkon which runs entirely inside the GPU: we call this **InCoreFalkon**, and it can only be used on smaller datasets which fit inside the GPU, leaving some space to spare which is used for the preconditioner and the other computations. Table 5 shows that InCoreFalkon gives a further speed-up of – on average – 2× compared to the standard implementation.

Table 3: Summary of the most important hyperparameter settings for all algorithm-dataset combinations. We denote by η the learning rate, by *subsample* the amount of training-set subsampling that was performed (i.e. training was done on a smaller dataset), and by Newton steps the number of separate runs of the main Falkon algorithm for Logistic Falkon (see Appendix B).

		AIRLINE	AIRLINE-CLS	MSD	SUSY	TIMIT	YELP	HIGGS	TAXI
	n	5.93×10^6	5.93×10^6	5.1×10^5	5×10^6	1.2×10^6	1.6×10^6	11×10^7	1.15×10^9
	d	8	8	90	18	440	6.5×10^7	28	9
	labels	reg	2-cls	reg	2-cls	144-cls	reg	2-cls	reg
Falkon	m	1×10^5	1×10^5	5×10^4	3×10^4	1×10^5	5×10^4	1.2×10^5	1×10^5
	σ	0.9	0.9	7	3	14.5	20	3.8	1
	λ	1×10^{-8}	1×10^{-8}	2×10^{-6}	1×10^{-6}	5×10^{-9}	1×10^{-6}	3×10^{-8}	2×10^{-7}
	epochs	20	10	10	5	5	10	10	7
LogFalkon	m	–	1×10^5	–	2×10^4	–	–	1×10^5	–
	σ	–	0.9	–	3	–	–	5	–
	λ	–	1×10^{-9}	–	1×10^{-8}	–	–	1×10^{-9}	–
	Newt. steps	–	9	–	6	–	–	9	–
GPyTorch	m	2000	2000	3000	2000	–	–	2000	1000
	η	5×10^{-3}	2×10^{-3}	2×10^{-3}	1×10^{-3}	–	–	2×10^{-2}	2×10^{-3}
	epochs	20	20	20	20	–	–	15	5
GPflow	m	2000	2000	3000	2000	2000	–	2000	1000
	η	5×10^{-3}	5×10^{-3}	2×10^{-3}	3×10^{-3}	1×10^{-2}	–	2×10^{-2}	3×10^{-3}
	epochs	25	20	45	10	15	–	60	10
	whiten	no	no	no	no	no	–	yes	no
EigenPro	$\eta\div$	10	12	20	1	1	–	–	–
	subsample	1×10^6	1×10^6	–	6×10^5	–	–	–	–
	epochs	9	10	9	1	4	–	–	–

Table 4: Benchmark timings using a single GPU. The relative slowdown with respect to Falkon on 2 GPUs is also provided for comparison with Table 2.

	1 GPU	2 GPUs	Relative change
TAXI	7215±4 s	3628±2 s	1.99×
HIGGS	715±6 s	443±2 s	1.61×
YELP	1981±6 s	1008±2 s	1.97×
TIMIT	416±4 s	288±3 s	1.44×
AIRLINE	334±2 s	245±5 s	1.36×
MSD	81±0 s	62±1 s	1.31×
AIRLINE-CLS	391±5 s	269±3 s	1.45×
SUSY	29±1 s	22±0 s	1.32×

Table 5: Comparing the running times of Falkon, the in-core version of Falkon and ThunderSVM on three image datasets. Hyperparameters (especially the number of inducing points m) were tuned so that the two algorithms obtained approximately the same accuracy.

	MNIST $n = 6 \cdot 10^4, d = 780$	CIFAR10 $n = 6 \cdot 10^4, d = 1024$	SVHN $n = 7 \cdot 10^4, d = 1024$
Falkon	10.9 s	13.7 s	17.2 s
InCoreFalkon	6.5 s	7.9 s	6.7 s
ThunderSVM	19.6 s	82.9 s	166.4 s

A.6 Performance comparisons in a literature review

We scanned the literature for results which used kernel methods on the datasets considered in this paper, which reported both accuracy and running times. This allowed us to confirm that the results reported in our benchmarks (see Table 2) were in-line with what had been previously reported. The outcome is shown in Table 6. We do not report results where running time is not mentioned. Some of the numbers in Table 6 have higher accuracy than Falkon: this comes from the use of deep GPs which – through a vast number of parameters – can learn better data representations. Such models are intrinsically different in spirit from kernel methods, and we do not aim to compare with them specifically; they are reported in Table 6 for the sake of completeness.

B Logistic Falkon Algorithm

In this section we provide some more details on how to derive fast algorithms with strong theoretical guarantees for smooth loss functions beyond squared loss. In particular, the main ideas from a theoretical and algorithmic viewpoint that we are going to recall here are developed in [31], [30]. Our goal, as stated in the main text, is to make these ideas practical, by efficiently implementing and deploying the algorithms and making full use of the available computational architectures. In particular, we will focus on the following set of *generalized self concordant* loss functions:

Definition 1. Generalized self-concordant (GSC) function [31] *Let $\mathcal{H}$ be a Hilbert space and let $z = (x, y)$ be an input-output pair. We say that $\ell_z : \mathcal{H} \to \mathbb{R}$ is a generalized self-concordant function on $\mathcal{G} \subset \mathcal{H}$, when $\mathcal{G}$ is a bounded subset of $\mathcal{H}$ and ℓ_z is a convex and three times differentiable mapping on $\mathcal{H}$ such that*

$$\forall f \in \mathcal{H},\ \forall h, k \in \mathcal{H},\ \nabla^{(3)}\ell_z(f)[h, k, k] \leq \sup_{g\in\mathcal{G}} |g \cdot h|\ \nabla^2\ell_z(f)[k, k].$$

Denote by R the quantity $\sup_{g\in\mathcal{G}} \|g\| < \infty$. For many loss functions $\mathcal{G}$ is just the ball in $\mathcal{H}$ centered in zero and with radius $R > 0$, then $\sup_{g\in\mathcal{G}} |g \cdot h| = R\|h\|$). The following loss functions, which are widely used in machine learning, are generalized self-concordant

Table 6: Survey of results on the datasets we considered, as reported in the literature. We report the result of our implementation (Falkon) next to other implementations, along with the time taken and the hardware used (where available).

Dataset	Falkon		Other methods		
	error	time	error	time	reference
TAXI (metric: RMSE)	311.7±0.1	3628±2 s	309.7	6000 s on 28 000 vCPUs (AWS)	ADVGP [36]
HIGGS (metric: c-err)	25.78±0.03 %	443±2 s	32.87 %	1392 s on 14 node cluster	liquidSVM [50]
YELP (metric: RMSE)	0.810±0.001	1008±2 s	0.861	≈ 3500 s	Nyström [56]
			0.854	≈ 30 000 s on 128 machines (AWS)	Full linear kernel [56]
AIRLINE (metric: MSE)	0.758±0.005	245±5 s	0.827±0.004	265±6 s on a laptop	VFF-GP [20]
			0.791±0.005	18 360±360 s on a cluster	SVIGP [20]
MSD (metric: rel. err.)	4.48×10^{-3}	62±1 s	$\approx 4.55\times10^{-3}$	210 s on IBM POWER8	Hierarchical [9]
			4.58×10^{-3}	289 s on 8 r3.8xlarge (AWS)	Faster KRR [3]
AIRLINE-CLS (metric: AUC)	0.739±0.002	**186±1 s**	0.781±0.001	14 328 s	Varitional Deep GP [62]
			0.694	5200 s	TT-GP [21]
			0.788	1375 s	Deep TT-GP [21]
			0.665	80 000 s	cVGP[19]
			0.785	≈ 5000 s	RF Deep GPs [11]
SUSY (metric: c-err)	19.67±0.02 %	**22±0 s**	≈ 20%	≈ 2000 s on IBM POWER8	Hierarchical [9]
			19.8%	58 s on 1 Titan Xp	EigenPro 2.0 [29]

Example 1. (Application to finite-sum minimization [31]) *The following loss functions are generalized self-concordant functions:*
(a) Logistic regression: $\ell_z(f) = \log(1 + \exp(-yf(x)))$, *where* $z = (x, y)$ *with* $x \in X$ *and* $y \in \{-1, 1\}$.
(b) Robust regression: $\ell_z(f) = \varphi(f(x) - y)$ *with* $\varphi(u) = \log(e^u + e^{-u})$. *Here* $z = (x, y)$ *with* $x \in X$ *and* $y \in \mathbb{R}$
(c) Softmax regression: $\ell_z(f) = \log(\sum_{j=1}^k [f(x)]_j) - [f(x)]_y$, *where now* $f : X \to \mathbb{R}^k$, $z = (x, y)$, *with* $y \in \{1, \ldots, k\}$ *and* v_j *denotes the* j*-th column of* $v \in \mathbb{R}^k$.
(d) generalized linear models with bounded features, which include conditional random fields (see more details in [31]). Note, in particular, that the loss functions above are generalized self concordant, but not *self concordant* as discussed in [31].

For the learning problem in Eq. (1) with generalized self-concordant loss functions, a strong theoretical result analogous to the one for kernel ridge regression (3) has been obtained [31]. In particular, the regularized empirical risk minimization solution (2) with generalized self-concordant losses achieves the bound

$$L(\hat{f}_\lambda) - \inf_{f \in \mathcal{H}} L(f) = \mathcal{O}\Big(n^{-1/2}\Big), \tag{10}$$

under standard regularity conditions on the learning problem and achieves fast learning rates similar to kernel ridge regression, considering more refined regularity conditions that are a natural extension of the conditions for kernel ridge regression [31].

The paper [30] suggests to solve the regularized empirical risk minimization problem (2) for generalized self-concordant losses, by using a set of techniques that are extensions of the Falkon algorithm in [42]. In particular, the problem is cast in terms of an approximate Newton method, with pseudocode shown in function `GSC-Falkon` of Algorithm 2. Nyström method is used a first time to reduce the size of the problem, and then a second time to derive an approximate Newton step [30]. Indeed a model of the form (4) is considered and the preconditioner now plays the role of approximate Hessian, to perform the iterated approximation Newton. Given $(\tilde{x}_j, \tilde{y}_j)_{j=1}^m$ selected uniformly at random from the dataset, the approximate Hessian $\tilde{H}$ at the step k is a weighted version of the Falkon preconditioner and has the form

$$\tilde{H} = \frac{1}{m} T \tilde{D}_k T^\top + \mu_k I,$$

where T is such that $T^\top T = K_{mm}$ (e.g. it is the Cholesky decomposition of K_{mm}) and $\tilde{D}_k \in \mathbb{R}^{m \times m}$ is a diagonal matrix whose jth element is $\ell^{(2)}(f_k(\tilde{x}_j), \tilde{y}_j)$ where we assume that the loss function is $\ell(f(x), y)$ and the second derivative is taken with respect to the first variable. As for Falkon, the approximate Hessian is never built explicitly, we compute instead its Cholesky decomposition in terms of the matrices T, A as $\tilde{H}^{-1} = \tilde{P}\tilde{P}^\top$ with $\tilde{P} = T^{-1}A^{-1}$, see the function `WeightedPreconditioner` in Alg. 2. Then conjugate gradient is applied to the preconditioned problem, to solve the equation

$$\tilde{P}^\top (K_{nm}^\top D_k K_{nm} + \lambda I) \tilde{P} \beta = \tilde{P}^\top K_{nm}^\top g_k.$$

where $D_k \in \mathbb{R}^{n\times n}$ is a diagonal matrix whose ith element is $\ell^{(2)}(f_k(x_i), y_i)$ and $g_k \in \mathbb{R}^n$ corresponds to $(g_k)_i = \ell^{(1)}(f_k(x_i), y_i)$. To conclude, as proven in [30], to achieve the same learning rate of (10) and good practical performances, `GSC-Falkon` (Alg. 2) needs to call `WeightedFalkon` only a small number of times with decreasing regularization parameters. Moreover, each time `WeightedFalkon` needs to execute only few iterations of the CG algorithm. The algorithm presented in Alg. 2 has an important theoretical appeal as proved in [30] since it is the fastest to date to achieve optimal learning rates for generalized self-concordant loss functions. The goal of our work is to make it also appealing from a practical viewpoint. This requires efficiently implementing and deploying Alg. 2, making full use of the available computational architectures. Clearly the main bottlenecks here are the same of Falkon for squared loss and they are introduced and discussed in Section 3.

C Out-Of-Core Algorithms

In this section we describe more in detail the out-of-GPU core algorithms for 1) Cholesky decomposition of a positive definite matrix and 2) multiplication of a triangular matrix by its transpose. Both algorithms use a similar technique of dividing the input matrix in smaller tiles such that operations can be performed in-core on the individual tiles. Then the main challenges of such algorithms consist in choosing when to bring which tiles in-core, and how to do so in parallel, handling data-dependencies between different tiles.

We handle parallelism between multiple GPUs using a static work-allocation scheme where the input matrix is divided into block rows or columns (made up of several tiles), and each GPU is assigned one or more such rows (or columns) block-cyclically, to ensure that the workload is approximately balanced. Ensuring a balanced workload is tricky since the input matrices are triangular, and for example a row at the top of a lower-triangular matrix will have many more tiles than a row towards the bottom of said matrix. Smaller tile-sizes (so thinner block rows/columns) make each processor's workload more even, but – in case the input matrix is not big enough – they reduce overall GPU utilization.

Triangular matrix multiplication. We begin by describing OOC triangular matrix multiplication, an operation which is known as LAUUM within the LAPACK library. Given an input upper triangular matrix $U \in \mathbb{R}^{n\times n}$, we want to calculate the upper triangle of $UU^\top$ and store it in the upper part of U (thus making this an in-place operation). We divide U in $N \times N$ tiles of size t (uneven tile sizes are possible, and indeed necessary to support all input sizes, but omitted from the description for clarity), and we index all matrices by their tiles: $U_{2,2}$ is the square tile at the second block-row and second block-column of U. The in-place LAUUM operation can be compactly described as $U_{i,j} = \sum_{k=j}^{N-1} U_{i,k}U_{j,k}^\top$ for $j \geq i$: to update a tile of U we need to multiply two block-rows of the original matrix. However, we can exploit the triangular structure of some of the above matrix multiplications to improve performance: for example, when $i = j$ it is possible to split the update into two parts $U_{i,i} = U_{i,i}U_{i,i}^\top + \sum_{k=i}^{N} U_{i,k}U_{i,k}^\top$ where the first part consists of an in-core LAUUM operation and the second of a symmetric

matrix multiplication (BLAS routine SYRK) which can be up to twice as fast as the general matrix multiplication routine. Similarly, for $i < j$, $U_{i,j} = U_{i,j}U_{j,j}^\top + \sum_{k=j+1}^{N} U_{i,k}U_{j,k}^\top$ where the first part can use the TRMM routine from the BLAS library and the second must use the generic GEMM routine. To avoid overwriting parts of U which are still needed for the updates – especially in a multi-GPU setting – the rows of U are to be updated one at a time, from top to bottom. To ensure synchronization between multiple GPUs we use a thread barrier so that all GPUs start updating a given row after having loaded its original, non-updated version in GPU memory. GPU memory requirements for Algorithm 3 are two block-columns (i.e. $2Nt^2$ numbers). As discussed above, rows are assigned to GPUs in a 1D block-cyclic way. Such allocations are recorded in the `blockAllocs` variable.

An adaptation of Algorithm 3 is possible when in-place operation is not needed: it is sufficient to remove the synchronization barrier, and change line 18 to write the output to a different matrix.

Cholesky decomposition. We want to decompose positive definite matrix A into lower triangular matrix L such that $L^\top L = A$. But A does not fit entirely in GPU memory, and potentially more than one GPU is available. As before it is convenient to subdivide A into smaller tiles such that the tiles fit in GPU memory.

$$\begin{pmatrix} A_{1,1} & & \\ A_{2,1} & A_{2,2} & \\ \vdots & \ddots & \\ A_{n,1} & \dots & A_{n,n} \end{pmatrix} = \begin{pmatrix} L_{1,1} & & \\ L_{2,1} & L_{2,2} & \\ \vdots & \ddots & \\ L_{n,1} & \dots & L_{n,n} \end{pmatrix} \begin{pmatrix} L_{1,1}^T & L_{2,1}^T & \dots & L_{n,1}^T \\ & L_{2,2}^T & \dots & L_{n,2}^T \\ & & \ddots & \vdots \\ & & & L_{n,n}^T \end{pmatrix}$$

Then the in-place decomposition may proceed column-wise across matrix A, where each column update requires three steps. The first step is to use the in-core `POTRF` function from cuSOLVER [33] on a single tile. Then, a triangular solution step is used to update the remaining rows of the first column (taking the first column as an example $A_{j,1} = L_{j,1}L_{1,1}^\top, 1 < j < N$, so clearly $L_{j,1} = A_{j,1}(L_{1,1}^{-1})^\top$). This can be done by using the `TRSM` operation from any GPU BLAS implementation. Finally, the *trailing* submatrix must be updated with those terms which can be computed from the current column, so that after this last step such column is not needed anymore. This step consists of running $A_{ij} = A_{ij} - L_{i,1}L_{j,1}^\top$ where if c is the current column $i > c, \quad c < j \leq i$ (refer to Figure 5 for a more intuitive picture).

Running this algorithm in parallel requires dealing with several data dependencies in-between tiles, and in general it will not be possible to achieve perfect parallelism due to the inherently serial step of performing the Cholesky decomposition of the first tile in a column. We avoid coarse synchronization mechanisms such as the thread barrier which was used for the LAUUM OOC implementation, since we found they could introduce very high waiting times (barriers would be needed after each of the three steps of the algorithm to ensure proper synchronization). Our solution, which somewhat follows [27], uses an integer table T with one entry per tile, which is shared between all GPU threads. The entries of T represent the current state of each tile: basically how many times the tile has been updated. Since we use

a static row-cyclic work allocation like for the triangular matrix multiplication, each thread knows the expected state of a tile for each step (e.g. to perform the first step on tile $A_{c,c}$ the tile must have been updated exactly c times). So it can wait until such state has been reached, then read the tile into GPU memory, perform the update, write back the tile to main RAM, and increment the corresponding entry in T. Such a scheme is implemented in Algorithm 4 with the help of the `Load` and `Write` sub-routines. Further optimizations are possible by being careful about which tiles are swapped in and out of GPU memory and at what times, overlapping computation with memory transfers when possible. Such optimizations generally require to increase the total memory allocated on the GPU, thus decreasing the maximum possible tile-size.

Algorithm 2 Pseudocode for appr. Newton method with Falkon, for GSC losses (based on [30]).

function GSC-FALKON($X \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n, \lambda, m, t, T$)
  Set $\alpha_0 = 0 \in \mathbb{R}^m$ and $\mu_0 > 0, q > 0$ according to [30].
  $X_m, \boldsymbol{y}_m \leftarrow$ RANDOMSUBSAMPLE($X, \boldsymbol{y}, m$)
  for $k \in \mathbb{N}$ **do**
    $f_{k+1} \leftarrow$ WEIGHTEDFALKON($X, \boldsymbol{y}, X_m, \boldsymbol{y}_m\mu_k, t, \alpha_k$)
    $\mu_{k+1} \leftarrow q\mu_k$
    Stop when $\mu_{k+1} < \lambda$ and set $\alpha_{last} \leftarrow \alpha_k$.
  end forreturn $\widehat{\alpha} \leftarrow$ WEIGHTEDFALKON($X, \boldsymbol{y}, X_m, \boldsymbol{y}_m, \lambda, T, \alpha_k$)
end function

function WEIGHTEDFALKON($X \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n, X_m \in \mathbb{R}^{m\times d}, \boldsymbol{y}_m \in \mathbb{R}^m, \lambda, t, \alpha_0 \in \mathbb{R}^m$)
  $T, A \leftarrow$ WEIGHTEDPRECONDITIONER($X_m, \boldsymbol{y}_m, \alpha_0, \lambda$)
  function LINOP($\boldsymbol{\beta} \in \mathbb{R}^m$)
    $\boldsymbol{v} \leftarrow A^{-1}\boldsymbol{\beta}$
    $z \leftarrow k(X, X_m)\boldsymbol{\beta}$ ▷ predictions on the dataset
    $D \leftarrow \mathtt{diag}[(\ell^{(2)}((z)_1, (\boldsymbol{y})_1), \dots, \ell^{(2)}((z)_n, (\boldsymbol{y})_n))]$
    $\boldsymbol{c} \leftarrow k(X_m, X)Dk(X, X_m)T^{-1}\boldsymbol{v}$
    return $A^{-\top}T^{-\top}\boldsymbol{c} + \lambda n\boldsymbol{v}$
  end function
  $R \leftarrow A^{-\top}T^{-\top}k(X, X_m)\boldsymbol{y}$
  $\boldsymbol{\beta} \leftarrow$ CONJUGATEGRADIENT(LINOP, R, t, α_0) ▷ CG solver starting from α_0
  return $T^{-1}A^{-1}\boldsymbol{\beta}$
end function

function WEIGHTEDPRECONDITIONER($X_m \in \mathbb{R}^{m\times d}, \boldsymbol{y}_m \in \mathbb{R}^m, \alpha \in \mathbb{R}^m, \lambda$)
  $K_{mm} \leftarrow k(X_m, X_m)$ ▷ Compute the kernel between inducing points
  $z \leftarrow K_{mm}\alpha$ ▷ predictions on the Nyström points
  $T \leftarrow \mathrm{chol}(K_{mm})$
  $D \leftarrow \mathtt{diag}[(\ell^{(2)}((z)_1, (\boldsymbol{y}_m)_1), \dots, \ell^{(2)}((z)_m, (\boldsymbol{y}_m)_m))]$
  $K_{mm} \leftarrow {}^1\!/_m TDT^\top + \lambda\boldsymbol{I}$
  $A \leftarrow \mathrm{chol}(K_{mm})$
  return T, A
end function

Note: LINOP performs the multiplications via matrix-vector products.

Algorithm 3 Out-of-core LAUUM operation on an upper-triangular matrix. The algorithm's inputs are matrix U, a synchronization object `barrier`, an array of arrays describing which row indices are assigned to which processor `blockAllocs`, and the number of tiles per side N. The function described below should be called for every available GPU with a different `procId` value.

function OocLauum($U \in \mathbb{R}^{n \times n}$, `barrier`, `blockAllocs`, `procId`, N)
  for $i = 1, \ldots, N$ **do**
    $C \in \mathbb{R}^{t \times t \cdot (N-i)} \leftarrow$ `ToGPU`(`procId`, $\left[U_{i,i}, \ldots, U_{i,N}\right]$)
    `barrier.wait()`
    for $j \in$ `blockAllocs`[`procId`] **do**
      if $i = j$ **then**
        $C_1 \leftarrow C_1 C_1^\top$ ▷ via `LAUUM`
        if $i \neq N$ **then**
          $C_1 \leftarrow C_1 + C_{1:(N-i+1)} C_{1:(N-i+1)}^\top$ ▷ via `SYRK`
        end if
      else if $j > i$ **then**
        $D \in \mathbb{R}^{t \times t \cdot (N-j)} \leftarrow$ `ToGPU`(`procId`, $\left[U_{j,j}, \ldots, U_{j,N}\right]$)
        $C_{(j-i)} \leftarrow C_{(j-i)} D_1^\top$ ▷ via `TRMM`
        if $j \neq N$ **then**
          $C_{(j-i)} \leftarrow C_{(j-i+1):(N-i+1)} D_{2:(N-j+1)}^\top$ ▷ via `GEMM`
        end if
      end if
      $U_{i,j} \leftarrow$ `FromGPU`(`procId`, $C_{(j-i)}$)
    end for
  end forreturn U
end function

Algorithm 4 Out-of-core, in-place Cholesky decomposition of symmetric positive definite matrix A. The lower triangle of A will be overwritten by L such that $L^\top L = A$. The function `OocPotrf` should be called for each available GPU with different values of the `procId` variable to parallelize the decomposition across GPUs. The inputs are the same as for Algorithm 3 but for work-table $T \in \mathbb{Z}^{N \times N}$ whose values are atomically updated by the different GPU processes to ensure synchronization.

```
1: function OocPotrf(A, blockAllocs, procId, T, N)
2:     for i = 1, ..., N do
3:         if i ∈ blockAllocs[procId] then
4:             B ← Load(A, T, i, j, i)
5:             B ← POTRF(B)
6:             A_{i,i} ← Write(B, T, i, i)
7:         end if
8:         for j ∈ blockAllocs[procId] do
9:             if j ≤ i then
10:                continue
11:            end if
12:            B ← Load(A, T, i, i, i + 1)
13:            C ← Load(A, T, j, i, i)
14:            C ← C(B^{-1})^⊤                ▷ via TRSM
15:            A_{j,i} ← Write(C, T, j, i)
16:        end for
17:        for j ∈ blockAllocs[procId] do
18:            if j ≤ i + 1 then
19:                continue
20:            end if
21:            C ← Load(A, T, j, i, i + 1)
22:            for y = i, ... j do
23:                E ← Load(A, T, j, y, i)
24:                if y = j then
25:                    E ← E − CC^⊤           ▷ via SYRK
26:                else
27:                    D ← Load(A, T, y, i, i + 1)
28:                    E ← E − DC^⊤           ▷ via GEMM
29:                end if
30:                A_{j,y} ← Write(E, T, j, y)
31:            end for
32:        end for
33:    end for
34: end function

35: function Load(A, T, i, j, exp)
36:     while T_{i,j} < exp do
37:         wait
38:     end while
39:     return ToGPU(A_{i,j})
40: end function

41: function Write(G, T, i, j)
42:     T_{i,j} ← T_{i,j} + 1
43:     return FromGPU(G)
44: end function
```

Efficient Hyperparameter Tuning for Large Scale Kernel Ridge Regression

Giacomo Meanti[1], Luigi Carratino[1], Ernesto De Vito[2], and Lorenzo Rosasco[1,3,4]

[1]*MaLGa, DIBRIS, Università degli Studi di Genova*
[2]*MaLGa, DIMA, Università degli Studi di Genova*
[3]*CBMM, Massachusetts Institute of Technology*
[4]*Istituto Italiano di Tecnologia*

January 19, 2022

Abstract

Kernel methods provide a principled approach to nonparametric learning. While their basic implementations scale poorly to large problems, recent advances showed that approximate solvers can efficiently handle massive datasets. A shortcoming of these solutions is that hyperparameter tuning is not taken care of, and left for the user to perform. Hyperparameters are crucial in practice and the lack of automated tuning greatly hinders efficiency and usability. In this paper, we work to fill in this gap focusing on kernel ridge regression based on the Nyström approximation. After reviewing and contrasting a number of hyperparameter tuning strategies, we propose a complexity regularization criterion based on a data dependent penalty, and discuss its efficient optimization. Then, we proceed to a careful and extensive empirical evaluation highlighting strengths and weaknesses of the different tuning strategies. Our analysis shows the benefit of the proposed approach, that we hence incorporate in a library for large scale kernel methods to derive adaptively tuned solutions.

1 Introduction

Learning from finite data requires fitting models of varying complexity to training data. The problem of finding the model with the right complexity is referred to as model selection in statistics and more broadly as hyperparameter tuning in machine learning. The problem is classical and known to be of utmost importance for machine learning algorithms to perform well in practice. The literature in statistics is extensive (Hastie et al., 2009), including a number of theoretical results (Arlot, 2007; Massart, 2007; Tsybakov, 2003). Hyperparameter (HP) tuning is also at the core of recent trends such as neural architecture search (Elsken et al., 2019) or AutoML (Hutter et al., 2018). In this paper, we consider the question of hyperparameter tuning in the context of kernel methods and specifically kernel ridge regression (KRR) (Smola and Schölkopf, 2000). Recent advances showed that kernel methods can be scaled to massive data-sets using approximate solvers (Chen et al., 2017; Ma and Belkin, 2019; Meanti et al., 2020). The latter take advantage of a number of ideas from optimization (Boyd and Vandenberghe, 2004) and randomized algorithms (El Alaoui and Mahoney, 2015), and exploit parallel computations with GPUs. While these solutions open up new possibilities for applying kernel methods, hyperparameter tuning is notably missing, ultimately hindering their practical use and efficiency. Indeed, available solutions which provide hyperparameter tuning are either limited to small data, or are restricted to very few hyperparameters (Pedregosa et al., 2011; Steinwart and Thomann, 2017; Suykens et al., 2002).

In this paper we work to fill in this gap. We consider approximate solvers based on the Nyström approximation and work towards an automated tuning of the regularization and kernel parameters, as well as the Nyström centers. On the one hand, we provide a careful review and extensive empirical comparison for a number of hyperparameter tuning strategies, while discussing their basic theoretical guarantees. On the other hand we propose, and provide an efficient implementation for, a novel criterion inspired by complexity regularization (Bartlett et al., 2002) and based on a data-dependent bound. This bound treats separately the sources of variance due to the stochastic nature of the data. In practice, this results in better stability properties of the corresponding tuning strategy. As a byproduct of our analysis we complement an existing library for large-scale kernel methods with the possibility to adaptively tune a large number of hyperparameters. Code is available at the following address: `https://github.com/falkonml/falkon`.

In Section 2 we introduce the basic ideas behind empirical risk minimization and KRR, as well as hyperparameter tuning. In Section 3 we propose our new criterion, and discuss its efficient implementation in Section 4. In Section 5 we conduct a thorough experimental study and finally, in Section 6 we provide some concluding remarks.

2 Background

We introduce the problem of learning a model's parameters, which leads to learning of the hyperparameters and then discuss various objective functions and optimization algorithms which have been proposed for the task.

2.1 Parameter and Hyperparameter Learning

Assume we are given a set of measurements $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$ related to each other by an unknown function $f^* : \mathcal{X} \to \mathcal{Y}$ and corrupted by some random noise ϵ_i with variance σ^2.

$$y_i = f^*(x_i) + \epsilon_i. \tag{1}$$

We wish to approximate the target function f^* using a model $f : \mathcal{X} \to \mathcal{Y}$ defined by a set of *parameters* which must be learned from the limited measurements at our disposal. In order for the learning procedure to succeed, one often assumes that f belongs to some hypothesis space $\mathcal{F}$, and this space typically depends on additional *hyperparameters* θ. Assume we are given a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$; we can learn a model by fixing the hyperparameters θ and minimizing the loss over the available training samples:

$$\hat{f}_\theta = \operatorname*{arg\,min}_{f \in \mathcal{F}_\theta} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

In this paper we are concerned with kernel ridge regression: a specific kind of model where the loss function is the squared loss $\ell(y, y') = \|y - y'\|^2$ and the hypothesis space is a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$. Associated to $\mathcal{H}$ is a kernel function $k_\gamma : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which depends on hyperparameters γ. To ensure that the minimization problem is well defined we must add a regularization term controlled by another hyperparameter λ:

$$\hat{f}_{\lambda,\gamma} = \operatorname*{arg\,min}_{f \in \mathcal{H}} \sum_{i=1}^{n} \|f(x_i) - y_i\|^2 + \lambda \|f\|_{\mathcal{H}}^2.$$

The solution to this minimization problem is unique (Caponnetto and De Vito, 2007), but is very expensive to compute requiring $O(n^3)$ operations and $O(n^2)$ space. An approximation to KRR considers a lower-dimensional subspace $\mathcal{H}_m \subset \mathcal{H}$ as hypothesis space, where $\mathcal{H}_m$

is defined from $m \ll n$ points $Z = \{z_j\}_{j=1}^m \subset \mathcal{X}$ (Williams and M. Seeger, 2001). While the inducing points Z (also known as Nyström centers) are often picked from the training set with different sampling schemes (Kumar et al., 2012), they can also be considered as hyperparameters. In fact this is common in sparse Gaussian Processes (GPs) and leads to models with a much smaller number of inducing points (Hensman, Fusi, et al., 2013; Hensman, Matthews, et al., 2015; Titsias, 2009). Minimizing the regularized error gives the unique solution

$$\hat{f}_{\lambda,Z,\gamma} = \sum_{j=1}^{m} \beta_j k_\gamma(\cdot, z_j), \quad \text{with} \quad \beta = (K_{nm}^\top K_{nm} + \lambda n K_{mm})^{-1} K_{nm}^\top Y \tag{2}$$

with $(K_{nm})_{i,j} = k_\gamma(x_i, z_j)$ and $(K_{mm})_{i,j} = k_\gamma(z_i, z_j)$. The Nyström KRR model (N-KRR) reduces the computational cost of finding the coefficients to $O(n\sqrt{n}\log n)$ when using efficient solvers (Ma and Belkin, 2019; Meanti et al., 2020; Rudi et al., 2017).

The ideal goal of hyperparameter optimization is to find a set of hyperparameters θ^* for which $\hat{f}_{\theta^*}$ minimizes the test error (over all unseen samples). By definition we cannot actually evaluate the test error: we must use the available data points. Naïvely one could think of minimizing the training error instead, but such a scheme inevitably chooses overly complex models which overfit the training set. Instead it is necessary to minimize a data-dependent criterion $\mathcal{L}$

$$\widehat{\theta} = \arg\min_\theta \mathcal{L}(\hat{f}_\theta)$$

such that complex models are penalized. In practice a common strategy for choosing $\mathcal{L}$ is for its expectation (with respect to the sampling of the data) to be equal to, or an upper bound of the test error. In the next section we will look at instances of $\mathcal{L}$ which appear in the literature and can be readily applied to N-KRR.

2.2 Objective Functions

Validation error A common procedure for HP tuning is to split the available n training samples into two parts: a training set and a validation set. The first is used to learn a model $\hat{f}_\theta$ with fixed hyperparameters θ, while the validation set is used to estimate the performance of different HP configurations.

$$\mathcal{L}^{\text{Val}}(\hat{f}_\theta) = \frac{1}{n_{\text{val}}} \sum_{i=1}^{n_{\text{val}}} \|\hat{f}_\theta(x_i^{\text{val}}) - y_i^{\text{val}}\|^2 \tag{3}$$

By using independent datasets for model training and HP selection, $\mathcal{L}^{\text{Val}}$ will be an unbiased estimator of the test error and it can be proven that its minimizer is close to θ^* under certain assumptions (Arlot and Celisse, 2010). However, since $\hat{f}_\theta$ has been trained with $n_{\text{tr}} < n$ samples, there is a small bias in the chosen hyperparameters (Varma and Simon, 2006). Furthermore the variance of the hold-out estimator is typically very high as it depends on a specific data split. Two popular alternatives which address this latter point are k-fold cross-validation (CV) which averages over k hold-out estimates and leave-one-out CV.

Leave-one-out CV and Generalized CV The LOOCV estimator is an average of the n estimators trained on all $n-1$ sized subsets of the training set and evaluated on the left out sample. The result is an almost unbiased estimate of the expected risk on the full dataset (Vapnik, 1998). For linear models a computational shortcut allows to compute the LOOCV estimator by training a single model on the whole dataset instead of n different ones (Cawley and Talbot, 2004). In particular in the case of N-KRR we can consider

$$\mathcal{L}^{\text{LOOCV}}(\hat{f}_\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_i - \hat{f}_\theta(x_i)}{1 - H_{ii}} \right)^2, \tag{4}$$

where the so-called hat matrix H is $H = K_{nm}(K_{nm}^\top K_{nm} + \lambda n K_{mm})^{-1} K_{nm}$.

GCV is an approach proposed in Golub et al. (1979) to further improve LOOCV's computational efficiency and to make it invariant to data rotations:

$$\mathcal{L}^{\text{GCV}}(\hat{f}_\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{f}_\theta(x_i)}{\frac{1}{n}\text{Tr}(I-H)}\right)^2. \tag{5}$$

For GCV Cao and Golubev (2006) proved an oracle inequality which guarantees convergence to the neighborhood of θ^* when estimating λ for KRR.

Complexity regularization Complexity regularization, or covariance penalties (Efron, 2004; Mallows, 1973) are a general framework for expressing objective functions as the empirical error plus a penalty term to avoid overly complex models. For linear models the trace of the hat matrix acts as penalty against complexity. Applying these principles to N-KRR gives the objective

$$\mathcal{L}^{\text{C}-\text{Reg}}(\hat{f}_{\lambda,Z,\gamma}) = \frac{1}{n}\|\hat{f}_{\lambda,Z,\gamma}(X) - Y\|^2 + \frac{2\sigma^2}{n}\text{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big) \tag{6}$$

where $\widetilde{K} = K_{nm}K_{mm}^\dagger K_{nm}^\top$ (the Nyström kernel), and $A^\dagger$ denotes the Moore-Penrose inverse of matrix A. The first term can be interpreted as a proxy to the bias of the error, and the second as a variance estimate. For estimating λ in (N-)KRR, Arlot and Bach (2009) proved an oracle inequality if a precise estimate of the noise σ^2 is available.

Sparse GP Regression (Titsias, 2009) A different approach comes from a Bayesian perspective, where the equivalent of KRR is Gaussian Process Regression (GPR). Instead of estimating the test error, HP configurations are scored based on the "probability of a model given the data" (Rasmussen and Williams, 2006). A fully Bayesian treatment of the hyperparameters allows to write down their posterior distribution, from which the HP likelihood has the same form of the marginal likelihood in the model parameter's posterior. Hence maximizing the (log) marginal likelihood (MLL) with gradient-based methods is common practice in GPR.

Like with N-KRR, inducing points are used in GPR to reduce the computational cost, giving rise to models such as SoR, DTC, FiTC (Quiñonero-Candela and Rasmussen, 2005). Here we consider the SGPR model proposed in Titsias (2009) which treats the inducing points as variational parameters, and optimizes them along with the other HPs by maximizing a lower bound to the MLL. The objective to be minimized is

$$\mathcal{L}^{\text{SGPR}}(\hat{f}_{\lambda,Z,\gamma}) = \log\Big|\widetilde{K} + n\lambda\boldsymbol{I}\Big| + Y^\top(\widetilde{K} + n\lambda\boldsymbol{I})^{-1}Y + \frac{1}{n\lambda}\text{Tr}\Big(K - \widetilde{K}\Big). \tag{7}$$

The first term of Eq. (7) penalizes complex models, the second pushes towards fitting the training set well and the last term measures how well the inducing points approximate the full training set. Recently the approximate MLL was shown to converge to its exact counterpart (Burt et al., 2020), but we note that this does not guarantee convergence to the optimal hyperparameters.

2.3 Optimization Algorithms

In this section we describe three general approaches for the optimization of the objectives introduced above.

Grid search In settings with few hyperparameters the most widely used optimization algorithm is grid-search which tries all possible combinations from a predefined set, choosing the one with the lowest objective value at the end. Random search (Bergstra and Bengio, 2012) and adaptive grid search (used for SVMs in Steinwart and Thomann (2017)) improve on this basic idea, but they also become prohibitively costly with more than ~ 5 HPs as the number of combinations to be tested grows exponentially.

Black-box optimization A more sophisticated way to approach the problem is to take advantage of any smoothness in the objective. Sequential model-based optimization (SMBO) algorithms (Brochu et al., 2010; Shahriari et al., 2016; Snoek et al., 2012) take evaluations of the objective function as input, and fit a Bayesian *surrogate* model to such values. The surrogate can then be cheaply evaluated on the whole HP space to suggest the most promising HP values to explore. These algorithms do not rely on gradient information so they don't require the objective to be differentiable and can be applied for optimization of discrete HPs. However, while more scalable than grid search, black-box algorithms become very inefficient in high (i.e. > 100) dimensions.

Gradient-based methods Scaling up to even larger hyperparameter spaces requires exploiting the objective's local curvature. While the optimization problem is typically non-convex, gradient descent will usually reach a good local minimum. When the objective can be decomposed as a sum over the data-points SGD can be used, which may provide computational benefits (e.g. the SVGP objective (Hensman, Fusi, et al., 2013) is optimized in mini-batches with SGD). In the context of KRR, gradient-based methods have been successfully used for HP optimization with different objective functions (Keerthi et al., 2007; M. W. Seeger, 2008). Recent extensions to gradient-based methods have been proposed for those cases when the trained model cannot be written in closed form. Either by unrolling the iterative optimization algorithm (Franceschi et al., 2017; Grazzi et al., 2020; Maclaurin et al., 2015), or by taking the model at convergence with the help of the implicit function theorem (Pedregosa, 2016; Rajeswaran et al., 2019), it is then possible to differentiate a simple objective (typically a hold-out error) through the implicitly defined trained model. This has proven to be especially useful for deep neural nets (Lorraine et al., 2020), but is unnecessary for N-KRR where the trained model can be easily written in closed form.

3 Hyper-parameter Optimization for Nyström KRR

The objectives introduced in the previous section can be applied to HP tuning for kernel methods. Always keeping in mind efficiency but also usability, our goal is to come up with an objective and associated optimization algorithm which: 1) can be used to tune the hyperparameters of Nyström kernel ridge regression including the inducing points and 2) can be computed efficiently, even for large scale problems.

To satisfy the first point, an algorithm of the first-order is needed since the inducing points are typically between a hundred and a few thousands (each point being of the same dimension as the data). Regarding the second point we found empirically that the unbiased objectives are prone to overfitting on certain datasets. An example of this behavior is shown in Figure 1 on a small subset of the HIGGS dataset. The first three objectives (Hold-out, GCV and C-Reg) are unbiased estimates of the test error, hence it is their variance which causes overfitting. To mitigate such possibility in our objective we may look into the different sources of variance: *hold-out* depends strongly on which part of the training set is picked for validation, *GCV* and *C-Reg* don't rely on data splitting but still suffer from the variance due to the random initial choice of inducing points.

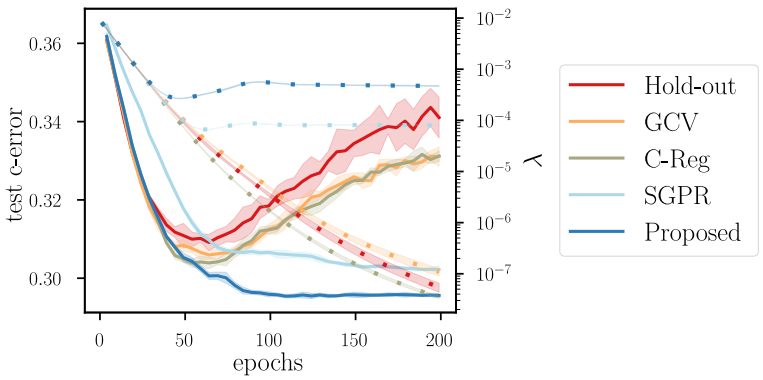


Figure 1: Test-error and penalty (λ) as a function of optimization epoch on the small-HIGGS dataset. $m = 100$ centers, d lengthscales and λ were optimized with equal initial conditions. The three unbiased proxy functions lead to overfitting, while SGPR and the proposed objective do not.

We set out to devise a new objective in the spirit of complexity regularization, which is an upper bound on the test error. A biased estimate – which is therefore overpenalizing – will be more resistant to noise than an unbiased one (as was noted in Arlot (2007)), and we tailor our objective specifically to N-KRR in order to explicitly take into account the variance from inducing point selection.

We base our analysis of the N-KRR error in the fixed design setting, where the points $x_i \in \mathcal{X}, i = (1, \dots, n)$ are assumed to be fixed, and the stochasticity comes from i.i.d. random variables $\epsilon_i, \dots, \epsilon_n$ such that $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{E}\left[\epsilon_i^\top \epsilon_i\right] = \sigma^2$. Denote the empirical error of an estimator $f \in \mathcal{H}$ as $\hat{L}(f) = n^{-1}\|f(X) - Y\|^2$ and the test error as $L(f) = n^{-1}\|f(X) - f^*(X)\|^2$ (recall f^* from Eq. (1)). Consider inducing points z_j and a subspace of $\mathcal{H}$: $\mathcal{H}_m = \text{span}\{k_\gamma(z_1, \cdot), \dots, k_\gamma(z_m, \cdot)\}$, $m \ll n$, and let P be the projection operator with range $\mathcal{H}_m$. Denote the regularized empirical risk as $\hat{L}_\lambda(f) = \hat{L}(f) + \lambda\|f\|^2_{\mathcal{H}}$,

Assessing a particular hyperparameter configuration (λ, Z, γ) requires estimating the expected test error at the empirical risk minimizer trained with that configuration $\hat{f}_{\lambda,Z,\gamma}$; the optimal HPs then are found by $(\lambda, Z, \gamma)^* = \arg\min_{(\lambda,Z,\gamma)} L(\hat{f}_{\lambda,Z,\gamma})$. The following lemma gives an upper bound on the ideal objective; a full proof is available in Appendix A.

Lemma 1. *Under the assumptions of fixed-design regression we have that,*

$$\begin{aligned}\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq & \frac{2\sigma^2}{n}\text{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big) \\ & + \frac{2}{n\lambda}\text{Tr}\Big(K - \widetilde{K}\Big)\mathbb{E}\Big[\hat{L}(f_{\lambda,\gamma})\Big] \\ & + 2\mathbb{E}\Big[\hat{L}(f_{\lambda,\gamma})\Big] \end{aligned} \tag{8}$$

Proof sketch. We decompose the test error expectation in the following manner

$$\begin{aligned}\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq \mathbb{E}\Big[& \underbrace{L(\hat{f}_{\lambda,Z,\gamma}) - \hat{L}(\hat{f}_{\lambda,Z,\gamma})}_{①} \\ & + \underbrace{\hat{L}(\hat{f}_{\lambda,Z,\gamma}) + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}} - \hat{L}_\lambda(Pf_{\lambda,\gamma})}_{②} + \underbrace{\hat{L}_\lambda(Pf_{\lambda,\gamma})}_{③}\Big]\end{aligned}$$

by adding and subtracting $\hat{L}(\hat{f}_{\lambda,Z,\gamma})$, $\hat{L}_\lambda(Pf_{\lambda,\gamma})$ and summing the positive quantity $\lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}}$. Since $\hat{f}_{\lambda,Z,\gamma}$ is the minimizer of $\hat{L}(\hat{f}_{\lambda,Z,\gamma}) + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}}$ in the space $\mathcal{H}_m$ and since $Pf_{\lambda,\gamma} \in \mathcal{H}_m$, the second term is negative and can be discarded.

Term ① is the variance of N-KRR and can be computed exactly by noting that

$$\begin{aligned}\mathbb{E}\Big[\hat{L}(\hat{f}_{\lambda,Z,\gamma})\Big] &= \mathbb{E}\Big[n^{-1}\|\hat{f}_{\lambda,Z,\gamma}(X) - f^*(X) - \epsilon\|^2\Big] \\ &= \mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] + \sigma^2 - \frac{2}{n}\mathbb{E}\Big[\langle \hat{f}_{\lambda,Z,\gamma}(X) - f^*(X), \epsilon\rangle\Big]\end{aligned}$$

where the first part cancels and we can ignore σ^2 which is fixed and positive. Expanding the inner product and taking its expectation we are left with

$$\frac{2}{n}\mathbb{E}\Big[\langle \hat{f}_{\lambda,Z,\gamma}(X) - f^*(X), \epsilon\rangle\Big] = \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big)$$

which is the *effective dimension* or the *degrees of freedom* of the hypothesis space $\mathcal{H}_m$, times the noise variance σ^2.

Term ③ takes into account the difference between estimators in $\mathcal{H}$ and in $\mathcal{H}_m$. We begin by upper-bounding the regularized empirical error of $Pf_{\lambda,\gamma}$ with a first part containing the projection operator and a second term without P

$$\mathbb{E}\Big[\hat{L}(Pf_{\lambda,\gamma}) + \lambda\|Pf_{\lambda,\gamma}\|^2_{\mathcal{H}}\Big] \leq \mathbb{E}\left[\frac{2}{n}\|K^{1/2}(I-P)\|^2\|f_{\lambda,\gamma}\|^2 + 2\hat{L}_\lambda(f_{\lambda,\gamma})\right].$$

Now $\|K^{1/2}(I-P)\|^2 \leq \mathrm{Tr}(K - \widetilde{K})$ the difference between full and approximate kernels, and $\|f_{\lambda,\gamma}\|^2 \leq \lambda^{-1}\hat{L}_\lambda(f_{\lambda,\gamma})$ which leads us to the desired upper bound. □

We now make two remarks on computing Eq. (8).

Remark 1. *(Computing $\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big]$)* In the spirit of complexity regularization we can approximate this bias term by the empirical risk of N-KRR $\hat{L}_\lambda(\hat{f}_{\lambda,Z,\gamma})$, so that the final objective will consist of a data-fit term plus two complexity terms: the effective dimension and the Nyström approximation error.

Remark 2. *(Estimating σ^2)* Once again following the principle of overpenalizing rather than risking to overfit, we note that in binary classification the variance of Y is capped at 1 for numerical reasons, while for regression we can preprocess the data dividing Y by its standard deviation. Then according to Eq. (1) we must have that the label standard deviation is greater than the noise standard deviation hence $\hat{\sigma}^2 = 1 \geq \sigma^2$.

Our final objective then has a form which we can compute efficiently

$$\begin{aligned}\mathcal{L}^{\mathrm{Prop}} =& \frac{2}{n}\mathrm{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big) \\ &+ \frac{2}{n\lambda}\mathrm{Tr}\Big(K - \widetilde{K}\Big)\hat{L}_\lambda(\hat{f}_{\lambda,Z,\gamma}) \\ &+ \frac{2}{n}\|\hat{f}_{\lambda,Z,\gamma}(X) - Y\|^2 + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}}. \end{aligned} \tag{9}$$

We make two further remarks on the connections to the objectives of Section 2.2.

Remark 3. *(Similarities with complexity regularization)* $\mathcal{L}^{\mathrm{Prop}}$ has a similar form to Eq. (6) with an extra term which corresponds to the variance introduced by the Nyström centers which we were aiming for (up to multiplication by the KRR bias).

Remark 4. *(Similarities with SGPR)* Eq. (9) shares many similarities with the SGPR objective: the log-determinant is replaced by the model's effective dimension – another measure of model complexity – and the term $\mathrm{Tr}(K - \widetilde{K})$ is present in both objectives. Furthermore the data-fit term in $\mathcal{L}^{\mathrm{SGPR}}$ is

$$\begin{aligned}Y^\top(\widetilde{K} + n\lambda I)^{-1}Y &= \frac{1}{\lambda}(n^{-1}\|\hat{f}_{\lambda,Z,\gamma}(X) - Y\|^2 + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}}) \\ &= \frac{1}{\lambda}\hat{L}_\lambda(\hat{f}_{\lambda,Z,\gamma})\end{aligned}$$

which is the same as in the proposed objective up to a factor λ^{-1}.

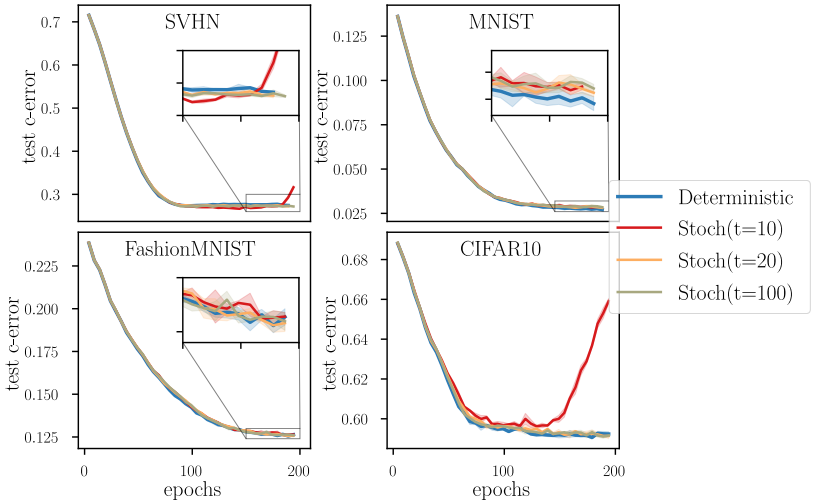


Figure 2: The effect of stochastic trace estimation. We plot the optimization curves of the exact objective $\mathcal{L}^{\text{Prop}}$ (*Deterministic*) and the approximated objectives with 10, 20 and 100 STE vectors. On the four datasets we optimized $m = 200$ centers, λ and γ.

4 Scalable Approximations

Some practical considerations are needed to apply the objective of Eq. (9) to large-scale datasets – for which direct computation is not possible due to space or time constraints. We examine the terms comprising $\mathcal{L}^{\text{Prop}}$ and discuss their efficient computations. In Figure 2, we verify that the resulting approximation is close to the exact objective.

Starting with the last part of the optimization objective (the one which measures data-fit) we have that

$$\|\hat{f}_{\lambda,Z,\gamma}(X) - Y\|^2 + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}} = Y^\top (I - \underbrace{K_{nm}(\overbrace{K_{nm}^\top K_{nm} + n\lambda K_{mm}}^{B})^{-1} K_{nm}^\top}_{=\hat{f}_{\lambda,Z,\gamma}(X)})Y$$

which can be computed quickly using a fast, memory-efficient N-KRR solver such as Falkon (Meanti et al., 2020) or EigenPro (Ma and Belkin, 2019). However we must also compute the objective's gradients with respect to all HPs, and since efficient solvers proceed by iterative minimization, such gradients cannot be trivially computed using automatic differentiation, indeed, it would be in principle possible to unroll the optimization loops and differentiate through them, the memory requirements for this operation would be too high for large datasets.

Efficient gradients A solution to compute the gradients efficiently is to apply the chain rule by hand until they can be expressed in terms of matrix vector products $(\nabla K)\boldsymbol{v}$ with K any kernel matrix (i.e. K_{nm} or K_{mm}) and $\boldsymbol{v}$ a vector. As an example the gradient of the data-fit term is

$$\nabla(Y^\top K_{nm} B^{-1} K_{nm}^\top Y) = 2Y^\top (\nabla K_{nm}) B^{-1} K_{nm}^\top Y - Y^\top K_{nm} B^{-1} (\nabla B) B^{-1} K_{nm}^\top Y$$

where we can obtain all $B^{-1}K_{nm}^\top Y$ vectors via a non-differentiable N-KRR solver, and multiply them by the (differentiable) kernel matrices for which gradients are required. Computing these elementary operations is efficient, with essentially the same cost as the forward pass $K\boldsymbol{v}$, and can be done row-wise over K. Block-wise computations are essential for low memory usage since kernel matrices tend to be huge but kernel-vector products are small, and

they allow trivial parallelization across compute units (CPU cores or GPUs). In many cases these operations can also be accelerated using KeOps (Charlier et al., 2021).

The remaining two terms of Eq. (9) are harder to compute. Note that in $\mathrm{Tr}(K - \widetilde{K})$ we can often ignore $\mathrm{Tr}(K)$ since common kernel functions are trivial when computed between a point and itself, but more in general it only requires evaluating the kernel function n times. We thus focus on

$$\mathrm{Tr}\Big(\widetilde{K}\Big) = \mathrm{Tr}\Big(K_{nm}K_{mm}^{\dagger}K_{nm}^{\top}\Big) \tag{10}$$

and on the effective dimension

$$\mathrm{Tr}\Big((\widetilde{K} + \lambda I)^{-1}\widetilde{K}\Big) = \mathrm{Tr}\Big(K_{nm}B^{-1}K_{nm}^{\top}\Big). \tag{11}$$

Both these terms are traces of huge $n \times n$ matrices. By their symmetry we can express them as squared norms reducing the space requirements to $n \times m$, but they still remain slow to compute: just the $K_{nm}^{\top}K_{nm}$ term costs more than training a N-KRR model with the Falkon solver.

Trace estimation A simple approximation can vastly improve the efficiency of computing Equations (10), (11), and their gradients: stochastic trace estimation (STE). The Hutchinson estimator (Hutchinson, 1990) approximates $\mathrm{Tr}(A)$ by $\frac{1}{t}\sum_{i=1}^{t} r_i^{\top}Ar_i$ where r_i are zero mean, unit standard deviation random vectors. We can use this to estimate Eq. (11) by running the Falkon solver with $R = [r_1, \dots, r_t]$ instead of the labels Y to obtain $(K_{nm}^{\top}K_{nm} + \lambda K_{mm})^{-1}K_{nm}^{\top}R$, then multiplying the result by $K_{nm}^{\top}R$ and normalizing by the number of stochastic estimators t. The same random vectors R can be used to compute $K_{nm}^{\top}R$ for Eq. (10), coupled with the Cholesky decomposition of K_{mm}. STE reduces the cost for both terms from $O(nm^2)$ to $O(nmt)$ which is advantageous since $t < m$. In Figure 4 we investigate whether the approximate objective matches the exact one, and how t affects the approximation. The observed behavior is that as few as 10 vectors are enough to approximate the full objective for a large part of the optimization run, but it can happen that such coarse approximation causes the loss to diverge. Increasing t to 20 solves the numerical issues, and on all the datasets tested we found $t = 20$ to be sufficient.

Alternatively, Eq. (10) can be approximated with a Nyström-like procedure: taking a random subsample of size p from the whole dataset, denote K_{pm} as the kernel matrix between such p points and the m Nyström centers; then

$$\mathrm{Tr}\Big(K_{nm}K_{mm}^{\dagger}K_{nm}^{\top}\Big) \approx \frac{n}{p}\mathrm{Tr}\Big(K_{pm}K_{mm}^{\dagger}K_{pm}^{\top}\Big)$$

which can be computed in $pm^2 + m^3$ operations. By choosing $p \sim m$ the runtime is then $O(m^3)$, which does not depend on the dataset size, and is more efficient than the STE approach. Unfortunately, this additional Nyström step cannot be effectively applied for computing Eq. (11) where the inversion of B is the most time-consuming step.

5 Experiments

To validate the objective we are proposing for HP optimization of N-KRR models we ran a series of experiments aimed at answering the following questions:

1. Since our objective is an upper-bound on the test error, is the over-penalization acceptable, and what are its biases?

2. What is its behavior during gradient-based optimization: does it tend to overfit, does it lead to accurate models?

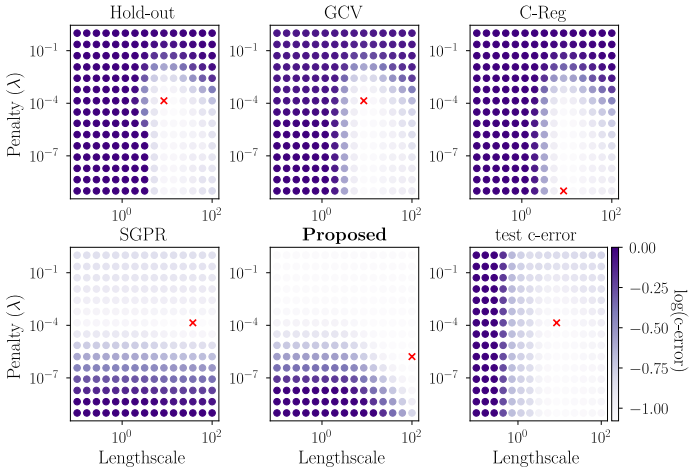


Figure 3: Effectiveness of test error proxies on a grid. The objective values (log transformed) are plotted at different λ, γ points for the *small-HIGGS* dataset. Lighter points indicate a smaller objective and hence a better hyperparameter configuration. The minimum of each objective is denoted by a cross.

3. Does the approximation of Section 4 enable us to actually tune the hyperparameters on large datasets?

The first point is a sanity check: would the objective be a good proxy for the test error in a grid-search scenario over two hyperparameters (λ and γ with the RBF kernel). This doesn't necessarily transfer to larger HP spaces, but gives an indication of its qualitative behavior. In Figure 3 we compare 5 objective functions to the test error on such 2D grid. It is clear that the three functions which are unbiased estimators of the test error have very similar landscapes. Both SGPR and the proposed objective instead have the tendency to *overpenalize*: SGPR strongly disfavors low values of λ, while our objective prefers high λ and γ. This latter feature is associated with simpler models: a high γ produces smooth functions and a large λ restricts the size of the hypothesis.

We will see that the subdivision of objective functions into two distinct groups persists during optimization. However, in general it will not be true that the unbiased objectives produce models with lower test error than the overpenalized ones. The best performing method is going to depend on the dataset.

Small-scale optimization We used the exact formulas, along with automatic differentiation and the Adam optimizer to minimize the objectives on 20 datasets taken from the UCI repository, the LibSVM datasets, or in-house sources (more details on the datasets in Appendix B). We automated the optimization runs as much as possible to avoid having to set many meta-hyperparameters: fixed learning rate, the initial value for λ set to $1/n$ and the initial value for γ set with the median heuristic (Garreau et al., 2017). We used early stopping when the objective values started increasing. The results – shown in Figure 4 – confirm our previous observations: there are some datasets (among which *small-HIGGS*, *buzz*, *house-electric*) on which the unbiased objectives overfit the training set while the proposed proxy function does not. In fact in some cases the hyperparameters found with our objective are much better than the ones found, for example, with the C-Reg objective. On the other hand, there is another group of datasets (e.g. *protein*, *energy* or *codrna*) where the extra bias of the proposed objective becomes detrimental as the optimization gets stuck into a suboptimal configuration with higher test error than what would be attainable with an unbiased objective.

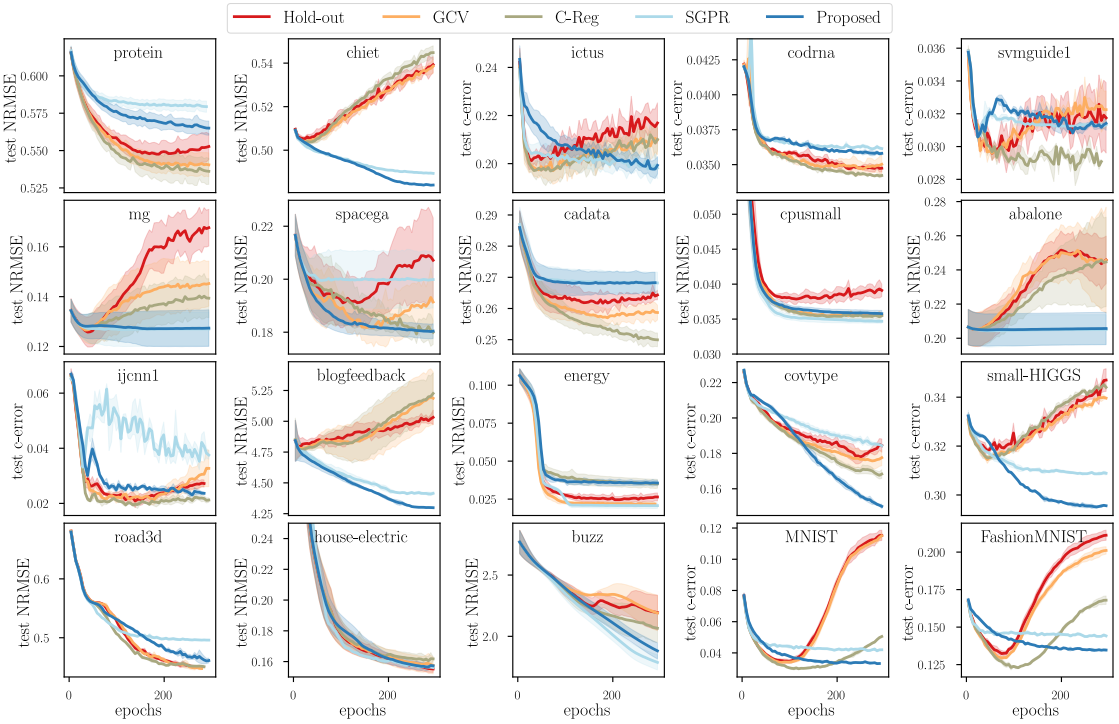


Figure 4: Empirical comparison of five objective functions for hyperparameter tuning. On each dataset we optimized $m = 100$ Nyström centers, a separate lengthscale for each dimension and λ for 200 epochs with a learning rate of 0.05 using the Adam optimizer. Also reported is the standard deviation from 5 runs of the same experiment with a different random seed. Each dataset has its own error metric. Labels of regression datasets were normalized to have unit standard deviation.

Among the three unbiased objectives, hold-out clearly performs the worst. This is due to its high variance, and could be mitigated (at the expense of a higher computational cost) by using k-fold cross-validation. The GCV and C-Reg objectives perform similarly to each other in many cases. Especially in the image datasets however, GCV overfits more than C-Reg.

SGPR closely matches the proposed objective as it doesn't overfit. However, on several datasets it produces worse HPs than our objective displaying a larger bias. On the other hand there are other datasets for which the ranking is reversed, so there is no one clear winner. We must note however that the SGPR objective cannot be efficiently computed due to the log-determinant term, when datasets are large.

Large-scale optimization We tested the performance of the proposed objective with STE on three large-scale datasets, comparing it against two variational sparse GP solvers (Gardner et al., 2018; Matthews et al., 2017) which also learn a compact model with optimized inducing points and a classic N-KRR model with lots of randomly chosen centers trained with Falkon. Our tests are all performed in comparable conditions, details available in Appendix C. The results in Table 1 tell us that we can approach (but not quite reach) the performance – both in terms of speed and accuracy – of a very large model using a small fraction of the inducing points. They also support the conclusion that our objective is effective at optimizing a large number of hyperparameters, at least on par with methods in the GPR framework.

Table 1: Error and running time of kernel solvers on large-scale datasets. We compare our objective with two approximate GPR implementations and hand-tuned N-KRR (Falkon).

		$\mathcal{L}^{\text{Prop}}$	GPyTorch	GPFlow	Falkon
Flights $n \approx 10^6$	error	0.794	0.803	0.790	0.758
	time(s)	355	1862	1720	245
	m	5000	1000	2000	10^5
Flights-Cls $n \approx 10^6$	error	32.2	33.0	32.6	31.5
	time(s)	310	1451	627	186
	m	5000	1000	2000	10^5
Higgs $n \approx 10^7$	error	0.191	0.199	0.196	0.180
	time(s)	1244	3171	1457	443
	m	5000	1000	2000	10^5

6 Conclusions

In this paper, we improved the usability of large scale kernel methods proposing a gradient-based solution for tuning a large number of hyperparameters, on large problems. We incorporate this method into an existing library for large scale kernel methods with GPUs. We showed that it is possible to train compact Nyström KRR models if the centers are allowed to deviate from the training set, which can speed up inference by orders of magnitude. A future work will be to consider complex parametrized kernels which allow to improve the state of the art of kernel-based models on structured datasets such as those containing images or text.

Acknowledgments

The authors would like to thank the Anonymous Reviewers for their helpful comments on trace approximation. Lorenzo Rosasco acknowledges the financial support of the European Research Council (grant SLING 819789), the AFOSR projects FA9550-18-1-7009, FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), the EU H2020-MSCA-RISE project NoMADS - DLV-777826, and the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

References

Arlot, S. (2007). "Resampling and Model selection". PhD thesis. University Paris-Sud (Orsay).

Arlot, S. and F. Bach (2009). "Data-driven calibration of linear estimators with minimal penalties". In: *NeurIPS 22*.

Arlot, S. and A. Celisse (2010). "A survey of cross-validation procedures for model selection". In: *Statistics Surveys* 4, pp. 40–79.

Bartlett, P. L., S. Boucheron, and G. Lugosi (2002). "Model Selection and Error Estimation". In: *Machine Learning* 48.

Bergstra, J. and Y. Bengio (2012). "Random Search for Hyper-Parameter Optimization". In: *J. Mach. Learn. Res.* 13, pp. 281–305.

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Brochu, E., V. M. Cora, and N. de Freitas (2010). *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. arXiv: 1012.2599.

Burt, D. R., C. E. Rasmussen, and M. van der Wilk (2020). "Convergence of Sparse Variational Inference in Gaussian Processes Regression". In: *JMLR* 21, pp. 1–63.

Cao, Y. and Y. Golubev (2006). "On oracle inequalities related to smoothing splines". In: *Mathematical Methods of Statistic* 15.4.

Caponnetto, A. and E. De Vito (2007). "Optimal Rates for the Regularized Least-Squares Algorithm". In: *Foundations of Computational Mathematics* 7, pp. 331–368.

Cawley, G. C. and N. L. C. Talbot (2004). "Fast exact leave-one-out cross-validation of sparse least-squares support vector machines". In: *Neural Networks* 17.10, pp. 1467–1475.

Charlier, B., J. Feydy, J. A. Glaunès, F.-D. Collin, and G. Durif (2021). "Kernel Operations on the GPU, with Autodiff, without Memory Overflows". In: *JMLR* 22.74, pp. 1–6.

Chen, J., H. Avron, and V. Sindhwani (2017). "Hierarchically Compositional Kernels for Scalable Nonparametric Learning". In: *JMLR* 18.1, pp. 2214–2255.

Efron, B. (2004). "The estimation of prediction error: covariance penalties and cross-validation". In: *Journal of the American Statistical Association* 99.467, pp. 619–632.

El Alaoui, A. and M. W. Mahoney (2015). "Fast randomized kernel methods with statistical guarantees". In: *NeurIPS 28*.

Elsken, T., J. H. Metzen, and F. Hutter (2019). "Neural architecture search: A survey". In: *JMLR* 20.1, pp. 1997–2017.

Franceschi, L., M. Donini, P. Frasconi, and M. Pontil (2017). "Forward and Reverse Gradient-Based Hyperparameter Optimization". In: *ICML 34*.

Gardner, J. R., G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration". In: *NeurIPS 31*.

Garreau, D., W. Jitkrittum, and M. Kanagawa (2017). *Large sample analysis of the median heuristic*. arXiv: 1707.07269.

Golub, G. H., M. Heath, and G. Wahba (1979). "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter". In: *Technometrics* 21.2, pp. 215–223.

Grazzi, R., L. Franceschi, M. Pontil, and S. Salzo (2020). "On the Iteration Complexity of Hypergradient Computation". In: *ICML 37*.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. Springer, Berlin.

Hensman, J., N. Fusi, and N. D. Lawrence (2013). "Gaussian Processes for Big Data". In: *UAI*.

Hensman, J., A. Matthews, and Z. Ghahramani (2015). "Scalable variational Gaussian process classification". In: *AISTATS*. PMLR, pp. 351–360.

Hutchinson, M. F. (1990). "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines". In: *Communications in Statistics-Simulation and Computation* 19.2, pp. 433–450.

Hutter, F., L. Kotthoff, and J. Vanschoren, eds. (2018). *Automated Machine Learning: Methods, Systems, Challenges*. Springer.

Keerthi, S. S., V. Sindhwani, and O. Chapelle (2007). "An Efficient Method for Gradient-Based Adaptation of Hyperparameters in SVM Models". In: *NeurIPS 19*.

Kumar, S., M. Mohri, and A. Talwalkar (2012). "Sampling Methods for the Nyström Method". In: *JMLR* 13, pp. 981–1006.

Lorraine, J., P. Vicol, and D. Duvenaud (2020). "Optimizing Millions of Hyperparameters by Implicit Differentiation". In: *AISTATS 23*.

Ma, S. and M. Belkin (2019). "Kernel machines that adapt to GPUs for effective large batch training". In: *Proceedings of the 2nd Conference on Machine Learning and Systems*.

Maclaurin, D., D. Duvenaud, and R. P. Adams (2015). "Gradient-Based Hyperparameter Optimization through Reversible Learning". In: *ICML 32*.

Mallows, C. L. (1973). "Some comments on C_p". In: *Technometrics* 15.4, pp. 661–675.

Massart, P. (2007). *Concentration inequalities and model selection*. Springer, Berlin.

Matthews, A., M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman (2017). "GPflow: A Gaussian process library using TensorFlow". In: *JMLR* 18.40, pp. 1–6.

Meanti, G., L. Carratino, L. Rosasco, and A. Rudi (2020). "Kernel methods through the roof: handling billions of points efficiently". In: *NeurIPS 34*.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). "Scikit-learn: Machine Learning in Python". In: *JMLR* 12, pp. 2825–2830.

Pedregosa, F. (2016). "Hyperparameter optimization with approximate gradient". In: *ICML 33*.

Quiñonero-Candela, J. and C. E. Rasmussen (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression". In: *JMLR* 6.65, pp. 1939–1959.

Rajeswaran, A., C. Finn, S. M. Kakade, and S. Levine (2019). "Meta-Learning with Implicit Gradients". In: *NeurIPS 32*, pp. 113–124.

Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Rudi, A., L. Carratino, and L. Rosasco (2017). "FALKON: An Optimal Large Scale Kernel Method". In: *NeurIPS 29*.

Seeger, M. W. (2008). "Cross-validation optimization for large scale structured classification kernel methods". In: *JMLR* 9, pp. 1147–1178.

Shahriari, B., K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas (2016). "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175.

Smola, A. J. and B. Schölkopf (2000). "Sparse Greedy Matrix Approximation for Machine Learning". In: *Proceedings of the 17th Conference on Machine Learning*.

Snoek, J., H. Larochelle, and R. P. Adams (2012). "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Neurips 25*.

Steinwart, I. and P. Thomann (2017). *liquidSVM: A fast and versatile SVM package*. arXiv: `1702.06899`.

Suykens, J., T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle (2002). *Least Squares Support Vector Machines*. World Scientific.

Titsias, M. (2009). "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *AISTATS 12*.

Tsybakov, A. B. (2003). "Optimal Rates of Aggregation". In: *Learning Theory and Kernel Machines*. Springer, pp. 303–313.

Vapnik, V. N. (1998). "Statistical Learning Theory". In: John Wiley & Sons. Chap. 10.

Varma, S. and R. Simon (2006). "Bias in error estimation when using cross-validation for model selection". In: *BMC Bioinformatics* 91.

Williams, C. K. I. and M. Seeger (2001). "Using the Nyström Method to Speed Up Kernel Machines". In: *NeurIPS 13*.

A Full Derivation of a Complexity Penalty for N-KRR

We split the proof of Theorem 1 into a few intermediate steps: after introducing the relevant notation and definitions we give a few ways in which the Nyström estimator can be expressed, useful in different parts of the proof. Then we proceed with three more technical lemmas, used later on. We split the main proof into two parts to handle the two terms of the decomposition introduced in the main text of the paper: Lemma 7 for the sampling variance and Lemma 8 for the inducing point variance. Finally we restate Theorem 1 for completeness, whose proof follows directly from the two variance bounds.

A.1 Definitions

Using the same notation as in the main text we are given data $\{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$ such that

$$y_i = f^*(x_i) + \epsilon_i$$

where $f^* : \mathcal{X} \to \mathcal{Y}$ is an unknown function, and the noise ϵ_i is such that $\mathbb{E}[\epsilon_i] = 0, \mathbb{E}\left[\epsilon_i^2\right] = \sigma^2$. We let $\mathcal{H}$ be a RKHS and its subspace $\mathcal{H}_m = \text{span}\{k_\gamma(z_1, \cdot), \dots, k_\gamma(z_m, \cdot)\}$ defined using the inducing points $\{z_j\}_{j=1}^m \subset \mathcal{X}$. We define a few useful operators, for vectors $\boldsymbol{v} \in \mathbb{R}^m$ and $\boldsymbol{w} \in \mathbb{R}^n$:

$$\begin{aligned}
&\widetilde{\Phi}_m : \mathcal{H} \to \mathbb{R}^m, \quad && \widetilde{\Phi}_m = (k_\gamma(z_1, \cdot), \dots, k_\gamma(z_m, \cdot)) \\
&\widetilde{\Phi}_m^* : \mathbb{R}^m \to \mathcal{H}, \quad && \widetilde{\Phi}_m^* \boldsymbol{v} = \sum_{j=1}^m \boldsymbol{v}_j k_\gamma(z_j, \cdot) \\
&\Phi : \mathcal{H} \to \mathbb{R}^n, && \Phi = (k_\gamma(x_1, \cdot), \dots, k_\gamma(x_n, \cdot)) \\
&\Phi^* : \mathbb{R}^n \to \mathcal{H}, && \Phi^* \boldsymbol{w} = \sum_{i=1}^n \boldsymbol{w}_j k_\gamma(x_i, \cdot).
\end{aligned}$$

Let $\Sigma : \mathcal{H} \to \mathcal{H} = \Phi^*\Phi$ be the covariance operator, and $K = \Phi\Phi^* \in \mathbb{R}^{n \times n}$ the kernel operator. Further define $K_{nm} = \Phi\widetilde{\Phi}_m^* \in \mathbb{R}^{n \times m}$, $K_{mm} = \widetilde{\Phi}_m\widetilde{\Phi}_m^* \in \mathbb{R}^{m \times m}$, and the approximate kernel $\widetilde{K} = K_{nm}K_{mm}^\dagger K_{nm}^\top \in \mathbb{R}^{n \times n}$. The SVD of the linear operator $\widetilde{\Phi}_m$ is

$$\widetilde{\Phi}_m = U\Lambda V^*$$

with $U : \mathbb{R}^k \to \mathbb{R}^m$, Λ the diagonal matrix of singular values sorted in non-decreasing order, $V : \mathbb{R}^k \to \mathcal{H}$, $k \leq m$ such that $U^*U = I$, $V^*V = I$. The projection operator with range $\mathcal{H}_m$ is given by $P = VV^*$.

The KRR estimator $\hat{f}_{\lambda,\gamma}$ is defined as follows,

$$\hat{f}_{\lambda,\gamma} = \operatorname*{arg\,min}_{f \in \mathcal{H}} \frac{1}{n}\|f(X) - Y\|^2 + \lambda\|f\|_{\mathcal{H}}^2.$$

It can be shown (Caponnetto and De Vito, 2007) that $\hat{f}_{\lambda,\gamma}$ is unique and can be expressed in closed form as $\hat{f}_{\lambda,\gamma} = \Phi^*(K + n\lambda I)^{-1}Y$. In the proofs, we will also use the noise-less KRR estimator, denoted by $f_{\lambda,\gamma}$ and defined as,

$$f_{\lambda,\gamma} = \operatorname*{arg\,min}_{f \in \mathcal{H}} \frac{1}{n}\|f(X) - f^*(X)\|^2 + \lambda\|f\|_{\mathcal{H}}^2.$$

This estimator cannot be computed since we don't have access to f^*, but it is easy to see that

$$f_{\lambda,\gamma} = \Phi^*(K + n\lambda I)^{-1} f^*(X).$$

The N-KRR estimator, found by solving

$$\hat{f}_{\lambda,Z,\gamma} = \underset{f\in\mathcal{H}_m}{\arg\min}\, \frac{1}{n}\|f(X) - Y\|^2 + \lambda\|f\|^2_{\mathcal{H}}.$$

is unique, and takes the form (see Rudi, Camoriano, et al. (2015), Lemma 1)

$$\hat{f}_{\lambda,Z,\gamma} = (P\Sigma P + n\lambda I)^{-1} P\Phi^* Y$$

where P is the projection operator with range $\mathcal{H}_m$.

The estimator $\hat{f}_{\lambda,Z,\gamma}$ can be characterized in different ways as described next.

A.2 Preliminary Results on the Nyström estimator

The following lemma provides three different formulation of the Nyström estimator. We will use the notation $A^\dagger$ to denote the Moore-Penrose pseudo-inverse of a matrix A.

Lemma 2. *(Alternative forms of the Nyström estimator)*
The following equalities hold

$$\hat{f}_{\lambda,Z,\gamma} = (P\Sigma P + n\lambda I)^{-1} P\Phi^* Y \tag{12}$$
$$= V(V^*\Sigma V + n\lambda I)^{-1} V^*\Phi^* Y \tag{13}$$
$$= \widetilde{\Phi}^*_m (K_{nm}^\top K_{nm} + \lambda n K_{mm})^\dagger K_{nm}^\top Y \tag{14}$$

This Lemma is a restatement of results already found in the literature (e.g. in Rudi, Carratino, et al. (2017), Lemmas 2 and 3) which are condensed here with slightly different proofs.

Proof. Going from Eq. (12) to Eq. (13) consists in expanding $P = VV^*$ and applying the push-through identity

$$\begin{aligned}(P\Sigma P + n\lambda I)^{-1} P\Phi^* Y &= (VV^*\Sigma VV^* + n\lambda I)^{-1} VV^*\Phi^* Y \\ &= V(V^*\Sigma VV^*V + n\lambda I)^{-1} V^*\Phi^* Y \\ &= V(V^*\Sigma V + n\lambda I)^{-1} V^*\Phi^* Y.\end{aligned}$$

To go from Eq. (14) to Eq. (13) we split the proof into two parts. We first expand Eq. (14) rewriting the kernel matrices

$$\begin{aligned}\widetilde{\Phi}^*_m (K_{nm}^\top K_{nm} + \lambda n K_{mm})^\dagger K_{nm}^\top Y &= \widetilde{\Phi}^*_m (\widetilde{\Phi}_m \Phi^*\Phi \widetilde{\Phi}^*_m + n\lambda \widetilde{\Phi}_m \widetilde{\Phi}^*_m)^\dagger K_{nm}^\top Y \\ &= \widetilde{\Phi}^*_m (\widetilde{\Phi}_m (\Sigma + n\lambda I) \widetilde{\Phi}^*_m)^\dagger K_{nm}^\top Y.\end{aligned}$$

Then, we use some properties of the pseudo-inverse (Ben-Israel and Greville, 2001) to simplify $(\widetilde{\Phi}_m(\Sigma + n\lambda I)\widetilde{\Phi}^*_m)^\dagger$, in particular, using the SVD of $\widetilde{\Phi}_m$, write

$$(\underbrace{U\Lambda}_{F}\ \underbrace{V^*(\Sigma + n\lambda I)V}_{H}\ \underbrace{\Lambda U^*}_{F^*})^\dagger.$$

Since U has orthonormal columns, $F^\dagger = (U\Lambda)^\dagger = \Lambda^{-1}U^\dagger = \Lambda^{-1}U^*$. A consequence is that $(F^*)^\dagger = (\Lambda U^*)^\dagger = (\Lambda^{-1}U^*)^* = U\Lambda^{-1}$. Then we split $(FHF^*)^\dagger$ into the pseudo-inverse of its three components in two steps. Firstly $(HF^*)^\dagger = (F^*)^\dagger H^\dagger$ if $H^\dagger H = I$ and $(F^*)(F^*)^\dagger = I$:

1. Since $H = V^*(\Sigma + n\lambda I)V$ is invertible, $H^\dagger = H^{-1}$ and the first condition is verified.
2. $F^*(F^*)^\dagger = \Lambda U^* U\Lambda^{-1} = I$.

Also we have $(FHF^*)^\dagger = (HF^*)^\dagger F^\dagger$ if $F^\dagger F = I$ and $HF^*(HF^*)^\dagger = I$:

1. $F^\dagger F = \Lambda^{-1}U^*U\Lambda = I$,
2. $HF^*(HF^*)^\dagger = HF^*(F^*)^\dagger H^\dagger = HH^\dagger = I$.

The end result of this reasoning is that

$$(FHF^*)^\dagger = (F^*)^\dagger H^{-1}F^\dagger = U\Lambda^{-1}(V^*(\Sigma + n\lambda I)V)^{-1}\Lambda^{-1}U^*$$

and hence

$$\begin{aligned}\widetilde{\Phi}_m^*(K_{nm}^\top K_{nm} + \lambda n K_{mm})^\dagger K_{nm}^\top Y &= V\Lambda U^*(U\Lambda V^*(\Sigma + n\lambda I)V\Lambda U^*)^\dagger U\Lambda V^*\Phi^* Y \\ &= V\Lambda U^*U\Lambda^{-1}(V^*(\Sigma + n\lambda I)V)^{-1}\Lambda^{-1}U^*U\Lambda V^*\Phi^* Y \\ &= V(V^*\Sigma V + n\lambda I)^{-1}V^*\Phi^* Y\end{aligned}$$

□

Another useful equivalent form, for the Nyström estimator is given in the following lemma

Lemma 3. *Given the kernel matrices $K_{nm} \in \mathbb{R}^{n\times m}$, $K_{mm} \in \mathbb{R}^{m\times m}$, and the Nyström kernel $\widetilde{K} = K_{nm}K_{mm}^\dagger K_{nm}^\top \in \mathbb{R}^{n\times n}$, the following holds*

$$(\widetilde{K} + n\lambda I)^{-1}\widetilde{K} = K_{nm}(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger K_{nm}^\top \tag{15}$$

Proof. We state some facts about the kernel and image of the Nyström feature maps

$$\begin{aligned}(\ker \widetilde{\Phi}_m)^\perp &= \operatorname{span}\{k(z_1,\cdot),\dots,k(z_m,\cdot)\} = \operatorname{Im}\widetilde{\Phi}_m^* \\ (\ker \widetilde{\Phi}_m^*)^\perp &= \operatorname{Im}\widetilde{\Phi}_m = \operatorname{Im}K_{mm} = (\ker K_{mm})^\perp = W \subseteq \mathbb{R}^m.\end{aligned}$$

The space $\mathbb{R}^m$ is hence composed of $\mathbb{R}^m = W \oplus \ker\widetilde{\Phi}_m^*$. Take a vector $v \in \ker\widetilde{\Phi}_m^*$. We have that $\widetilde{\Phi}_m^* v = 0$, and $(K_{nm}^\top K_{nm} + n\lambda K_{mm})v = \widetilde{\Phi}_m(\Phi^*\Phi + n\lambda I)\widetilde{\Phi}_m^* v = 0$.

If instead $v \in W$, then $\widetilde{\Phi}_m(\Phi^*\Phi + n\lambda I)\widetilde{\Phi}_m^* v \in W$. Hence we have that

$$K_{nm}^\top K_{nm} + n\lambda K_{mm} : W \to W$$

and that K_{mm} is invertible when restricted to the subspace W, but also $K_{nm}^\top K_{nm} + n\lambda K_{mm}$ is invertible on W. Furthermore by the properties of the pseudo-inverse, we have that

$$(K_{nm}^\top K_{nm} + n\lambda K_{mm})(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger = P_W \tag{16}$$

with P_W the projector onto set W.

Furthermore we have the following equalities concerning the projection operator: $K_{mm}^\dagger K_{mm} = P_W$, as before; since $K_{nm} = \Phi\widetilde{\Phi}_m^*$, $K_{nm}P_W = \Phi\widetilde{\Phi}_m^* P_W = K_{nm}$ and similarly its transpose $K_{nm}^\top = \widetilde{\Phi}_m\Phi^*$ hence $P_W K_{nm}^\top = K_{nm}^\top$.

Using these properties we can say

$$\begin{aligned}K_{nm}K_{mm}^\dagger(K_{nm}^\top K_{nm} + n\lambda K_{mm}) &= K_{nm}K_{mm}^\dagger K_{nm}^\top K_{nm} + n\lambda K_{nm}K_{mm}^\dagger K_{mm} \\ &= K_{nm}K_{mm}^\dagger K_{nm}^\top K_{nm}P_W + n\lambda K_{nm}P_W \\ &= (K_{nm}K_{mm}^\dagger K_{nm}^\top + n\lambda I)K_{nm}P_W\end{aligned}$$

which implies that

$$(K_{nm}K_{mm}^\dagger K_{nm}^\top + n\lambda I)^{-1}K_{nm}K_{mm}^\dagger(K_{nm}^\top K_{nm} + n\lambda K_{mm}) = K_{nm}P_W.$$

Multiplying both sides by $(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger$, and using Eq. (16)

$$(K_{nm}K_{mm}^\dagger K_{nm}^\top + n\lambda I)^{-1} K_{nm}K_{mm}^\dagger P_W = K_{nm}P_W(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger \tag{17}$$

Hence we can write the left-hand side of our statement (Eq. (15)), and use the properties of projection P_W and Eq. (17) to get

$$\begin{aligned}(K_{nm}K_{mm}^\dagger K_{nm}^\top + n\lambda I)^{-1} K_{nm}K_{mm}^\dagger K_{nm}^\top &= (K_{nm}K_{mm}^\dagger K_{nm}^\top + n\lambda I)^{-1} K_{nm}K_{mm}^\dagger P_W K_{nm}^\top \\ &= K_{nm}P_W(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger K_{nm}^\top \\ &= K_{nm}(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger K_{nm}^\top\end{aligned}$$

which is exactly the right-hand side of our statement. □

Finally, the algebraic transformation given in the following lemma allows to go from a form which frequently appears in proofs involving the Nyström estimator ($\mathrm{Tr}((I-P)\Sigma)$) to a form which can easily be computed: the trace difference between the full and the Nyström kernel.

Lemma 4. *Let $\widetilde{\Phi}_m : \mathcal{H} \to \mathbb{R}^m$ be the kernel feature-map of the inducing points with SVD $\widetilde{\Phi}_m = U\Lambda V^*$, such that the projection operator onto $\mathcal{H}_m$ can be written $P = VV^*$. Also let $\widetilde{K} = K_{nm}K_{mm}^\dagger K_{nm}^\top$ be the Nyström kernel. Then the following equivalence holds*

$$\mathrm{Tr}((I-P)\Sigma) = \mathrm{Tr}\Big(K - \widetilde{K}\Big). \tag{18}$$

Proof. Note that we can write $K_{mm} = \widetilde{\Phi}_m\widetilde{\Phi}_m^* = U\Lambda V^*V\Lambda U^* = U\Lambda^2 U^*$, which is a full-rank factorization since both $U\Lambda$ and $\Lambda U^\top$ are full-rank. Then we can use the formula for the full-rank factorization of the pseudoinverse (Ben-Israel and Greville (2001), Chapter 1, Theorem 5, Equation 24) to get

$$\begin{aligned}K_{mm}^\dagger &= (U\Lambda V^*V\Lambda U^*)^\dagger = (U\Lambda\Lambda U^*)^\dagger \\ &= U\Lambda(\Lambda U^*U\Lambda^2U^*U\Lambda)^{-1}\Lambda U^* \\ &= U\Lambda^{-2}U^*.\end{aligned}$$

Now we can prove the statement by expanding the left-hand side, and recalling $U^\top U = I$

$$\begin{aligned}\mathrm{Tr}((I-P)\Sigma) &= \mathrm{Tr}((I-VV^*)\Sigma) \\ &= \mathrm{Tr}\big((I - V(\Lambda U^*U\Lambda^{-2}U^*U\Lambda)V^*)\Phi^*\Phi\big) \\ &= \mathrm{Tr}\Big(\Phi(I - V\Lambda U^*(\widetilde{\Phi}_m\widetilde{\Phi}_m^*)^\dagger U\Lambda V^*)\Phi^*\Big) \\ &= \mathrm{Tr}\Big(\Phi\Phi^* - \Phi\widetilde{\Phi}_m^*(\widetilde{\Phi}_m\widetilde{\Phi}_m^*)^\dagger\widetilde{\Phi}_m\Phi^*\Big) \\ &= \mathrm{Tr}\Big(K - K_{nm}K_{mm}^\dagger K_{nm}^\top\Big) = \mathrm{Tr}\Big(K - \widetilde{K}\Big).\end{aligned}$$

□

The following two lemmas provide some ancillary results which are used in the proof of the main lemmas below.

Lemma 5. *Let P be the projection operator onto $\mathcal{H}_m$, and $f_{\lambda,\gamma}$ be the noise-less KRR estimator. Then the following bound holds*

$$\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2 \leq \|f_{\lambda,\gamma}\|^2. \tag{19}$$

Proof. This is a simple application of the definition of operator norm, coupled with the fact that orthogonal projection operators have eigenvalues which are either 0 or 1 (hence their norm is at most 1).

$$\begin{aligned}\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2 &\leq \|P\|^2\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2 \\ &\leq \|f_{\lambda,\gamma}\|_{\mathcal{H}}^2.\end{aligned}$$

□

Lemma 6. *Recall the notation $\hat{L}_\lambda(f) = n^{-1}\|f(X) - Y\|^2 + \lambda\|f\|_{\mathcal{H}}^2$, and let $f_{\lambda,\gamma}$ be the noise-less KRR estimator as before. Then the following statement holds:*

$$\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2 \leq \mathbb{E}\left[\frac{\hat{L}_\lambda(f_{\lambda,\gamma})}{\lambda}\right] \tag{20}$$

where the expectation is taken with respect to the noise.

Proof. Recall that in the fixed design setting, given a fixed (i.e. not dependent on the label-noise) estimator, we always have

$$\mathbb{E}\Big[\hat{L}(f)\Big] = L(f) + \sigma^2$$

where σ^2 is the label-noise variance.

In our case, noting that $L(f_{\lambda,\gamma})$ is always non-negative

$$\begin{aligned}\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2 &= \frac{\lambda}{\lambda}\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2 \\ &\leq \frac{L(f_{\lambda,\gamma}) + \lambda\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2}{\lambda} \\ &\leq \frac{L(f_{\lambda,\gamma}) + \sigma^2 + \lambda\|f_{\lambda,\gamma}\|_{\mathcal{H}}^2}{\lambda} \\ &= \frac{\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big]}{\lambda}.\end{aligned}$$

□

A.3 Proof of the main Theorem

The proof of Theorem 1 starts from the error decomposition found in Section 3 which we report here:

$$\begin{aligned}\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq \mathbb{E}\Big[&\underbrace{L(\hat{f}_{\lambda,Z,\gamma}) - \hat{L}(\hat{f}_{\lambda,Z,\gamma})}_{①} \\ &+ \underbrace{\hat{L}(\hat{f}_{\lambda,Z,\gamma}) + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|_{\mathcal{H}}^2 - \hat{L}_\lambda(Pf_{\lambda,\gamma})}_{②} + \underbrace{\hat{L}_\lambda(Pf_{\lambda,\gamma})}_{③}\Big]\end{aligned}$$

and proceeds by bounding terms ① (see Lemma 7) and ③ (see Lemma 8). After the two necessary lemmas we restate the proof of the main theorem which now becomes trivial.

Lemma 7. *(Bounding the data-sampling variance)*
Denoting by $\hat{f}_{\lambda,Z,\gamma}$ the N-KRR estimator, the expected difference between its empirical and test errors can be calculated exactly:

$$\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma}) - \hat{L}(\hat{f}_{\lambda,Z,\gamma})\Big] = \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big)$$

with σ^2 the noise variance and $\widetilde{K}$ the Nyström kernel.

Proof. For the sake of making the proof self-contained we repeat the reasoning of Section 3. Starting with the expectation of the empirical error we decompose it into the expectation of the test error minus an inner product term:

$$\begin{aligned}\mathbb{E}\Big[\hat{L}(\hat{f}_{\lambda,Z,\gamma})\Big] &= \mathbb{E}\left[\frac{1}{n}\|\hat{f}_{\lambda,Z,\gamma}(X) - f^*(X) - \epsilon\|^2\right] \\ &= \mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] + \sigma^2 - \frac{2}{n}\mathbb{E}\Big[\langle \hat{f}_{\lambda,Z,\gamma}(X) - f^*(X), \epsilon\rangle\Big].\end{aligned}$$

The σ^2 term is fixed for optimization purposes, so we must deal with the inner-product. We use the form of $\hat{f}_{\lambda,Z,\gamma}$ from Eq. (14), Lemma 2, and $\mathbb{E}[\epsilon] = 0$, and to clean the notation we call $H := K_{nm}(K_{nm}^\top K_{nm} + n\lambda K_{mm})^\dagger K_{nm}^\top$:

$$\begin{aligned}\frac{2}{n}\mathbb{E}\Big[\langle \hat{f}_{\lambda,Z,\gamma}(X) - f^*(X), \epsilon\rangle\Big] &= \frac{2}{n}\mathbb{E}[\langle H(f^*(X) + \epsilon) - f^*(X), \epsilon\rangle] \\ &= \frac{2}{n}\mathbb{E}\Big[\epsilon^\top H\epsilon\Big] = \frac{2\sigma^2}{n}\mathrm{Tr}(H),\end{aligned}$$

and using Lemma 3 H can be expressed as $(\widetilde{K} + n\lambda I)^{-1}\widetilde{K}$.

Going back to the original statement we have

$$\begin{aligned}\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma}) - \hat{L}(\hat{f}_{\lambda,Z,\gamma})\Big] &= \mathbb{E}\left[L(\hat{f}_{\lambda,Z,\gamma}) - L(\hat{f}_{\lambda,Z,\gamma}) + \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big)\right] \\ &= \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K} + n\lambda I)^{-1}\widetilde{K}\Big)\end{aligned}$$

□

Lemma 8. *(Bounding the Nyström variance)*
Under the fixed-design assumptions, denote by P the orthogonal projector onto space $\mathcal{H}_m$, by $\hat{L}_\lambda(f)$ the regularized empirical risk of estimator f, and by $f_{\lambda,\gamma} \in \mathcal{H}$ the noise-less KRR estimator. Then the following upper-bound holds

$$\mathbb{E}\Big[\hat{L}_\lambda(Pf_{\lambda,\gamma})\Big] \leq \frac{2}{n\lambda}\mathrm{Tr}\Big(K - \widetilde{K}\Big)\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big] + 2\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big]. \tag{21}$$

Proof. Note that for estimators $f \in \mathcal{H}$ we can always write $f(X) = \Phi f$. Hence for the projected KRR estimator we use that $(Pf_{\lambda,\gamma})(X) = \Phi P f_{\lambda,\gamma}$. We start by rewriting the left hand side to obtain a difference between projected and non-projected terms:

$$\begin{aligned}\mathbb{E}\Big[\hat{L}(Pf_{\lambda,\gamma}) + \lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\Big] &= \mathbb{E}\left[\frac{1}{n}\|\Phi P f_{\lambda,\gamma} - f^*(X) - \epsilon\|^2 + \lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\right] \\ &= \mathbb{E}\left[\frac{1}{n}\|\Phi P f_{\lambda,\gamma} - f^*(X)\|^2 + \frac{1}{n}\|\epsilon\|^2 + \lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\right] \\ &= \mathbb{E}\left[\frac{1}{n}\|\Phi P f_{\lambda,\gamma} - \Phi f_{\lambda,\gamma} + \Phi f_{\lambda,\gamma} - f^*(X)\|^2 + \frac{1}{n}\|\epsilon\|^2 + \lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\right] \\ &\leq \mathbb{E}\left[\frac{2}{n}\|\Phi P f_{\lambda,\gamma} - \Phi f_{\lambda,\gamma}\|^2 + \frac{2}{n}\|\Phi f_{\lambda,\gamma} - f^*(X)\|^2 + \frac{2}{n}\|\epsilon\|^2 + 2\lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\right]\end{aligned}$$

where we used the fact that $\mathbb{E}[\epsilon] = 0$, and the triangle inequality in the last step.

By Lemma 5, and the definition of $\mathbb{E}\Big[\hat{L}(f)\Big]$ we have that

$$\mathbb{E}\left[\frac{2}{n}\|\Phi f_{\lambda,\gamma} - f^*(X)\|^2 + \frac{2}{n}\|\epsilon\|^2 + 2\lambda\|Pf_{\lambda,\gamma}\|_{\mathcal{H}}^2\right] \leq 2\mathbb{E}\Big[\hat{L}(f_{\lambda,\gamma})\Big].$$

Next we use again the definition of operator norm to deal with the difference between projected and non-projected noise-less KRR estimators:

$$\mathbb{E}\Big[\frac{2}{n}\|\Phi P f_{\lambda,\gamma} - \Phi f_{\lambda,\gamma}\|^2\Big] = \frac{2}{n}\|\Phi(P-I)f_{\lambda,\gamma}\|^2 \\ \leq \frac{2}{n}\|\Phi(I-P)\|^2\|f_{\lambda,\gamma}\|^2.$$

The first part of this latter term is

$$\|\Phi(I-P)\|^2 = \|(I-P)\Phi^\top\Phi(I-P)\| \leq \mathrm{Tr}\Big((I-P)\Phi^\top\Phi\Big) = \mathrm{Tr}((I-P)\Sigma)$$

since the trace norm controls the operator norm, and using the cyclic property of the trace and the idempotence of the projection operator $I-P$. By Lemma 4 we have that $\|\Phi(I-P)\|^2 \leq \mathrm{Tr}\Big(K-\widetilde{K}\Big)$. For the second part we use Lemma 6 so that

$$\|f_{\lambda,\gamma}\|^2 \leq \mathbb{E}\left[\frac{\hat{L}_\lambda(f_{\lambda,\gamma})}{\lambda}\right]$$

which concludes the proof. □

We now have all the ingredients to prove Theorem 1 which we restate below for the reader.

Theorem. *(Restated from the main text)*
Under the assumptions of fixed-design regression we have that,

$$\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K}+\lambda I)^{-1}\widetilde{K}\Big) \\ + \frac{2}{n\lambda}\mathrm{Tr}\Big(K-\widetilde{K}\Big)\mathbb{E}\Big[\hat{L}(f_{\lambda,\gamma})\Big] \\ + 2\mathbb{E}\Big[\hat{L}(f_{\lambda,\gamma})\Big] \tag{22}$$

Proof. The decomposition is the same:

$$\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq \mathbb{E}\Big[\underbrace{L(\hat{f}_{\lambda,Z,\gamma}) - \hat{L}(\hat{f}_{\lambda,Z,\gamma})}_{①} \\ + \underbrace{\hat{L}(\hat{f}_{\lambda,Z,\gamma}) + \lambda\|\hat{f}_{\lambda,Z,\gamma}\|^2_{\mathcal{H}} - \hat{L}_\lambda(Pf_{\lambda,\gamma})}_{②} + \underbrace{\hat{L}_\lambda(Pf_{\lambda,\gamma})}_{③}\Big]$$

where ② ≤ 0. We may then use Lemma 7 for term ① and Lemma 8 for term ③ to obtain

$$\mathbb{E}\Big[L(\hat{f}_{\lambda,Z,\gamma})\Big] \leq \frac{2\sigma^2}{n}\mathrm{Tr}\Big((\widetilde{K}+n\lambda I)^{-1}\widetilde{K}\Big) + \frac{2}{n\lambda}\mathrm{Tr}\Big(K-\widetilde{K}\Big)\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big] + 2\mathbb{E}\Big[\hat{L}_\lambda(f_{\lambda,\gamma})\Big].$$

□

B Datasets

We used a range of datasets which represent a wide spectrum of scenarios for which kernel learning can be used. They can be divided into three groups: medium sized unstructured datasets (both for regression and binary classification), medium sized image recognition

datasets (multiclass classification) and large unstructured datasets (classification and regression). We applied similar preprocessing steps to all datasets (namely standardization of the design matrix, standardization of the labels for regression datasets, one-hot encoding of the labels for multiclass datasets). When an agreed-upon test-set existed we used it (e.g. for MNIST), otherwise we used random 70/30 or 80/20 train/test set splits, with each experiment repetition using a different split. Below we provide more details about the datasets used, grouping several of them together if the same procedures apply. The canonical URLs at which the datasets are available, along with their detailed dimensions and training/test splits are shown in Table 2

The error metrics used are dataset-dependent, and outlined below. For regression problems we use the RMSE, defined as $\sqrt{n^{-1}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2}$ and its normalized version the NRMSE:

$$NRMSE : \left| \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2}}{\frac{1}{n}\sum_{i=1}^{n} y_i} \right|.$$

For classification problems we use the fraction of misclassified examples (c-error), and the area under the curve (AUC) metric.

SpaceGA, Abalone, MG, CpuSmall, Energy Small regression datasets between 1385 (MG) and 8192 (CpuSmall) samples, label standardization is performed and error is measured as NRMSE. The predictor matrix is also standardized.

Road3D, Buzz, Protein, HouseElectric, BlogFeedback Regression datasets of medium to large size from the UCI ML repositories. We used label standardization for Road3D, BlogFeedback, Buzz and Protein, and an additional log transformation for HouseElectric. Measured error is NRMSE. The predictor matrix is standardized.

MNIST, FashionMNIST, SVHN, CIFAR-10 Four standard image recognition datasets. Here the labels are one-hot encoded (all datasets have 10 classes), and the design matrix is normalized in the 0-1 range. Standard train/test splits are used.

Chiet A time-series dataset for short-term wind prediction. The labels and predictors are standardized, and the error is measured with the NRMSE. A fixed split in time is used.

Ictus A dataset simulating brain MRI. Predictors are standardized and a random 80/20 split is used.

Cod-RNA, SVMGuide1, IJCNN1, CovType Four datasets for binary classification ranging between approximately 3000 points for SvmGuide1 and 5×10^5 points for CovType. The design matrix is standardized while the labels are -1 and $+1$.

Higgs, SmallHiggs HIGGS is a very large binary classification dataset from high energy physics. We took a small random subsample to generate the SmallHiggs dataset, which has predefined training and test sets. The design matrix is normalized by the features' variance. For the HIGGS dataset we measure the error as 1 minus the AUC.

Flights, Flights-Cls A regression dataset found in the literature (Hensman, Durrande, et al., 2017; Hensman, Fusi, et al., 2013) which can also be used for binary classification by thresholding the target at 0.

Table 2: Key details on the datasets used.

	n	d	train/test	error
SpaceGA	3107	6	70%/30%	NRMSE
Abalone	4177	8	70%/30%	NRMSE
MG	1385	6	70%/30%	NRMSE
CpuSmall	8192	12	70%/30%	NRMSE
Energy	768	8	80%/20%	NRMSE
Road3D	434 874	3	70%/30%	RMSE
Buzz	2 049 280	11	70%/30%	RMSE
Protein	45 730	9	80%/20%	NRMSE
BlogFeedback	60 021	280	52 397/7624	RMSE
MNIST	70 000	784	60 000/10 000	10 class c-error
FashionMNIST	70 000	784	60 000/10 000	10 class c-error
SVHN	99 289	1024	73 257/26 032	10 class c-error
CIFAR-10	60 000	1024	50 000/10 000	10 class c-error
Chiet	34 059	144	26 227/7832	NRMSE
Ictus	29 545	992	80%/20%	binary c-error
Cod-RNA	331 152	8	59 535/271 617	binary c-error
SVMGuide1	7089	4	3089/4000	binary c-error
IJCNN1	141 691	22	49 990/91 701	binary c-error
CovType	581 012	54	70%/30%	binary c-error
SmallHiggs	30 000	28	10 000/20 000	binary c-error
Higgs	1.1×10^7	20	80%/20%	1 - AUC
Flights	5.93×10^6	8	66%/34%	MSE
Flights-Cls	5.93×10^6	8	5 829 413/100 000	binary c-error

C Experiment Details

All experiments were run on a machine with a single NVIDIA Quadro RTX 6000 GPU, and 256GB of RAM. The details of all hyperparameters and settings required to reproduce our experiments are provided below. Relevant code is available in the repository at `https://github.com/falkonml/falkon`.

C.1 Small scale experiments

We ran the small scale experiments by optimizing the exact formulas for all objectives, computed with Cholesky decompositions and solutions to triangular systems of equations. We used the Adam optimizer with default settings and ran it for 200 epochs with a fixed learning rate of 0.05. We optimized $m = 100$ inducing points initialized to the a random data subset, used the Gaussian kernel with a separate length-scale for each data-dimension (the initialization using the median heuristic was the same for each dimension), and the amount of regularization λ which was initialized to $1/n$. The validation set size (for the *Hold-out* objective) was fixed to 60% of the full training data. While this may seem large, in our setting the size of the hyperparameter space (in first approximation $m \times d$) is larger than the number of model parameters ($m \times o$ where o is the dimension of the target space $\mathcal{Y}$, most commonly $o = 1$).

C.2 Large scale experiments

We ran the large-scale experiments for just the $\mathcal{L}^{\text{Prop}}$ objective, while the other performance numbers in Table 1 are taken from Meanti et al. (2020). For our objective we again used the Adam optimizer. For the Flights and Higgs dataset we trained with learning rate 0.05 for 20 epochs, while we trained Flights-Cls with a smaller learning rate of 0.02 for 10 epochs. We used the Gaussian kernel with a single length-scale, initialized as in (Meanti et al., 2020) (Flights $\sigma_0 = 1$, Flights-Cls $\sigma_0 = 1$, Higgs $\sigma_0 = 4$) and $\lambda_0 = 1/n$. We used $t = 20$ stochastic trace estimation vectors for all three experiments, sampling them from the standard Gaussian distribution. The STE vectors were kept fixed throughout optimization. The conjugate gradient tolerance for the Falkon solver was set to 5×10^{-4} for Flights-Cls, and 1×10^{-3} for Flights and Higgs (a higher tolerance corresponds to longer training time), while we always capped the number of Falkon iterations to 100.

References

Ben-Israel, A. and T. N. E. Greville (2001). *Generalized Inverses: Theory and Applications*. 2nd ed. Springer.

Caponnetto, A. and E. De Vito (2007). "Optimal Rates for the Regularized Least-Squares Algorithm". In: *Foundations of Computational Mathematics* 7, pp. 331–368.

Hensman, J., N. Durrande, and A. Solin (2017). "Variational Fourier Features for Gaussian Processes". In: *JMLR* 18.1, pp. 5537–5588.

Hensman, J., N. Fusi, and N. D. Lawrence (2013). "Gaussian Processes for Big Data". In: *UAI*.

Meanti, G., L. Carratino, L. Rosasco, and A. Rudi (2020). "Kernel methods through the roof: handling billions of points efficiently". In: *NeurIPS 34*.

Rudi, A., R. Camoriano, and L. Rosasco (2015). "Less is More: Nyström Computational Regularization". In: *NeurIPS 28*.

Rudi, A., L. Carratino, and L. Rosasco (2017). "FALKON: An Optimal Large Scale Kernel Method". In: *NeurIPS 29*.

Learn Fast, Segment Well: Fast Object Segmentation Learning on the iCub Robot

Federico Ceola, *Student Member, IEEE,* Elisa Maiettini, Giulia Pasquale, Giacomo Meanti, Lorenzo Rosasco, and Lorenzo Natale, *Senior Member, IEEE*

***Abstract*—The visual system of a robot has different requirements depending on the application: it may require high accuracy or reliability, be constrained by limited resources or need fast adaptation to dynamically changing environments. In this work, we focus on the instance segmentation task and provide a comprehensive study of different techniques that allow adapting an object segmentation model in presence of novel objects or different domains.**

We propose a pipeline for fast instance segmentation learning designed for robotic applications where data come in stream. It is based on an hybrid method leveraging on a pre-trained CNN for feature extraction and fast-to-train Kernel-based classifiers. We also propose a training protocol that allows to shorten the training time by performing feature extraction during the data acquisition. We benchmark the proposed pipeline on two robotics datasets and we deploy it on a real robot, i.e. the iCub humanoid. To this aim, we adapt our method to an incremental setting in which novel objects are learned on-line by the robot.

The code to reproduce the experiments is publicly available on GitHub[1].

***Index Terms*—Visual Learning, Object Detection, Segmentation and Categorization, Humanoid Robots, Efficient Instance Segmentation Learning.**

I. INTRODUCTION

PERCEIVING the environment is the first step for a robot to interact with it. Robots may be required to solve different tasks, as for instance grasping an object, interacting with a human or navigate in the environment avoiding obstacles.

Different applications have different requirements for the robot vision system. For example, for an application in which a robot interacts with a predefined set of objects, fast learning is not the primary requirement. On the other hand, when a robot is operating in a dynamic environment (for instance a service robot operating in a hospital, a supermarket or a domestic environment), fast adaptation is fundamental.

The *computer vision* literature is progressing at fast pace providing algorithms for object detection and segmentation that are remarkably powerful. These methods, however, are mostly based on deep neural networks and quite demanding in terms of training samples and optimization time. For this reason, they are badly suited for applications in robotics that require fast adaptation. Because the dominant trend in computer vision is to push performance as much as possible, comparably little effort is spent to propose methods that are designed to reduce training time. To fill this gap, in this work, we propose a comprehensive analysis in which we study various techniques for adaptation on a novel task. In particular, we consider approaches based on deep neural networks and on a combination of deep neural networks and Kernel methods, focusing on the trade-off between training time and accuracy.

We target the instance segmentation problem which consists in classifying every pixel of an image as belonging to an instance of a known object or to the background. In particular, we consider the scenario in which the robot encounters new objects during its operation and it is required to adapt its vision system so that it is able to segment them after a learning session that is as short as possible. We observe that this scenario offers opportunities to shorten the training time, for example if we are able to perform some of the training steps (i.e., feature extraction) already during data acquisition, and we propose a new method that is specifically optimized to reduce training time without compromising performance.

Specifically, we propose an instance segmentation pipeline which extends and improves our previous work [1]. In [1], we proposed a fast learning method for instance segmentation of novel objects. One limitation of that method was to rely on a pre-trained region proposal network. In this work, we address this by making the region proposal learning on-line too. While this improves performance, it leads to a more complex and longer training pipeline if addressed naïvely as it is done in [2]. To this aim, we propose an approximated training protocol which can be separated in two steps: *(i)* feature extraction and *(ii)* fast and simultaneous training of the proposed approaches for region proposal, object detection and mask prediction. We show that this allows to further reduce the training time in

Federico Ceola, Elisa Maiettini, Giulia Pasquale and Lorenzo Natale are with Humanoid Sensing and Perception (HSP), Istituto Italiano di Tecnologia (IIT), Genoa, Italy (email: name.surname@iit.it).

Federico Ceola, Giacomo Meanti (email: giacomo.meanti@edu.unige.it) and Lorenzo Rosasco are with Laboratory for Computational and Statistical Learning (LCSL), with Machine Learning Genoa Center (MaLGa) and with Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi (DIBRIS), University of Genoa, Genoa, Italy.

Lorenzo Rosasco is also with Istituto Italiano di Tecnologia (IIT) and with Center for Brains, Minds and Machines (CBMM), Massachusetts Institute of Technology (MIT), Cambridge, MA (email: lrosasco@mit.edu).

[1]https://github.com/hsp-iit/online-detection

the aforementioned robotic scenario.

In addition, we provide an extensive experimental analysis to investigate the training time/accuracy trade-off on two public datasets (i.e., YCB-Video [3] and HO-3D [4]). In particular, we show that our method is much more accurate than [1], while requiring a comparable training time. Moreover, the proposed method allows to obtain accuracy similar to conventional fine-tuning approaches, while being trained much faster.

In summary, the contributions of this work are:

- We propose a new pipeline and training protocol for instance based object segmentation, which is specifically designed for fast, on-line training.
- We benchmark the obtained results on two robotics datasets, namely YCB-Video [3] and HO-3D [4].
- We provide an extensive study to compare our pipeline against conventional fine-tuning techniques, with an in-depth analysis of the trade-off between the required training time and the achieved accuracy.
- We deploy and demonstrate the proposed training pipeline on the iCub [5] humanoid robot, adapting the algorithm for an incremental setting where target classes are not known a-priori.

This paper is organized as follows. In Sec. II, we review state-of-the-art approaches for instance segmentation, focusing on methods designed for robotics. Then, in Sec. III, we describe the proposed training pipeline for fast learning of instance segmentation. In Sec. IV, we report on the experimental setup used to validate our approach. We then benchmark our approach on the two considered robotics datasets in Sec. V. In Sec. VI, we specifically quantify the benefit of the adaptation of the region proposal. In Sec. VII, we simulate the robotic scenario in which data come into stream and we discuss various performance trade-offs. Then, in Sec. VIII, we describe an incremental version of the proposed pipeline and we deploy it on a robotic platform. Finally, in Sec. IX we draw conclusions.

II. RELATED WORK

In this section, we provide an overview of state-of-the-art methods for instance segmentation (Sec. II-A), focusing on their application in robotics (Sec. II-B).

A. Instance Segmentation

Approaches proposed in the literature to address instance segmentation can be classified in the following three groups.

Detection-based instance segmentation. Methods in this category extend approaches for object detection, by adding a branch for mask prediction within the bounding boxes proposed by the detector. Therefore, as for object detection methods, they can be grouped in *(i) multi-stage* (also known as *region-based*) and *(ii) one-stage*. Methods from the first group rely on detectors that firstly predict a set of candidate regions and then classify and refine each of them (e.g. Faster R-CNN [6] or R-FCN [7]). *One-stage* detectors, instead, solve the object detection task in one forward pass of the network. Differently from *multi-stage* approaches, they do not perform any per-region operation, like e.g. per-region feature extraction and classification (see for instance, EfficientDet [8] and YOLOv3 [9]).

The representative method among the *multi-stage* approaches is Mask R-CNN [10] that builds on top of the detection method Faster R-CNN [6], by adding a branch for mask prediction (segmentation branch) in parallel to the one for bounding box classification and refinement (detection branch). In Mask R-CNN, input images are initially processed by a convolutional backbone to extract a feature map. This is then used by the Region Proposal Network (RPN) to propose a set of *Regions of Interest* (*RoIs*) that are candidate to contain an object, by associating a class-agnostic objectness score to each region. Then, the *RoI Align* layer associates a convolutional feature map to each *RoI* by *warping* and *cropping* the output of the backbone. These features are finally used for *RoIs* classification, refinement and, subsequently, for mask prediction. In the literature, many other state-of-the-art *multi-stage* approaches for instance segmentation build on top of Mask R-CNN, like Mask Scoring R-CNN [11] or PANet [12].

YOLACT [13] and BlendMask [14] are representative of *one-stage* methods. YOLACT [13] extends a backbone RetinaNet-like [15] detector with a segmentation branch. BlendMask [14], instead, extends FCOS [16] for mask predictions. An alternative paradigm for instance segmentation based on the *one-stage* detector CenterNet [17] is Deep Snake [18]. Differently from the methods mentioned above that predict per-pixel confidence within the proposed bounding boxes, it exploits the circular convolution [18] to predict an offset for each mask vertex point, starting from an initial coarse contour.

Labelling pixels followed by clustering. Approaches in this group build on methods for semantic segmentation, which is the task of classifying each pixel of an image according to its category (being thus agnostic to different object instances). Building on these methods, approaches in the literature separate the different instances by clustering the predicted pixels. As an example, SSAP [19] uses the so-called *affinity pyramid* in parallel with a branch for semantic segmentation to predict the probability that two pixels belong to the same instance in a hierarchical manner. This is done with the aim of grouping pixels of the same instance. InstanceCut [20], instead, exploits an instance-agnostic segmentation and an instance-aware edge predictor to compute the instance-aware segmentation of an image. Finally, the method proposed in [21] learns the watershed transform with a convolutional neural network, the *Deep Watershed Transform*, given an image and a semantic segmentation. This is done to predict an energy map of the image, where the energy basins represent the object instances. This information is then used, with a cut at a single energy level, to produce connected components corresponding to different object instances.

Dense sliding window. These approaches simultaneously

predict mask instances and their class-agnostic or class-specific scores. For instance, DeepMask [22] predicts in parallel a class-agnostic mask and an objectness score for each patch of an input image with a shallow convolutional neural network. InstanceFCN [23], alternatively, predicts an instance sensitive score map for each window of the considered input image. This method exploits local coherence for class-agnostic masks prediction, and, as DeepMask, per-window class-agnostic scores. Similarly, TensorMask [24] predicts class-agnostic instance masks, but it leverages on the proposed mask representation as a 4D tensor to preserve the spatial information among pixels. Moreover, the classification branch of the proposed approach outputs a class-specific score, thus improving the class-agnostic predictions provided by DeepMask and InstanceFCN.

B. Instance Segmentation in Robotics

The instance segmentation task plays a central role in robotics, not only for providing an accurate 2D scene description for a robot, but also to support other tasks like 6D object pose estimation [3] or computation of grasp candidates [25]. In the literature, the problem is tackled in different ways, depending on the target application. In [26] and [27] the problem is addressed in cluttered scenarios, while [28] and [29] propose adopting synthetic data (both images and depth information) for training. In this work, instead, we focus on learning to segment previously unseen objects. In the following paragraphs, we will cover the main literature on this topic.

Some works propose to generalize to unseen objects in a class-agnostic fashion. However, these methods either focus on particular environments, such as tabletop settings, as in [30] and [31], or require some post-processing [32] which may be unfeasible during the robot operation.

Approaches as the ones proposed in [33] and [34] learn to segment new objects instances by interacting with them. Nevertheless, similarly to the class-agnostic approaches, they are constrained to tabletop settings.

The latest literature on *Video Object Segmentation* provides some methods for learning to segment a set of previously unseen objects in videos. They deal with the problem either in a semi-supervised way [35], leveraging on the ground-truth masks of the objects in the first frame of the video, or in an unsupervised fashion [36]. They allow to learn to segment new object instances in a shorter time than that required by the fully supervised approaches presented in Sec. II-A. They typically rely on pre-training a network for instance segmentation and on the subsequent fine-tuning on the target video sequence frames [37]. Some of these approaches have been targeted for robotic scenarios. For instance, the method in [38] proposes to learn to segment novel objects in a *Human-Robot Interaction (HRI)* application, leveraging only on objects motion cues. Nevertheless, these approaches are known to suffer from changes of the objects appearance through the video sequence and error drifts [35].

We instead focus on learning to segment novel objects in a class-specific fashion, keeping the performance provided by the state-of-the-art but reducing the required training time. All the approaches mentioned in Sec. II-A rely on convolutional neural networks that require to be trained end-to-end via backpropagation and stochastic gradient descent. Despite providing impressive performance, they require long training time and large amounts of labeled images to be optimized. These constraints make the adoption of such approaches in robotics difficult, especially for robots operating in unconstrained environments, that require fast adaptation to new objects.

Incremental learning aims at learning new objects instances without degrading performance on the previously known classes. Nevertheless, these approaches rarely focus on speeding-up the training of the models, which may be crucial in robotic applications. Moreover, the current literature in this field mainly focuses on object recognition [39], [40], object detection [41], [42] or semantic segmentation problems [43], while we target an instance segmentation application. As we show in Sec. VIII, we deploy the proposed pipeline on the iCub humanoid robot, adapting it to an incremental setting, where the target classes are not known a-priori.

In this work, we propose a pipeline and a training protocol for instance segmentation which is specifically designed to reduce training time, while preserving performance as much as possible. This approach is based on Mask R-CNN [10], in which the final layers of the RPN and of the detection and segmentation branches have been replaced with "shallow" classifiers based on a fast Kernel-based method optimized for large scale problems [44], [45]. The backbone of the network is trained off-line, while the Kernel-based classifiers are adapted on-line. In this paper, we build on our previous work [1], in that we include the adaptation of the region proposal network and a novel training protocol which allows to further reduce the training time. This makes the pipeline suitable for on-line implementation.

III. METHODS

The proposed hybrid pipeline allows to quickly learn to predict masks of previously unseen objects (TARGET-TASK). We rely on convolutional weights pre-trained on a different set of objects (FEATURE-TASK) and we rapidly adapt three modules for region proposal, object detection and mask prediction on the new task. This allows to achieve on-line adaptation on novel objects and visual scenarios.

A. Overview of the Pipeline

The proposed pipeline is composed of four modules, which are depicted in Fig. 1. They are:

- **Feature Extraction Module.** This is composed of the first layers of Mask R-CNN, which has been pre-trained off-line on the FEATURE-TASK. We use it to extract the convolutional features to train the three on-line modules on the TARGET-TASK. In particular, we

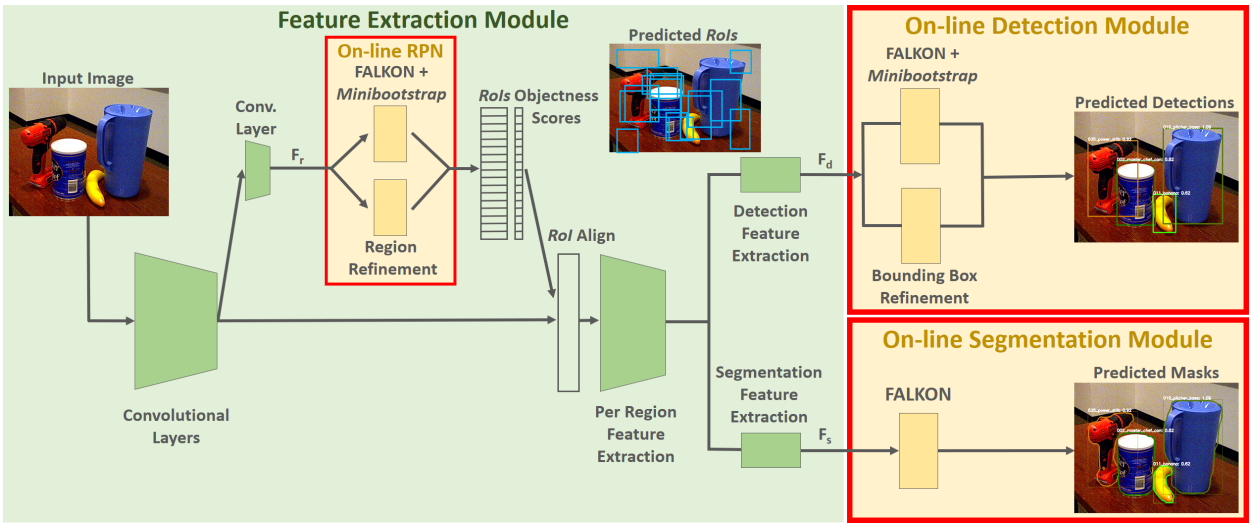


Fig. 1: **Overview of the proposed pipeline**. The *Feature Extraction Module* is composed of Mask R-CNN's first layers trained off-line on the FEATURE-TASK. The three sets of features for *(i)* region proposal (F_r), *(ii)* object detection (F_d) and *(iii)* instance segmentation (F_s) are fed to *(i)* the *On-line RPN*, *(ii)* the *On-line Detection Module* and *(iii)* the *On-line Segmentation Module*. At inference time, we substitute the final layers of the Mask R-CNN's RPN with the *On-line RPN* trained on the TARGET-TASK and, as in Mask R-CNN, the output of the *On-line Detection Module* is fed as input to the *RoI Align* to compute the objects masks within the proposed bounding boxes.

use it to extract the features F_r, F_d and F_s from the penultimate layers of the RPN, and of the detection and segmentation branches, respectively.

- **On-line RPN.** This replaces the last layers of the Mask R-CNN's RPN to predict a set of regions that likely contain an object in an image, given a feature map F_r. We describe the training procedure in Sec. III-B.
- **On-line Detection Module.** This is composed of classifiers and regressors that, starting from a set of feature tensors F_d, classify and refine the regions proposed by the *On-line RPN*. See Sec. III-B for the description of the training procedure.
- **On-line Segmentation Module.** Given a feature map F_s, this module predicts the masks of the objects within the detections proposed by the *On-line Detection Module*. We describe the training procedure in Sec. III-C.

In the three on-line modules described above, we use FALKON for classification. This is a Kernel-based method optimized for large-scale problems [44]. In particular, we use the implementation available in [45].

B. Bounding Box Learning

The prediction of region proposal candidates and object detection are problems that share similarities. In both cases, input bounding boxes are classified and then refined, starting from associated feature tensors as input. In our pipeline, these problems are carried out by the *On-line RPN* and the *On-line Detection Module*, which are implemented by N_c FALKON binary classifiers and $4N_c$ Regularized Least Squares (RLS) regressors [46]. Specifically, N_c represents: *(i)* the number of anchors for the *On-line RPN* (see the following paragraphs) or *(ii)* the number of semantic classes of the TARGET-TASK for the *On-line Detection Module*. In both the on-line modules, we tackle the well known problem of foreground-background imbalance of training samples in object detection [15] by adopting the *Minibootstrap* strategy proposed in [47], [48] for FALKON training. The *Minibootstrap* is an approximated procedure for hard negatives mining [49], [46] that allows to iteratively select a subset of hard negative samples to balance the training sets associated to each of the N_c classes. We report the pseudo-code of the *Minibootstrap* procedure in App. E. The RLS regressors for boxes refinement, instead, are trained on a set of positive (foreground) instances.

In the *On-line RPN*, the classifiers are trained on a binary task to discriminate anchors representing the background from those representing *RoIs*, i.e., containing an instance of any of the TARGET-TASK classes. An anchor [6] is a bounding box of a predefined size and aspect ratio centered on an image pixel. For each pixel, there are a fixed number of anchors of different form factors and one classifier is instantiated for each of them. In the *On-line Detection*, instead, a binary classifier is instantiated for each class. Each classifier is trained to discriminate regions proposed by the *On-line RPN* depicting an object of its class from other classes or background.

On-line RPN. In Mask R-CNN's RPN, the classification is performed on a set of anchors A (i.e., $N_c = A$). Given the input feature map computed by the backbone of height h, width w and with f channels, the RPN firstly processes it with a convolutional layer to obtain a feature map F_r of the same size ($h \times w \times f$). Then, F_r is processed by two convolutional layers. One is composed of A convolutional

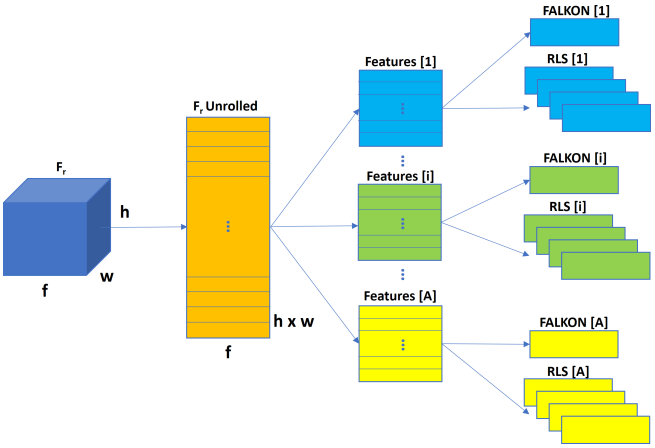


Fig. 2: **On-line RPN.** Given the feature map F_r, this is unrolled into $h \times w$ tensors of features of size f (F_r *Unrolled*). A subset of these features is chosen to train a FALKON classifier and four RLS regressors for each anchor.

kernels which compute the objectness scores of each considered anchor. This layer computes an output tensor of size $h \times w \times A$, in which the ija^{th} element is the objectness score of the a^{th} anchor in the location (i, j). The other output layer, instead, is composed of $4A$ kernels for the refinement of such bounding boxes. It computes $h \times w \times 4A$ values for the refinement of the regions associated to the anchors at each location (i, j). Both the output convolutional kernels have size 1 and stride 1.

As shown in Fig. 2, we replace the A convolutional kernels for the computation of the objectness scores with A FALKON binary classifiers and we train them with the *Minibootstrap*. We use the $h \times w$ tensors of size f resulting from the flattening of the feature maps F_r as training features. The considered positive features are those associated to a specific location of an anchor whose *Intersection over Union* (*IoU*) with at least a ground-truth bounding box is greater than 0.7[2] (in case there are no anchors overlapping the ground-truth bounding boxes with *IoU* > 0.7, the ones with the highest *IoU* are chosen as positives). The feature tensors for the background, instead, are those whose *IoU* with the ground-truths is smaller than 0.3. Similarly, we replace the $4A$ convolutional kernels for the refinement of the proposed regions with $4A$ RLS regressors (4 RLS for each anchor). We consider as training samples for the 4 regressors associated to each anchor a set of features chosen as the positive samples for the FALKON classifiers, but setting the *IoU* threshold to 0.6[3].

On-line object detection. We train the *On-line Detection Module* with the strategy illustrated above, considering the N classes of the TARGET-TASK (i.e., $N_c = N$). As training samples, we consider the tensors of features produced by the penultimate layer of the Mask R-CNN's detection branch (F_d) associated to each *RoI* proposed by the region proposal method. In particular, we consider as positive samples for the n^{th} FALKON classifier, those *RoIs* with *IoU* > 0.6[4] with a ground-truth box of an instance of class n ($n \in [1, \ldots, N]$). The same positive samples are also used for training the n^{th} RLS regressor[3]. Then, as negative samples, we consider the *RoIs* with *IoU* < 0.3[5] with the ground-truths of class n.

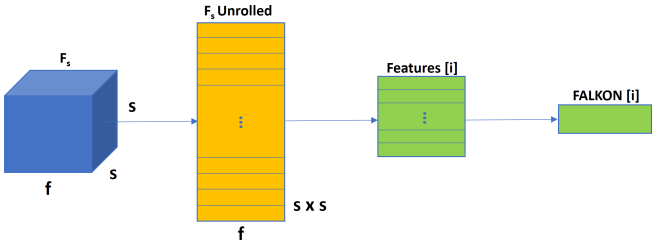


Fig. 3: **On-line Segmentation.** Given the feature map F_s associated to a *RoI* of class i, this is unrolled into $s \times s$ tensors of features of size f (F_s *Unrolled*) from which positive and negative features are sampled to train the i^{th} FALKON per-pixel classifier. Note that this procedure is performed for each *RoI* of the N classes.

C. On-line Segmentation

In Mask R-CNN, in the configuration that does not use the *Feature Pyramid Network* (FPN) [50] in the backbone, the segmentation branch is a shallow fully convolutional network (FCN) composed of two layers that takes as input a feature map of size $s \times s \times f$ associated to each *RoI*. The first layer processes the input feature map into another feature map F_s of the same size. The last convolutional layer, instead, has N channels (one for each class of the TARGET-TASK) and kernel size 1 and stride 1. Therefore, the output of the Mask R-CNN's segmentation branch is a tensor of size $s \times s \times N$, where the ijn^{th} value of such tensor represents the confidence that the pixel in the location (i, j) of the *RoI* corresponds to the n^{th} class.

For the fast learning of the *On-line Segmentation Module*, we rely on the first layer of the segmentation branch for feature extraction, but we substitute the last convolutional layer for per-pixel prediction with N FALKON binary classifiers. To train such classifiers, we consider the ground-truth boxes of each training image, we compute the feature map of size $s \times s \times f$ for each of them and we flatten each of such feature maps into $s \times s$ tensors of size f, as shown in Fig. 3. Among these tensors, we consider as positive samples for the n^{th} classifier the features associated to the pixels in the ground-truth masks of class n. Instead, we consider as negative samples the features associated to the background pixels contained in ground-truth bounding boxes of class

[2]For the *On-line RPN*, we set the positive and negative thresholds for the classifiers as in Mask R-CNN's RPN [10].

[3]We set the value of the *IoU* threshold for the RLS regressors as in R-CNN [46].

[4]We consider as positive samples for the classifiers in the *On-line Detection Module* the training features for region refinement as in [10].

[5]For the classifiers in the *On-line Detection Module* we define the negative samples as in [46].

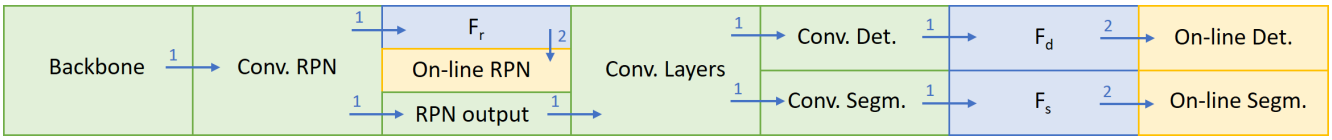


Fig. 4: **Ours** training protocol. We rely on the feature extraction layers of Mask R-CNN pre-trained on the FEATURE-TASK to simultaneously extract F_r, F_d and F_s. We then use these features to train the three on-line modules on the TARGET-TASK. The values on the arrows correspond to the training steps in Sec. III-D.

n. Given the great amount of training samples, to speed-up the training procedure, we randomly subsample both the positive and the negative features by a factor r. According to the analysis provided in [1], we set r to 0.3.

D. Training Protocol

In this work, we propose a training protocol that allows to quickly update the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. The proposed method (referred to as **Ours**) starts with the weights of Mask R-CNN pre-trained on the FEATURE-TASK and adapts the on-line modules on the TARGET-TASK. This is composed of the two steps depicted in Fig. 4:

1) *Feature extraction.* This is done with a forward pass of the pre-trained Mask R-CNN feature extractor to compute F_r, F_d and F_s.
2) *On-line training.* The set of features F_r, F_d and F_s are used to respectively train *(i)* the *On-line RPN*, *(ii)* the *On-line Detection Module* and *(iii)* the *On-line Segmentation Module* on the TARGET-TASK.

The training features fed to the *On-line Detection Module* are those associated to the regions proposed by the Mask R-CNN's RPN pre-trained on the FEATURE-TASK. These regions are different (and therefore sub-optimal) with respect to the ones that would be proposed by a region proposal method that has been adapted on the TARGET-TASK. Training the *On-line Detection Module* using features extracted from the *On-line RPN* after its adaptation is possible. This, however, would require two feature extraction steps (one for training the *On-line RPN* and the other to train the *On-line Detection Module* and the *On-line Segmentation Module*), which is computationally expensive. For this reason, we consider the *On-line Detection Module* obtained with **Ours** as an approximation of the one that would be provided by the serial training (see Sec. VI-B). Instead, the adaptation of the *On-line Segmentation Module* is not affected by this approximation, since we sample the training features for this module from the ground-truth bounding boxes.

While this approximation is a key component of the proposed training protocol, at inference time features fed to the *On-line Detection Module* and to the *On-line Segmentation Module* are those associated to the regions proposed by the *On-line RPN* trained on the TARGET-TASK, as depicted in Fig. 1.

In this work, we show that a single feature extraction step can be performed paying a small price in terms of accuracy (see Sec. VI-B), allowing to further improve the training in the on-line implementation (see Sec. VII and Sec. VIII).

IV. EXPERIMENTAL SETUP

In this section, we report on the experimental settings that we employ for validating the proposed approach. We first evaluate our approach in an off-line setting (Sec. V, VI and VII), analyzing performance on two different robotics datasets. Then, we validate it in a real robotic application (Sec. VIII), i.e., in an on-line setting. Therefore, in this section, we firstly report on the off-line experimental setup (Sec. IV-A) and the datasets (Sec. IV-B) that we use in our experiments. Then, in Sec. IV-C, we describe the settings considered for the deployment in a real robotic scenario.

A. Off-line Experiments

For our experiments, we compare the proposed method, **Ours**, with two Mask R-CNN [10] baselines. In particular, we consider:

- **Mask R-CNN (output layers)**: starting from the Mask R-CNN weights pre-trained on the FEATURE-TASK, we re-initialize the output layers of the RPN and of the detection and segmentation branches, and we fine-tune them on the TARGET-TASK, freezing all the other weights of the Mask R-CNN network.
- **Mask R-CNN (full)**: we use the weights of Mask R-CNN pre-trained on the FEATURE-TASK as a warm-restart to train Mask R-CNN on the TARGET-TASK.

Specifically, we rely on Mask R-CNN [10], using ResNet-50 [51] as backbone, for the feature extraction of **Ours** and for the baselines. In App. A, we report a summary of the training protocols used in this work.

In all cases, we choose hyper-parameters providing the highest performance on a validation set. Specifically, for **Ours** we cross-validate the standard deviation of FALKON's Gaussian kernels (namely, σ) and FALKON's regularization parameter (namely, λ) for the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. Regarding the baselines, instead, for the experiments in Sec. V and VI, we train **Mask R-CNN (output layers)** and **Mask R-CNN (full)** for the number of epochs that provides the highest segmentation accuracy on the validation set.

For **Ours**, we set the number of Nyström centers M of the FALKON classifiers composing the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module* to 1000, 1000 and 500, respectively. Moreover, to train both

the *On-line RPN* and the *On-line Detection Module*, we set to 2000 the size of the batches, BS, considered in the *Minibootstrap*.

Evaluation metrics. We consider the *mean Average Precision (mAP)* as defined in [52] for both object detection and segmentation. Specifically, the accuracy of the predicted bounding boxes will be referred to as **mAP bbox(%)** and the accuracy of the mask instances as **mAP segm(%)**. For each of them, we consider as positive matches the bounding boxes and the masks whose *IoU* with the ground-truths is greater or equal to a threshold. In our experiments we consider two different thresholds to evaluate different levels of accuracy, namely, 50% (**mAP50**) and 70% (**mAP70**). In App. A we overview the acronyms considered for *mAP* computation. We also evaluate the methods on the time required for training[6]. For the Mask R-CNN baselines, the training time is the time needed for their optimization via backpropagation and stochastic gradient descent. As regards **Ours**, instead, except where differently specified, it is the time necessary for extracting the features and training the on-line modules. For each experiment, we run three trials for each method. We report results in terms of average and standard deviation of the accuracy and of average training time.

B. Datasets

In our experiments we consider three datasets. Specifically, we use MS COCO [53] as FEATURE-TASK and the two datasets YCB-Video [3] and HO-3D [4] as TARGET-TASKs to validate our approach. We opted to validate our system on these datasets, which are composed of streams of frames in tabletop and hand-held settings, to be close to our target application. These datasets are usually considered for the task of 6D object pose estimation, however, they are annotated also with object masks. Specifically:

- **MS COCO** [53] is a general-purpose dataset, which contains 80 objects categories, for object detection and segmentation.
- **YCB-Video** [3] is a dataset for 6D pose estimation in which 21 objects from the YCB [54] dataset are arranged in cluttered tabletop scenarios, therefore presenting strong occlusions. It is composed of video sequences where the tabletop scenes are recorded under different viewpoints. We use as training images a set of 11320 images, obtained by extracting one image every ten from the total 80 training video sequences. As test set, instead, we consider the 2949 *keyframe* [3] images chosen from the remaining 12 sequences. For hyper-parameters cross-validation, we randomly select a subset of 1000 images from the 12 test sequences, excluding the *keyframe* set.
- **HO-3D** [4] is a dataset for hand-object pose estimation, in which objects are a subset of the ones in **YCB-Video**. It is composed of video sequences, in which a moving hand-held object is shown to a fixed camera. For choosing the training and test sets, we split the available annotated sequences in HO-3D[7], such that, we gather one and at most four sequences for testing and training, respectively. In particular, we use 20156 images as training set, which result from the selection of one every two images from 34 sequences. Instead, we consider as test set 2020 images chosen one every five frames taken from other 9 sequences. For hyper-parameters cross-validation, we consider 2160 frames chosen one every five images, from a subset of 9 sequences taken from the training set (see App. B for further details).

C. Robotic Setup

We deploy the proposed pipeline for on-line instance segmentation on the humanoid robot iCub[8] [5]. It is equipped with a *Intel(R) RealSense D415* on a headset for the acquisition of RGB images and depth information. We rely on the YARP [55] middleware for the implementation and the communication between the different modules (see Sec. VIII). With the exception of the proposed one, we rely on publicly available modules[9]. We set all the training hyper-parameters as described in Sec. IV-A.

V. RESULTS

In this section, we benchmark the proposed approach on YCB-Video (Sec. V-A) and HO-3D (Sec. V-B).

A. Benchmark on YCB-Video

We consider the 21 objects from YCB-Video as TARGET-TASK and we compare the performance of **Ours** against the baseline **Mask R-CNN (output layers)**. We also report the performance of **Mask R-CNN (full)**, which can be considered as an upper-bound because, differently from the proposed method, it updates both the feature extraction layers and the output layers (i.e., the backbone, the RPN and the detection and segmentation branches) fitting more the visual domain of the TARGET-TASK. In **Ours**, we empirically set the number of batches in the *Minibootstrap* to 10, to achieve the best training time/accuracy trade-off (see Fig. 6 for details).

Results in Tab. I show that **Ours** achieves similar performance as **Mask R-CNN (output layers)** in a fraction ($\sim\mathbf{12.8}\times$ smaller) of the training time. In comparison with the upper bound, **Ours** is not as accurate as **Mask R-CNN (full)** ($\sim\mathbf{9.0}\%$ less precise if we consider the **mAP50 segm(%)**), but is trained $\sim\mathbf{6.9}\times$ faster.

[6]All the *off-line experiments* have been performed on a machine equipped with Intel(R) Xeon(R) W-2295 CPU @ 3.00GHz, and a single NVIDIA Quadro RTX 6000.

[7]Note that, in HO-3D, the annotations for instance segmentation are not provided for the test set. Therefore, we extract training and test sequences from the original HO-3D training set.

[8]We run the module with the proposed method on a machine equipped with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, and a single NVIDIA RTX 2080 Ti.

[9]https://github.com/robotology

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	89.66 ± 0.47	91.26 ± 0.56	84.67 ± 0.81	80.26 ± 0.59	1h 35m 42s
Mask R-CNN (output layers)	84.51 ± 0.40	81.70 ± 0.17	75.81 ± 0.30	70.46 ± 0.24	2h 57m 12s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

TABLE I: Benchmark on the YCB-Video dataset. We compare the proposed approach **Ours** to the baseline **Mask R-CNN (output layers)** and to the upper bound **Mask R-CNN (full)**.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
Mask R-CNN (output layers)	88.05 ± 0.32	86.11 ± 0.29	74.75 ± 0.19	65.04 ± 0.62	1h 50m 33s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

TABLE II: Benchmark on the HO-3D dataset. We report on the performance obtained with **Ours** and we compare it to **Mask R-CNN (output layers)** and **Mask R-CNN (full)** for the analysis in Sec. V-B.

B. Benchmark on HO-3D

We evaluate the proposed approach on the HO-3D dataset. As in Sec. V-A, we compare **Ours** with **Mask R-CNN (output layers)** and we consider **Mask R-CNN (full)** as upper bound. For this experiment, we empirically set the number of the *Minibootstrap* batches of the *On-line RPN* and of the *On-line Detection Module* in **Ours** to 12 and we report the obtained results in Tab. II.

Similarly to the experiment on YCB-Video, **Ours** can be trained $\sim\mathbf{2.3}\times$ and $\sim\mathbf{6.6}\times$ faster than **Mask R-CNN (full)** and **Mask R-CNN (output layers)**, respectively. Models obtained with **Ours** are slightly less precise than those provided by **Mask R-CNN (output layers)** for the task of instance segmentation, while they are $\sim\mathbf{15.3}\%$ less accurate if we consider the **mAP70 bbox(%)**. We will show in Sec. VI-B that this gap can be recovered with a different training protocol (**Ours Serial** in Sec. VI-B). However, **Ours**, achieves the best training time with an accuracy that is close to the state-of-the-art.

VI. FAST REGION PROPOSAL ADAPTATION

In this section, we investigate the impact of region proposal adaptation on the overall performance. In particular, in Sec. VI-A, we show that, with respect to our previous work [1], updating the RPN provides a significant gain in accuracy, maintaining a comparable training time. Then, in Sec. VI-B we analyze the speed/accuracy trade-off achieved with the proposed approximated training protocol.

A. Is Region Proposal Adaptation Key to Performance?

The adaptation of the region proposal on a new task provides a significant gain in accuracy for object detection (in this paper we report some evidence while additional experiments can be found in [2]). In particular, adapting the region proposal is especially effective when FEATURE-TASK and TARGET-TASK present a significant domain shift (which represents a common scenario in robotics). In this section, we show that better region proposals improves also the downstream mask estimation.

For testing performance under domain shift, we consider as FEATURE-TASK the categorization task of the general-purpose dataset MS COCO. Instead, we consider as TARGET-TASKs the identification tasks of the YCB-Video and HO-3D datasets, which depict tabletop and in-hand scenarios, respectively.

We consider **Mask R-CNN (full)** as the upper bound of the experiment, since it updates the entire network on the new task. We compare **Ours** with the method proposed in [1] (Sec. III), namely **O-OS**[10], in which the RPN remains constant during training on the TARGET-TASK. For a fair comparison, we set **O-OS** training hyper-parameters according to Sec. IV-A (i.e., changing the number of Nyström centers of FALKON in the *On-line Detection Module* and in the *On-line Segmentation Module* with respect to [1]).

Results in Tab. III and in Tab. IV show that, as expected, there is an accuracy gap between **Mask R-CNN (full)** and all the other considered methods (**Ours** and **O-OS**). However, notably, the adaptation of the region proposal on the TARGET-TASK in **Ours** allows to significantly reduce the accuracy gap between **Mask R-CNN (full)** and **O-OS**. Moreover, **Ours** outperforms the accuracy of **O-OS** with a comparable training time. For instance, in the HO-3D experiment (see Tab. IV), the segmentation **mAP50** obtained with **Ours** is, on average, $\sim\mathbf{7.1}$ points greater than **O-OS**, with a difference in training time of only *3m 20s*.

B. Approximated On-line Training: Speed/Accuracy Trade-off

In this section, we evaluate the impact of the approximation in **Ours**. To do this, we compare it with a different training protocol (referred to as **Ours Serial**). This relies on the same on-line modules as **Ours**. However, **Ours Serial** performs two steps of feature extraction, one to train the *On-line RPN* and the other for the *On-line Detection Module* and the *On-line Segmentation Module*. This latter is done after region proposal adaptation and allows to use better *RoIs* to train the module for on-line detection, improving the overall performance of the pipeline. In details, **Ours Serial** is composed of the four steps depicted in Fig. 5:

1) Feature extraction for region proposal. This is done to extract F_r (see Sec. III-A) on the images of the TARGET-TASK.

[10]In [1], **O-OS** is referred to as **Ours**.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	89.66 ± 0.47	91.26 ± 0.56	84.67 ± 0.81	80.26 ± 0.59	1h 35m 42s
O-OS	76.15 ± 0.31	74.44 ± 0.11	68.06 ± 0.34	63.90 ± 0.36	11m 14s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

TABLE III: Comparison between **Ours**, **Mask R-CNN (full)** and **O-OS** trained on YCB-Video. For **O-OS**, we reproduce the experiment of Tab. I in [1], but we run the experiment three times (reporting mean and standard deviation of the obtained results) on the hardware used for this work and we set the training hyper-parameters as described in Sec. VI-A.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
O-OS	75.27 ± 0.26	77.42 ± 0.45	57.89 ± 0.24	57.86 ± 0.21	13m 31s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

TABLE IV: We report on the performance obtained on HO-3D with **Ours** and we compare it to **Mask R-CNN (full)** and **O-OS** for the analysis in Sec. VI-A.

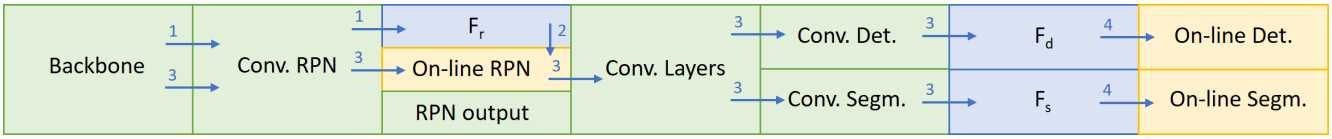


Fig. 5: **Ours Serial** training protocol. We rely on the feature extraction layers of Mask R-CNN pre-trained on the FEATURE-TASK to extract F_r and we train the *On-line RPN* on the TARGET-TASK. Then, we rely on the feature extraction layers of Mask R-CNN and on the *On-line RPN* trained on the TARGET-TASK to extract F_d and F_s. Finally, we train the *On-line Detection Module* and the *On-line Segmentation Module* on the TARGET-TASK. The values on the arrows correspond to the training steps in Sec. VI-B.

2) These features are then used to train the *On-line RPN* on the TARGET-TASK, as described in Sec. III-B.
3) The new *On-line RPN* is used to extract more precise regions and the corresponding features for detection and segmentation (respectively, F_d and F_s).
4) F_d and F_s are used to train the *On-line Detection Module* and the *On-line Segmentation Module* on the TARGET-TASK, as described in Sec. III-B and Sec. III-C, respectively.

We evaluate **Ours** and **Ours Serial** in the same setting used for previous experiments (Sec. V and Sec. VI-A). In **Ours Serial**, we set the Nyström centers of the FALKON classifiers and the batch size *BS* considered in the *Minibootstrap* to train the *On-line RPN* and the *On-line Detection Module* as described in Sec. IV-A. Moreover, we empirically set the number of *Minibootstrap* iterations n_B to 8 and 7 in the experiments on YCB-Video and HO-3D, respectively.

We report results in Tab. V (YCB-Video) and in Tab. VI (HO-3D). Specifically, the first one shows that the accuracy of **Ours** in the YCB-Video experiment is comparable to the one of **Ours Serial**, demonstrating that the approximated training procedure substantially does not affect performance in this case. Instead, in the HO-3D experiment (see Tab. VI), **Ours** is slightly less precise than **Ours Serial** for the task of instance segmentation, while being $\sim\mathbf{11.6}\%$ less accurate if we consider the **mAP70 bbox(%)**. However, **Ours** is trained $\sim\mathbf{1.8}\times$ and $\sim\mathbf{2.2}\times$ faster than **Ours Serial** in the YCB-Video and in the HO-3D experiments, respectively.

Still, with respect to **Mask R-CNN (output layers)**, **Ours Serial** achieves comparable performance, but with training time that is much shorter. However, the approximated training protocol proposed in this paper allows further optimization which is discussed in the next section.

VII. STREAM-BASED INSTANCE SEGMENTATION

We now consider a robotic application, in which the robot is tasked to learn new objects on-line, while automatically acquiring training samples. In this case, training data arrive continuously in stream, and the robot is forced to either use them immediately or store them for later use. We investigate to what extent it is possible to reduce the training time and how this affects segmentation performance.

Because data acquisition takes a considerable amount of time, there is the opportunity to perform, in parallel, some of the processing required for training. In the proposed pipeline, for example, the training protocol **Ours** has been designed to separate feature extraction and the training of the Kernel-based components. In this case, feature extraction can be performed while images and ground-truth labels are received by the robot. In this section, we investigate to what extent this possibility can be exploited also with the conventional Mask R-CNN architecture.

We compare the proposed **Ours** with three different Mask R-CNN baselines. Specifically, we consider **Mask R-CNN (full)** and two variations of **Mask R-CNN (output layers)** as presented in Sec. IV-A.

Because images arrive in a stream, similar views of the same objects are represented in subsequent frames. In App. C we show that a proper training of **Mask R-CNN (full)** and **Mask R-CNN (output layers)** requires that the images are shuffled randomly. This requires storing all

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (output layers)	84.51 ± 0.40	81.70 ± 0.17	75.81 ± 0.30	70.46 ± 0.24	2h 57m 12s
Ours Serial	83.97 ± 0.59	83.00 ± 0.78	75.06 ± 0.88	69.12 ± 0.56	24m 42s
Ours	83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

TABLE V: Comparison between the proposed approach **Ours**, the baseline **Mask R-CNN (output layers)** and **Ours Serial** trained on YCB-Video. Refer to Sec. VI-B for further details.

Method	mAP50 bbox(%)	mAP50 segm(%)	mAP70 bbox(%)	mAP70 segm(%)	Train Time
Mask R-CNN (output layers)	88.05 ± 0.32	86.11 ± 0.29	74.75 ± 0.19	65.04 ± 0.62	1h 50m 33s
Ours Serial	88.70 ± 0.43	87.87 ± 0.37	71.65 ± 0.93	64.76 ± 0.70	37m 18s
Ours	83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

TABLE VI: We report on the performance obtained on HO-3D with **Ours** and we compare it to **Mask R-CNN (output layers)** and **Ours Serial** for the analysis in Sec. VI-B.

images and waiting until the end of the data acquisition process, before starting the training. We hence consider an additional baseline, **Mask R-CNN (store features)**, in which, similarly to **Mask R-CNN (output layers)**, we fine-tune the output layers of the RPN and of the detection and segmentation branches. In this case, however, we compute and store the backbone feature maps for each input image during data acquisition to save time. This can be done because, during the fine-tuning, the weights of the backbone remain unaltered.

Both **Ours** and **Mask R-CNN (store features)** can perform the feature extraction while receiving the stream of images: this allows to further reduce the training time. This is possible because the frame rate for feature extraction in both cases is greater than the frame rate of the stream of incoming data. For instance, with **Ours**, we extract features at $\mathbf{14.7}$ FPS for YCB-Video while the stream of images that is used for training has a frame rate of 3 FPS (note that the dataset has been collected at 30 FPS, but we use one image over ten to avoid data redundancy). This allows to completely absorb the time for feature extraction in the time for data acquisition for both approaches. Since the time required for the data acquisition is the same for the two compared methods, we remove it from the training time computation, therefore comparing only the processing time that follows this phase. This represents the time to wait for a model to be ready in the target robotic application. As explained above, the time required for feature extraction cannot be removed in the case of **Mask R-CNN (full)** and **Mask R-CNN (output layers)**.

We present results for this experiment for YCB-Video and HO-3D in Fig. 6 and in Fig. 7, respectively. Specifically, we compare the performance of the four considered methods for increasing training time. For the Mask R-CNN baselines, we take the accuracy in different moments of the fine-tuning, while, for **Ours**, we increase the number of iterations of the *Minibootstrap* from 2 to 15. In both Fig. 6 and Fig. 7, we report in the first row the *mAP* trends at *IoU* 50% while in the second row we report the results for *IoU* 70%, for both detection and segmentation.

As it can be noticed, **Ours** achieves the best accuracy for short training time. For instance, in the YCB-Video experiment, if we consider a training time of $\sim\mathbf{20s}$, which is the necessary training time if we set the minimum number of *Minibootstrap* iterations n_B=2, **Ours** achieves a **mAP** for instance segmentation of *(i)* $\sim\mathbf{82.2}$ and *(ii)* $\sim\mathbf{67.9}$ for the *IoU* thresholds set to *(i)* 50% and *(ii)* 70%. With a similar optimization time, the Mask R-CNN baselines perform quite poorly. For example, **Mask R-CNN (full)** (which is the best among the baselines) reaches a **mAP** of *(i)* $\sim\mathbf{53.9}$ and *(ii)* $\sim\mathbf{39.8}$ for the *IoU* thresholds set to *(i)* 50% and *(ii)* 70%.

Moreover, the plots show that, for all the experiments, **Mask R-CNN (output layers)** achieves the worst performance, while **Mask R-CNN (store features)** has a steeper slope. This is due to the fact that this method does not perform the forward pass of the Mask R-CNN backbone for feature extraction. On the contrary, **Mask R-CNN (full)** presents a better trend than **Mask R-CNN (output layers)** and **Mask R-CNN (store features)**. This might be due to the following reasons. Firstly, **Mask R-CNN (full)** optimizes more parameters of the network. While requiring more time for each training step, this allows to speed-up the optimization process, requiring less iterations on the dataset to achieve comparable accuracy. Secondly, **Mask R-CNN (full)** performs a warm restart of the the output layers of the RPN, while in the other baselines they are re-initialized from scratch. However, to achieve a similar performance to **Ours**, **Mask R-CNN (full)** requires $\sim\mathbf{75s}$ for the YCB-Video experiment and $\sim\mathbf{50s}$ on HO-3D.

Finally, as it can be noticed, the standard deviations of most of the Mask R-CNN baselines are greater than the ones of **Ours**. This derives from the fact that while **Ours** samples features from all the training images, the Mask R-CNN baselines are optimized only on a subset of them due to time constraints (e.g. in the YCB-Video experiment **Mask R-CNN (full)** processes images at $\sim\mathbf{8.0}$ FPS when trained for $\mathbf{1}$ minute). Reducing the number of training images increases the variability of the results.

In the video attached as supplementary material to the manuscript[11], we show qualitative results to compare **Ours** to **Mask R-CNN (full)** when trained for the same time.

[11] https://youtu.be/eLatoDWY4OI

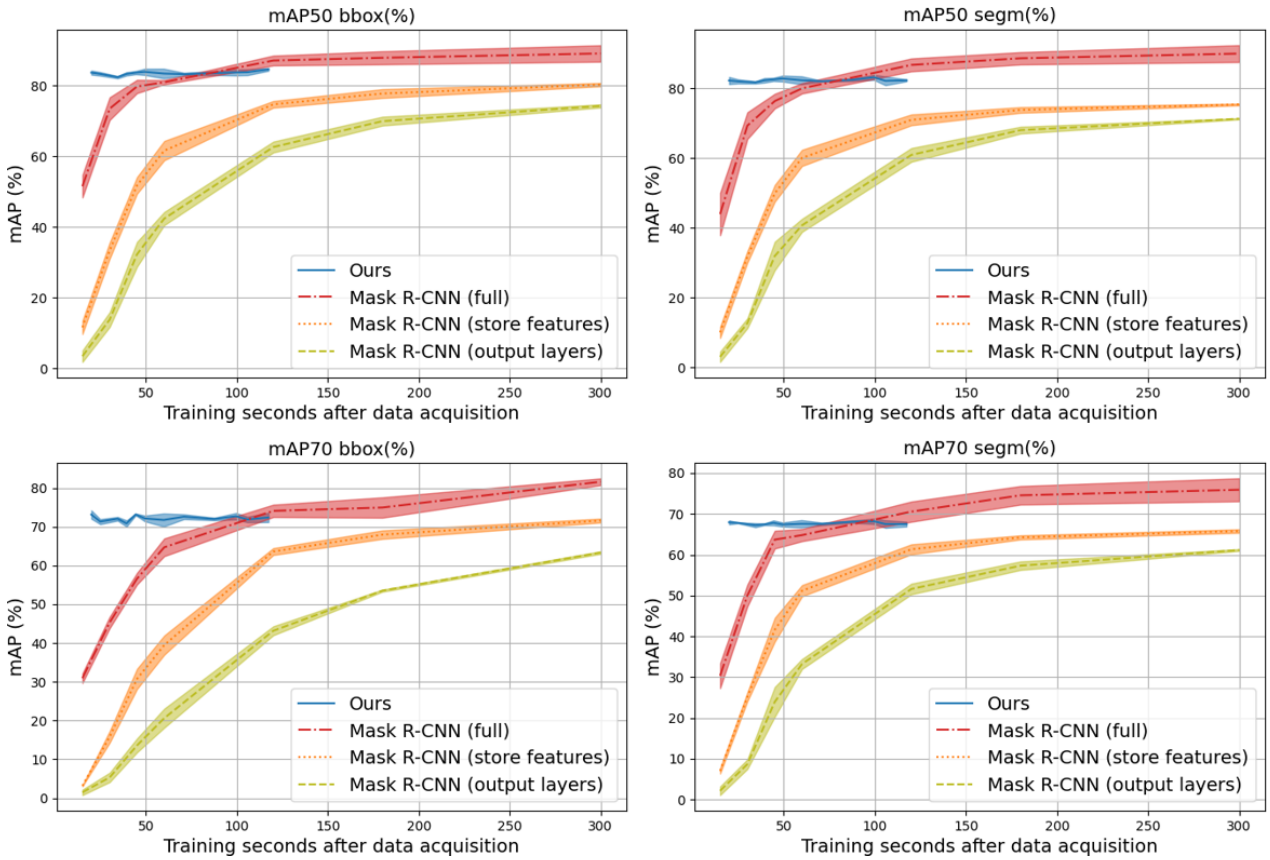


Fig. 6: Detection and segmentation *mAPs* for increasing number of *Minibootstrap* iterations for **Ours** and for increasing training time of the Mask R-CNN baselines, considering YCB-Video as TARGET-TASK. The plots show the average and the standard deviation of the accuracy obtained over three training sessions with the same parameters.

VIII. ROBOTIC APPLICATION

In this section, we describe the pipeline based on the proposed method, that we developed for the iCub [5] robot. We set our application in a teacher-learner scenario, in which the robot learns to segment novel objects shown by a human. The proposed application depicts a similar setting to the experiments on HO-3D showing the effectiveness of the approach to learn new objects also in presence of domain shift.

While in the off-line experiments all the input images and the object instances are fixed beforehand, in the application this information is not known in advance. New objects may appear in the scene and, while learning to segment them, the robot has to keep and integrate the knowledge of the classes that are already known. We therefore propose a strategy to process the incoming images and extract corresponding features such that, for each new class, a detection model is trained with **Ours**, integrating the knowledge of old and new objects. This is done by first training new classifiers on the new classes, considering also the information from the objects already known. Then, the classifiers previously trained on the old classes are updated using features of the new classes.

The proposed application consists of four main modules (the blocks depicted in Fig. 8). It allows to train and update an instance segmentation model by: *(i)* automatically collecting ground-truth for instance segmentation with an interactive pipeline for incoming training images, *(ii)* extracting corresponding features and aggregating them such that the information of old and new objects are integrated in the *Minibootstrap* and *(iii)* updating the *On-line RPN*, the *On-line Detection Module* and the *On-line Segmentation Module*. In the next paragraphs, we provide further details for each of the main blocks.

Human-Robot Interaction (HRI). This block allows the human to give commands to the robot with a module for speech recognition (*Speech Recognition* in Fig. 8), triggering different states of the system. This allows the user to either teach the robot a new object, by presenting and rotating it in front of the camera (*train*) or to perform *inference*, i.e., to segment objects already known in the scene.

Automatic Data Acquisition. When the state of the system is set to *train*, this block extracts a blob of pixels representing the closest object to the robot [56]. This is used as ground-truth annotation for the new object that is presented by the human. This blob is computed by exploiting the depth information to segment the object from the background (*Automatic GT Extractor*). Moreover, in order to enhance the

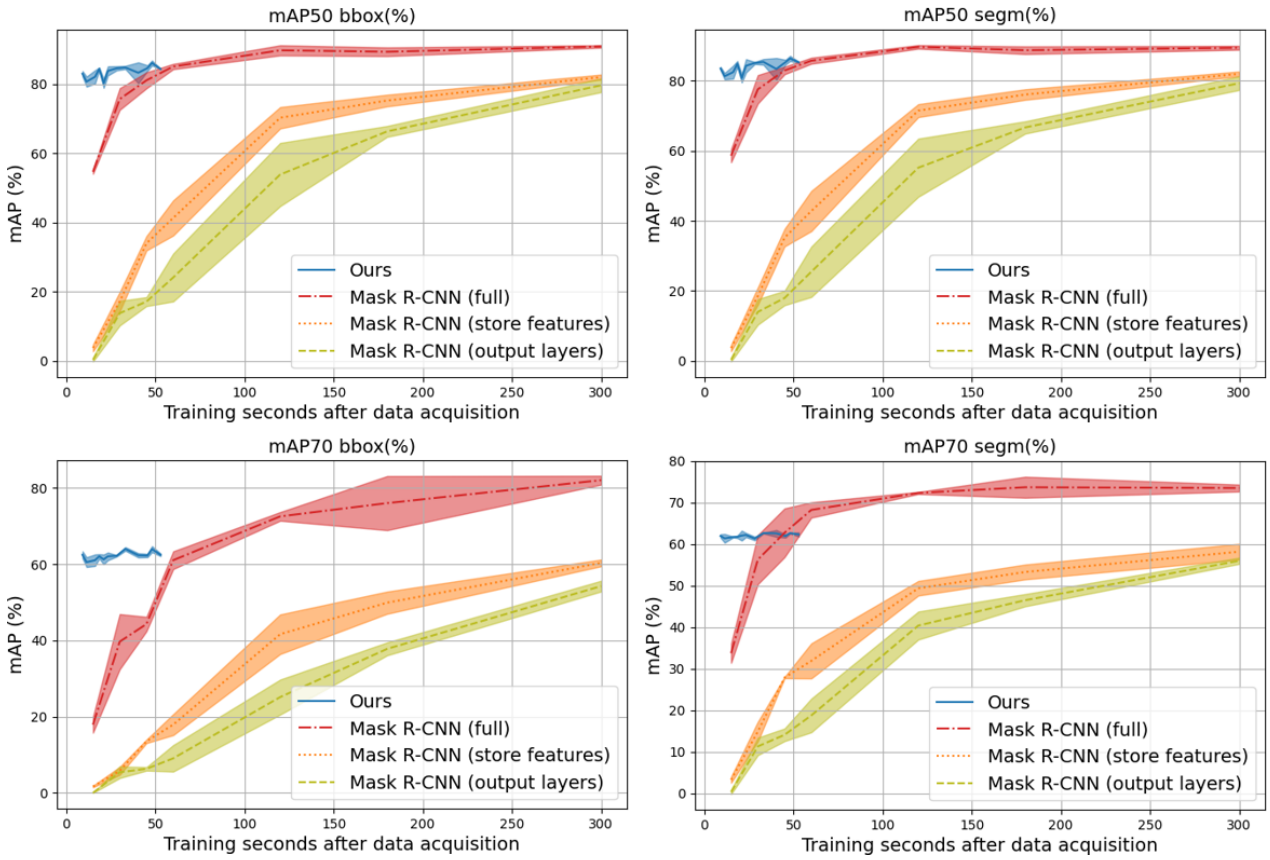


Fig. 7: We consider HO-3D as TARGET-TASK and we report the average and the standard deviation of the *mAPs* over three training sessions with the same parameters for increasing number of *Minibootstrap* iterations for **Ours**, and for increasing training time of **Mask R-CNN (full)**, **Mask R-CNN (output layers)** and **Mask R-CNN (store features)**.

background variability in the training images, the extracted blob is also used by the robot to follow the object with the gaze (*Gaze Controller*). To deal with noise in the depth image, we post-process the masks to ensure spatio-temporal coherence between consecutive frames. Specifically, we consider as valid ground-truth masks those overlapping over a certain threshold with the ones of previous and subsequent frames.

Feature Extraction. It is used to extract the features to train the three on-line modules. It relies on the ground-truth masks provided by the *Automatic GT Extractor* and on the corresponding image collected by the robot. This block implements the *Feature Extraction Module* as described in Sec. III-A, with some modifications introduced to adapt it to the interactive setting of the demonstration. We describe the major differences in Sec. VIII-A.

On-line Segmentation. This block is trained with the proposed approach **Ours** (see Sec. III-D) relying on the features extracted by the *Feature Extraction* block. At inference time, it predicts objects masks on a given image. To do this, similarly to **Ours**, it relies on Mask R-CNN pre-trained on the MS COCO dataset for feature extraction and on the proposed on-line modules as described in Sec. III-A.

A. Incremental Instance Segmentation Learning

When a new object has to be learned, the three on-line modules need to be updated. However, only for the *On-line RPN* and the *On-line Detection Module* specific operations are required to integrate the knowledge of the old classes with the new one and to re-train the two modules with the updated information. Instead, for the *On-line Segmentation Module* only the classifier of the novel class must be trained. This is due to the fact that, for each class, the latter extracts masks labels from the ground-truth bounding boxes for that class, while the other modules use all the images in the dataset (see Sec. III-B and Sec. III-C for details).

To this end, in the following paragraphs we describe how we adapt the feature extraction procedures for the *On-line RPN* and the *On-line Detection Module* reported in Sec. III-B such that past and novel classes can be properly integrated for the training. We refer the reader to App. D for the probabilistic equivalence between the feature sampling procedures of the two on-line modules described in Sec. III-B and the ones presented in this section. Please note that, for each module, we consider training features sampled independently from each training image.

For the sake of simplicity, in the following analysis, we consider a single incremental task scenario where a sequence

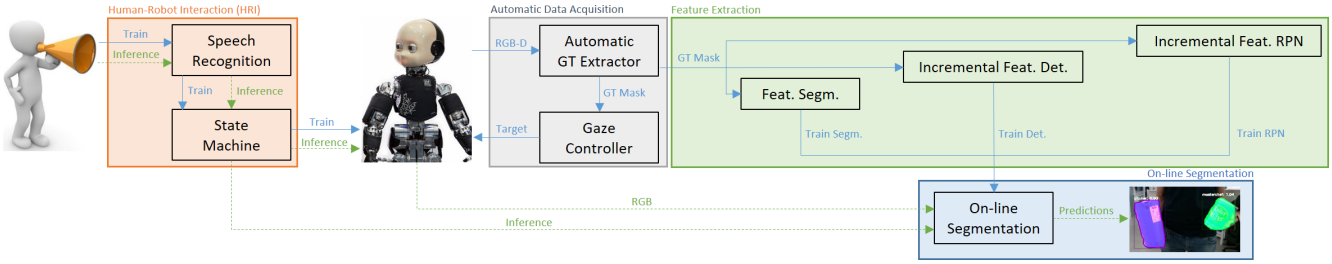


Fig. 8: **Overview of the proposed robotic pipeline** for on-line instance segmentation. At training time (solid arrows), a human teacher shows a new object to the robot, which automatically acquires the ground-truth annotations exploiting the depth information. Then, it extracts the features to train the on-line modules. At inference time (dashed arrows), the robot employs such modules to predict the masks of the images acquired by the camera.

Fig. 9: Predictions on test images from the incremental application deployed on the iCub.

Fig. 10: **Dealing with False Positives**. *Left image*: an unknown object (a *glass*) is misclassified (as a *masterchef*). *Center*: training. The robot is provided with the correct label and a demonstration of the object. *Right*: after training the new object is correctly classified.

of images representing an instance of a new class is shown to the robot, which is required to learn the new object at the end of the demonstration. Specifically, the robot has already been trained on $N-1$ classes and must learn the N^{th} object. However, Alg. 2 and Alg. 3 describe in detail the general procedures, which are suitable also if multiple objects must be learned simultaneously. Once the training features for these two modules have been computed with the described approaches, they can be trained with the same procedure described in Sec. III-B, namely with the steps 2 and 3 of the *Minibootstrap* procedure (see App. E).

Positive samples for the N^{th} object, as defined in Sec. III-B for FALKON classifiers and RLS regressors in the *On-line RPN* and in the *On-line Detection Module*, are taken from the current image stream and are not affected by previous sequences. Therefore, we stick to the method described in Sec. III-B for their extraction. Instead, for the computation of the negative features we design two different procedures that we describe in the following paragraphs.

On-line RPN. When learning the N^{th} class, we first collect features for *On-line RPN* training for that class. We do this by extracting convolutional features from the images of the associated sequence and sub-sample them as described in Sec. III-B. Then, we need to integrate the extracted features with those from the previous $N-1$ classes for each anchor, such that *(i)* the number and size of *Minibootstrap* batches are kept fixed and *(ii)* the number of negative samples per-image is kept balanced. To this end, we randomly remove a fraction of the samples collected for the *Minibootstrap* batches of the previous $N-1$ classes and substitute them with those for the N^{th} class. More details about this procedure can be found in App. F.

On-line Detection. Similarly to what is done for the *On-line RPN*, when the N^{th} class arrives, we extract convolutional features from the images of the associated sequence and sub-sample them as described in Sec. III-B. Then, the extracted features have to be integrated with those from the previous $N-1$ classes. This is done in a twofold way: *(i)* we create the N^{th} dataset to train a classifier for the new object integrating the extracted features with a subset of the

previous $N-1$ ones and *(ii)* we update the $N-1$ datasets for training the previous classes with features from the N^{th}. As for the *On-line RPN*, the number and size of *Minibootstrap* batches are kept fixed and we balance the number of per-image negative samples. More details about this procedure can be found in App. G.

B. Discussion and Qualitative Results

We design the incremental feature extraction procedures for the *On-line RPN* and the *On-line Detection Module* to be analogous to the ones used in the off-line experiments (batch procedures), such that, training the on-line modules with *Minibootstrap* batches obtained with the former, provides comparable models to the ones obtained with the batch procedures (and therefore, comparable accuracy). This is due to the fact that a set of negative samples has the same probability to end up in the *Minibootstrap* batches of a classifier (either of the *On-line RPN* or of the *On-line Detection Module*), either using the batch pipelines as described in Sec. III-B or the incremental procedures presented in the previous section. This is demonstrated in App. D. Specifically, we demonstrate that the per-image negative selection probabilities with the two procedures are equivalent, because each image is considered independently in both cases. This proves their equivalence.

We qualitatively show the effectiveness of the incremental pipeline by deploying it on the iCub robot. We train ten object instances and we report the results of the inference on some exemplar frames in Fig. 9. A video of the complete demonstration, comprising the training of all the considered objects and the inference of the trained models, is attached as supplementary material to the manuscript[11]. In Fig. 10, we show how the proposed incremental approach allows to deal with false positive predictions. Key to achieve this is the re-training of the $N-1$ classifiers for previous classes when the N^{th} object arrives. Indeed, integrating data from the N^{th} object when updating the previous $N-1$ allows to strongly reduce the amount of false predictions at inference time.

IX. CONCLUSIONS

The ability of rapidly adapting their visual system to novel tasks is an important requirement for robots operating in dynamic environments. While state-of-the art approaches for visual tasks mainly focus on boosting performance, a relatively small amount of methods are designed to reduce training time. In this perspective, we presented a novel pipeline for fast training of the instance segmentation task. The proposed approach allows to quickly learn to segment novel objects also in presence of domain shifts. We designed a two-stage hybrid pipeline to operate in the typical robotic scenario where streams of data are acquired by the camera of the robot. Indeed, our pipeline allows to shorten the total training time by extracting a set of convolutional features during the data acquisition and to use them in a second step to rapidly train a set of Kernel-based classifiers.

We benchmarked our results on two robotics datasets, namely YCB-Video and HO-3D. On these datasets, we provided an extensive empirical evaluation of the proposed approach to evaluate different training time/accuracy trade-offs, comparing results against previous work [1] and several Mask R-CNN baselines.

Finally, we demonstrated the application of this work on a real humanoid robot. At this aim, we adapted the fast training pipeline for incremental region proposal adaptation and instance segmentation, showing that the robot is able to learn new objects following a short interactive training session with a human teacher.

APPENDIX A

In Tab. VII and in Tab. VIII, we overview the training protocols and the acronyms for *mAP* evaluation used in this work.

APPENDIX B

In this appendix, we report the sequences considered in the experiments on the HO-3D dataset.

- **Training** sequences: *ABF10*, *ABF11*, *ABF12*, *ABF13*, *BB10*, *BB11*, *BB12*, *BB13*, *GPMF10*, *GPMF11*, *GPMF12*, *GPMF13*, *GSF10*, *GSF11*, *GSF12*, *GSF13*, *MC1*, *MC2*, *MC4*, *MC5*, *MDF10*, *MDF11*, *MDF12*, *MDF13*, *ShSu10*, *ShSu12*, *ShSu13*, *ShSu14*, *SM2*, *SM3*, *SM4*, *SMu1*, *SMu40*, *SMu41*.
- **Validation** sequences: *ABF13*, *BB13*, *GPMF13*, *GSF13*, *MC5*, *MDF13*, *ShSu14*, *SM4*, *SMu41*.
- **Test** sequences: *ABF14*, *BB14*, *GPMF14*, *GSF14*, *MC6*, *MDF14*, *SiS1*, *SM5*, *SMu42*.

APPENDIX C

In the stream-based scenario, data is used as soon as it is received (Sec. VII). In this case, the Mask R-CNN baselines can be trained only for one epoch and their weights can be updated only when a new image is received. Therefore, we compare the performance achieved by **Ours** against the Mask R-CNN baselines, training **Mask R-CNN (output layers)** and **Mask R-CNN (full)** for one epoch and without shuffling the input images.

Result are reported in Tab. IX for YCB-Video and in Tab. X for HO-3D. As it can be noticed, while training the Mask R-CNN baselines for just one epoch allows to achieve a performance similar to the one provided in the benchmarks (Sec. V), shuffling the input images turns out to be crucial. Indeed, in both cases, the accuracy provided by the Mask R-CNN baselines drops when the input images are not shuffled, while **Ours** is not affected by this constraint. Moreover, shuffling is particularly critical for the Mask R-CNN baselines on the HO-3D dataset because training objects are shown subsequently, one by one, as in the target teacher-learner setting (Sec. VIII). Therefore, in the stream-based scenario, such baselines cannot be trained in practice.

Method	Training Protocol	Section
Ours	This is the proposed approach. It is composed of two steps. One for *feature extraction* and one for simultaneous *on-line training* of the proposed methods for region proposal, object detection and mask prediction.	III-C
Ours Serial	It is composed of the same modules as **Ours**, but it differs on the training protocol. This is composed of two steps for *feature extraction*, one for the *on-line training* of the proposed method for region proposal and one to *train* the modules for on-line detection and segmentation.	VI-B
O-OS	This is the approach proposed in [1]. It is composed of two steps. One for *feature extraction* and one for *on-line training* of the modules for object detection and mask prediction.	VI-A
Mask R-CNN (full)	This protocol relies on Mask R-CNN pre-trained on the FEATURE-TASK as a warm-restart to train Mask R-CNN on the TARGET-TASK.	IV-A
Mask R-CNN (output layers)	Starting from the Mask R-CNN weights pre-trained on the FEATURE-TASK, it fine-tunes the output layers of the RPN and of the detection and segmentation branches on the TARGET-TASK.	IV-A
Mask R-CNN (store features)	Similarly to **Mask R-CNN (output layers)**, it fine-tunes the output layers of the RPN and of the detection and segmentation branches. However, since the weights of the backbone remain unaltered, it computes and stores the backbone feature maps for each input image during data acquisition.	VII

TABLE VII: Training protocols overview.

***Intersection over Union (IoU)* with ground-truth**	**Object Detection**	**Instance Segmentation**
50%	mAP50 bbox(%)	mAP50 segm(%)
70%	mAP70 bbox(%)	mAP70 segm(%)

TABLE VIII: Object detection and segmentation metrics taxonomy.

Method		**mAP50 bbox(%)**	**mAP50 segm(%)**	**mAP70 bbox(%)**	**mAP70 segm(%)**	**Train Time**
1 Epoch	Mask R-CNN (full)	89.21 ± 0.77	90.75 ± 0.80	83.27 ± 0.54	78.83 ± 1.70	35m 8s
	Mask R-CNN (output layers)	82.78 ± 0.50	79.38 ± 0.40	75.04 ± 0.97	68.42 ± 0.55	26m 48s
1 Epoch No Shuffling	Mask R-CNN (full)	46.29 ± 1.26	43.93 ± 0.84	28.66 ± 1.89	32.83 ± 1.37	33m 41s
	Mask R-CNN (output layers)	69.63 ± 0.48	66.03 ± 0.68	38.48 ± 2.88	54.70 ± 0.46	26m 39s
Ours		83.66 ± 0.84	83.06 ± 0.92	72.97 ± 1.02	68.11 ± 0.29	13m 53s

TABLE IX: Comparison between the training of **Ours** and the Mask R-CNN baselines trained for one epoch and for one epoch without shuffling the input images, considering YCB-Video as TARGET-TASK.

Method		**mAP50 bbox(%)**	**mAP50 segm(%)**	**mAP70 bbox(%)**	**mAP70 segm(%)**	**Train Time**
1 Epoch	Mask R-CNN (full)	92.21 ± 0.88	90.70 ± 0.17	86.73 ± 0.71	77.25 ± 0.62	38m 38s
	Mask R-CNN (output layers)	86.77 ± 0.68	85.45 ± 0.64	70.57 ± 0.41	63.57 ± 0.43	17m 53s
1 Epoch No Shuffling	Mask R-CNN (full)	6.72 ± 2.35	6.53 ± 2.33	6.24 ± 2.21	6.01 ± 2.18	37m 30s
	Mask R-CNN (output layers)	19.28 ± 2.33	21.01 ± 0.75	11.03 ± 1.70	16.74 ± 1.26	17m 52s
Ours		83.63 ± 1.64	84.50 ± 1.63	63.33 ± 1.65	61.54 ± 0.33	16m 51s

TABLE X: Comparison between the training of **Ours** and the Mask R-CNN baselines trained for one epoch and for one epoch without shuffling the input images, considering HO-3D as TARGET-TASK.

APPENDIX D

In this appendix, we prove the probabilistic equivalence of the two sampling procedures which are used in the *Minibootstrap* and in the incremental feature extraction pipelines for both the *On-line RPN* and for the *On-line Detection Module*.

We consider a pool of tensors S_0 and a set of tensors $\hat{S} \subseteq S_0$. We compute the probability of sampling $\hat{S}$ from S_0 with the two following procedures:

- We sample $|\hat{S}|$ tensors from S_0. We refer to the probability that the sampled tensors are equal to $\hat{S}$ as $P(\hat{S} \sim S_0)$.
- We recursively sample m sets from S_0 and we obtain the pools $S_1, .., S_m$ such that $|S_0| \geq |S_1| \geq ... \geq |S_m| \geq |\hat{S}|$. Namely, for each S_i, with i in $0, ..., m-1$, S_{i+1} is a random sample of size $|S_{i+1}|$ of S_i. Finally, we sample $|\hat{S}|$ tensors from S_m. We refer to the probability that the sampled tensors are equal to $\hat{S}$ as $P(\hat{S} \sim S_m)$.

We note that, for the *On-line RPN* and for the *On-line Detection Module*, the pool of tensors S_0 represents the whole set of features associated to an image. $S_1, .., S_m$, instead, represent consecutive sub-samples of S_0 in the incremental feature extraction pipelines. Finally, $\hat{S}$ correspond to the final per-image set of features chosen for training the on-line modules either with the *Minibootstrap* (as presented in III-B) or with the incremental pipelines.

We prove that $P(\hat{S} \sim S_0)$ is equal to $P(\hat{S} \sim S_m)$.

Proof. $P(\hat{S} \sim S_0)$ can be computed as follows:

$$P(\hat{S} \sim S_0) = \frac{1}{\binom{|S_0|}{|\hat{S}|}} \tag{1}$$

Instead, due to the law of total probability, we can decompose $P(\hat{S} \sim S_m)$ as follows (note that if $\hat{S}$ is not a subset of S_m, $P(\hat{S} \sim S_m | \hat{S} \not\subseteq S_m) = 0$):

$$P(\hat{S} \sim S_m) = P(\hat{S} \sim S_m | \hat{S} \subseteq S_m) \times P(\hat{S} \subseteq S_m) \tag{2}$$

Again, due to the law of total probability, we can decompose $P(\hat{S} \subseteq S_m)$ from equation 2 as follows (note that, for each i in $0, ..., m-1$, $P(\hat{S} \subseteq S_i | \hat{S} \not\subseteq S_{i-1}) = 0$):

$$\begin{aligned} P(\hat{S} \subseteq S_m) &= P(\hat{S} \subseteq S_m | \hat{S} \subseteq S_{m-1}) \times P(\hat{S} \subseteq S_{m-1}) \\ &= \prod_{k=m}^{1} P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) \times P(\hat{S} \subseteq S_0) \end{aligned} \tag{3}$$

Note that $P(\hat{S} \subseteq S_0) = 1$ by definition (i.e., $\hat{S}$ is always in S_0). Therefore:

$$P(\hat{S} \subseteq S_m) = \prod_{k=m}^{1} P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) \tag{4}$$

We note that $\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}$ is the total number of feasible samples s.t. $\hat{S} \subseteq S_k$ given that $\hat{S} \subseteq S_{k-1}$. Namely, we fix $\hat{S}$ in S_k and we compute the number of possible combinations of the remaining $|S_k|-|\hat{S}|$ tensors that can be in S_k sampled from the remaining pool of size $|S_{k-1}| - |\hat{S}|$. Therefore:

$$P(\hat{S} \subseteq S_k | \hat{S} \subseteq S_{k-1}) = \frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} \tag{5}$$

We can decompose the component in the product as follows:

$$\frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} = \frac{(|S_{k-1}| - |\hat{S}|)!}{(|S_k| - |\hat{S}|)!} \times \frac{|S_k|!}{|S_{k-1}|!} \tag{6}$$

Note that, for each of these elements with $k \neq 1$ and $k \neq m$, if we multiply it by the element at $k-1$ and by the element at $k+1$, all the elements are simplified. Therefore, we can rewrite $P(\hat{S} \sim S_m)$ from equation 2 as:

$$\begin{aligned} P(\hat{S} \sim S_m) &= P(\hat{S} \sim S_m | \hat{S} \subseteq S_m) \times P(\hat{S} \subseteq S_m) \\ &= \frac{1}{\binom{|S_m|}{|\hat{S}|}} \times \prod_{k=m}^{1} \frac{\binom{|S_{k-1}|-|\hat{S}|}{|S_k|-|\hat{S}|}}{\binom{|S_{k-1}|}{|S_k|}} \\ &= \frac{(|S_m| - |\hat{S}|)! \times |\hat{S}|!}{|S_m|!} \times \frac{|S_m|! \times (|S_0| - |\hat{S}|)!}{(|S_m| - |\hat{S}|)! \times S_0!} \\ &= \frac{|\hat{S}|! \times (|S_0| - |\hat{S}|)!}{|S_0|!} \\ &= \frac{1}{\binom{|S_0|}{|\hat{S}|}} \end{aligned} \tag{7}$$

This concludes our proof, since:

$$P(\hat{S} \sim S_0) = P(\hat{S} \sim S_m) \tag{8}$$

□

APPENDIX E

In Alg. 1, we report the pseudo-code of the *Minibootstrap* [48] procedure. Note that, we use the Sample(*Set, Sample size*) function to extract *Sample size* random tensors from the given *Set*. We will use this function also in Alg. 2 and Alg. 3.

APPENDIX F

We report the pseudo-code for the incremental feature extraction pipeline for the *On-line RPN*. Since the procedure is equal for all the considered anchors, in Alg. 2 we report the algorithm for a generic anchor a.

APPENDIX G

In Alg. 3, we report the pseudo-code for the incremental feature extraction pipeline for the *On-line Detection Module*.

ACKNOWLEDGMENT

This research received support by the ERA-NET CHIST-ERA call 2017 project HEAP. It is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. L. R. acknowledges the financial support of the European Research Council (grant SLING 819789), the AFOSR projects FA9550-18-1-7009, FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), and the EU H2020-MSCA-RISE project NoMADS - DLV-777826.

REFERENCES

[1] Federico Ceola, Elisa Maiettini, Giulia Pasquale, Lorenzo Rosasco, and Lorenzo Natale. Fast object segmentation learning with kernel-based methods for robotics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13581–13588, 2021.

[2] Federico Ceola, Elisa Maiettini, Giulia Pasquale, Lorenzo Rosasco, and Lorenzo Natale. Fast region proposal learning for object detection for robotics. *arXiv preprint arXiv:2011.12790*, 2020.

[3] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.

[4] Shreyas Hampali, Mahdi Rad, Markus Oberweger, and Vincent Lepetit. HOnnotate: A method for 3D annotation of hand and object poses. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3196–3206, 2020.

[5] Giorgio Metta, Lorenzo Natale, Francesco Nori, Giulio Sandini, David Vernon, Luciano Fadiga, Claes von Hofsten, Kerstin Rosander, Manuel Lopes, José Santos-Victor, Alexandre Bernardino, and Luis Montesano. The iCub humanoid robot: an open-systems platform for research in cognitive development. *Neural networks : the official journal of the International Neural Network Society*, 23(8-9):1125–34, 1 2010.

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Neural Information Processing Systems (NIPS)*, 2015.

[7] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016.

[8] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

Algorithm 1 Minibootstrap Pseudo-code for the *Minibootstrap* in the off-line experiments. See [48] for further details.

Input: BS, n_B: size and number of *Minibootstrap* batches
I: set of training images
N: number of classes
Output: M: N trained classifiers
***Stage 1**: Feature extraction*
for $n = 1$ to N **do** ▷ *Initialize N empty sets of per class **positives** and **negatives** features*
$Pos[n] \leftarrow \emptyset$; $Neg[n] \leftarrow \emptyset$
end for
for $i = 1$ to $|I|$ **do**
for $n = 1$ to N **do**
if ($I[i]$ has positives of class n $Pos_{i,n}$) **then**
$Pos[n] \leftarrow Pos[n] \bigcup Pos_{i,n}$ ▷ ***Add positives** for class n from the i^{th} image*
end if
$Neg[n] \leftarrow Neg[n] \bigcup \text{Sample}(Neg_{i,n}, \lceil \frac{n_B \times BS}{|I|} \rceil)$ ▷ ***Add negatives** for class n from the i^{th} image*
end for
end for
***Stage 2**: Shuffling and batches creation*
for $n = 1$ to N **do**
$Neg[n] \leftarrow \text{Sample}(Neg[n], n_B \times BS)$
$Neg[n] \leftarrow \text{Split}(Neg[n], n_B, BS)$ ▷ ***Split** $Neg[n]$ in n_B batches of size BS*
end for
***Stage 3**: Classifiers training*
for $n = 1$ to N **do**
$F[n] \leftarrow Pos[n] \bigcup Neg[n][1]$
$M[n] \leftarrow \text{TrainClassifier}(F[n])$ ▷ ***Train** classifier using the first batch*
$Neg_{chosen}[n] \leftarrow \text{PruneEasy}(M[n], Neg[n][1])$ ▷ ***Prune easy negatives** from $Neg[n][1]$ using $M[n]$*
for $j = 2$ to n_B **do**
$N^H \leftarrow \text{SelectHard}(M[n], Neg[n][j])$ ▷ ***Select hard negatives** from $Neg[n][j]$ using $M[n]$*
$F[n] \leftarrow Pos[n] \bigcup Neg_{chosen}[n] \bigcup N^H$ ▷ ***Add hard negatives** from $Neg[n][j]$ to the training set*
$M[n] \leftarrow \text{TrainClassifier}(F[n])$ ▷ ***Train** classifier using the new dataset*
$Neg_{chosen}[n] \leftarrow Neg_{chosen}[n] \bigcup N^H$ ▷ ***Update** the chosen **negatives***
$Neg_{chosen}[n] \leftarrow \text{PruneEasy}(M[n], Neg_{chosen}[n])$ ▷ ***Prune easy negatives** from $Neg_{chosen}[n]$ using $M[n]$*
end for
end for
Return M ▷ ***Return** the final classifiers*

[9] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[10] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[11] Zhaojin Huang, Lichao Huang, Yongchao Gong, Chang Huang, and Xinggang Wang. Mask Scoring R-CNN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6409–6418, 2019.

[12] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[13] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. YOLACT: Real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9157–9166, 2019.

[14] Hao Chen, Kunyang Sun, Zhi Tian, Chunhua Shen, Yongming Huang, and Youliang Yan. BlendMask: Top-down meets bottom-up for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8573–8581, 2020.

[15] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007, 2017.

[16] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9627–9636, 2019.

[17] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[18] Sida Peng, Wen Jiang, Huaijin Pi, Xiuli Li, Hujun Bao, and Xiaowei Zhou. Deep Snake for real-time instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8533–8542, 2020.

[19] Naiyu Gao, Yanhu Shan, Yupei Wang, Xin Zhao, Yinan Yu, Ming Yang, and Kaiqi Huang. SSAP: Single-shot instance segmentation with affinity pyramid. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 642–651, 2019.

[20] Alexander Kirillov, Evgeny Levinkov, Bjoern Andres, Bogdan Savchynskyy, and Carsten Rother. InstanceCut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5008–5017, 2017.

[21] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5221–5229, 2017.

[22] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *European conference on*

Algorithm 2 Incremental On-line RPN Pseudo-code of the *incremental* feature extraction procedure for the *On-line RPN*.

Input: BS, n_B: size and number of *Minibootstrap* batches
Pos^{t-1}, Neg^{t-1}: **positive** and per-image **negative** features at iteration $t-1$ for a generic anchor a
I^t: t^{th} sequence of training images
$\#IMG^t = \#IMG^{t-1} + |I^t|$: total number of training images seen in the previous iterations and in this sequence
Output: Pos^t, Neg^t: **positive** and per-image **negative** features at iteration t for anchor a
Stage 1*: Sample negative features from the ones at iteration* $t-1$
for $j = 1$ to $\#IMG^{t-1}$ **do**
$Neg^t[j] \leftarrow \text{Sample}(Neg^{t-1}[j], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
end for
Stage 2*: Append new features from sequence* I^t
$Pos^t \leftarrow Pos^{t-1}$
for $i = 1$ to $|I^t|$ **do**
if $I^t[i]$ has positives for anchor a $Pos_{I^t[i]}$ **then**
$Pos^t \leftarrow Pos^t \bigcup Pos_{I^t[i]}$ ▷ ***Add positives*** *for anchor* a *from the image* $I^t[i]$
end if
$Neg^t \leftarrow Neg^t \bigcup \text{Sample}(Neg_{I^t[i]}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add negatives*** *for anchor* a *from the image* $I^t[i]$
end for
Return Pos^t, Neg^t ▷ ***Return*** *positive and negative features at iteration* t

computer vision, pages 75–91. Springer, 2016.

[23] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer, 2016.

[24] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollár. TensorMask: A foundation for dense object segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2061–2069, 2019.

[25] Xin Shu, Chang Liu, Tong Li, Chunkai Wang, and Cheng Chi. A self-supervised learning manipulator grasping approach based on instance segmentation. *IEEE Access*, 6:65055–65064, 2018.

[26] Kentaro Wada, Kei Okada, and Masayuki Inaba. Joint learning of instance and semantic segmentation for robotic pick-and-place with heavy occlusions in clutter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9558–9564. IEEE, 2019.

[27] Andrew Li, Michael Danielczuk, and Ken Goldberg. One-shot shape-based amodal-to-modal instance segmentation. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1375–1382. IEEE, 2020.

[28] Siyi Li, Jiaji Zhou, Zhenzhong Jia, Dit-Yan Yeung, and Matthew T. Mason. Learning accurate objectness instance segmentation from photorealistic rendering for robotic manipulation. In Jing Xiao, Torsten Kröger, and Oussama Khatib, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 245–255, Cham, 2020. Springer International Publishing.

[29] Michael Danielczuk, Matthew Matl, Saurabh Gupta, Andrew Li, Andrew Lee, Jeffrey Mahler, and Ken Goldberg. Segmenting unknown 3d objects from real depth images using Mask R-CNN trained on synthetic data. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7283–7290. IEEE, 2019.

[30] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on robot learning*, pages 1369–1378. PMLR, 2020.

[31] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 2021.

[32] Weicheng Kuo, Anelia Angelova, Jitendra Malik, and Tsung-Yi Lin. ShapeMask: Learning to segment novel objects by refining shape priors. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9207–9216, 2019.

[33] Deepak Pathak, Yide Shentu, Dian Chen, Pulkit Agrawal, Trevor Darrell, Sergey Levine, and Jitendra Malik. Learning instance segmentation by interaction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2042–2045, 2018.

[34] Andreas Eitel, Nico Hauff, and Wolfram Burgard. Self-supervised transfer learning for instance segmentation through physical interaction. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4020–4026. IEEE, 2019.

[35] B. Zhang P. Zhang, L. Hu and P. Pan. Spatial constrained memory network for semi-supervised video object segmentation. *The 2020 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*, 2020.

[36] V. Goel S. Garg and S. Kumar. Unsupervised video object segmentation using online mask selection and space-time memory networks. *The 2020 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*, 2020.

[37] Paul Voigtlaender and Bastian Leibe. Online adaptation of convolutional neural networks for video object segmentation. *arXiv preprint arXiv:1706.09364*, 2017.

[38] Mennatullah Siam, Chen Jiang, Steven Lu, Laura Petrich, Mahmoud Gamal, Mohamed Elhoseiny, and Martin Jagersand. Video object segmentation using teacher-student adaptation in a human robot interaction (hri) setting. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 50–56. IEEE, 2019.

[39] Raffaello Camoriano, Giulia Pasquale, Carlo Ciliberto, Lorenzo Natale, Lorenzo Rosasco, and Giorgio Metta. Incremental robot learning of new objects with fixed update time. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3214, 2017.

[40] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019.

[41] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE international conference on computer vision*, pages 3400–3409, 2017.

[42] Juan-Manuel Perez-Rua, Xiatian Zhu, Timothy M Hospedales, and Tao Xiang. Incremental few-shot object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13846–13855, 2020.

[43] Umberto Michieli and Pietro Zanuttigh. Incremental learning techniques for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[44] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. FALKON: An optimal large scale kernel method. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3888–3898. Curran Associates, Inc., 2017.

[45] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro

Algorithm 3 Incremental On-line Detection *Incremental* feature extraction pseudo-code for the *On-line Detection Module*.

Input: BS, n_B: size and number of the *Minibootstrap* batches
N^t: number of classes. It comprises the N^{t-1} classes known at iteration $t-1$ and the novel classes at iteration t
Pos^{t-1}, Img^{t-1}_{Neg}: sets of N^{t-1} (one for each class) **positive** and per-image **negative** features at iteration $t-1$
Img^{t-1}_{Buf}: set of per-image **buffer** features at iteration $t-1$
I^t: t^{th} sequence of training images
$\#IMG^t = \#IMG^{t-1} + |I^t|$: total number of training images in the previous iterations and in this sequence
Output: Pos^t, Neg^t: N^t sets of **positive** and **negative** training features at iteration t

***Stage 1**: Sample negative per-image features from the previous sequences*
for $i = 1$ to $\#IMG^{t-1}$ **do**
  for $n = 1$ to N^t **do**
    if $n \leq N^{t-1}$ **then**
      $Img^t_{Neg}[n][i] \leftarrow$ Sample$(Img^{t-1}_{Neg}[n][i], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
    else
      $Img^t_{Neg}[n][i] \leftarrow \emptyset$ ▷ *Set per-image negatives of the old sequences to an **empty set** for the **new classes***
    end if
  end for
  $Img^t_{Buf}[i] \leftarrow$ Sample$(Img^{t-1}_{Buf}[i], \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$
end for
$Pos^t \leftarrow Pos^{t-1} \bigcup \bigcup_{i=0}^{N^t - N^{t-1}} \emptyset$ ▷ ***Compute** Pos^t **from** Pos^{t-1} adding an **empty set** for each **new class***

***Stage 2**: Sample features from the t^{th} sequence of images*
for $i = 1$ to $|I^t|$ **do**
  for $n = 1$ to N^t **do**
    if $I^t[i]$ has positives of class n $Pos_{I^t[i],n}$ **then**
      $Pos^t[n] \leftarrow Pos^t[n] \bigcup Pos_{I^t[i],n}$ ▷ ***Add positives** for class n from $I^t[i]$*
      $Img^t_{Neg}[n] \leftarrow Img^t_{Neg}[n] \bigcup$ Sample$(Neg_{I^t[i],n}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add** per-image **negatives** for class n*
    else
      $Img^t_{Neg}[n] \leftarrow Img^t_{Neg}[n] \bigcup \emptyset$
    end if
  end for
  $Img^t_{Buf} \leftarrow Img^t_{Buf} \bigcup$ Sample$(All_feat_{I^t[i]}, \lceil \frac{n_B \times BS}{\#IMG^t} \rceil)$ ▷ ***Add** per-image **buffer negatives** from $I^t[i]$*
end for

***Stage 3**: Fill **negative** batches*
for $n = 1$ to N^t **do**
  $Neg^t[n] \leftarrow \emptyset$
  for $i = 1$ to $\#IMG^t$ **do**
    if $(Img_{Neg}[n][i] \neq \emptyset)$ **then** ▷ *If any, add to the **negatives** of class n, per-image **negatives** for class n*
      $Neg^t[n] \leftarrow Neg^t[n] \bigcup Img^t_{Neg}[n][i]$
    else ▷ *Otherwise, add to the **negatives** of class n, per-image **buffer negatives***
      $Neg^t[n] \leftarrow Neg^t[n] \bigcup Img^t_{Buf}[i]$
    end if
  end for
end for
Return Pos^t, Neg^t ▷ ***Return** positive and negative features at iteration t*

Rudi. Kernel methods through the roof: handling billions of points efficiently. *arXiv preprint arXiv:2006.10350v1*, 2020.

[46] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[47] E. Maiettini, G. Pasquale, L. Rosasco, and L. Natale. Speeding-up object detection training for robotics with FALKON. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018.

[48] Elisa Maiettini, Giulia Pasquale, Lorenzo Rosasco, and Lorenzo Natale. On-line object detection: a robotics challenge. *Autonomous Robots*, Nov 2019.

[49] Kah Kay Sung. *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996. AAI0800657.

[50] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[52] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal visual object classes (VOC) challenge.

International Journal of Computer Vision, 88(2):303–338, June 2010.
[53] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, Zürich, 2014. Oral.
[54] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The YCB object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*, pages 510–517. IEEE, 2015.
[55] G. Metta, P. Fitzpatrick, and L. Natale. YARP: Yet another robot platform. *International Journal of Advanced Robotics Systems*, 3(1):43–48, 2006.
[56] Giulia Pasquale, Tanis Mar, Carlo Ciliberto, Lorenzo Rosasco, and Lorenzo Natale. Enabling depth-driven visual attention on the iCub humanoid robot: Instructions for use and new perspectives. *Frontiers in Robotics and AI*, 3:35, 2016.

Federico Ceola is a Ph.D. student at the Humanoid Sensing and Perception (HSP) group at the Istituto Italiano di Tecnologia and at the Laboratory for Computational and Statistical Learning (LCSL) at the University of Genova. He received his Bachelor's Degree in Information Engineering and his Master's Degree (with honors) in Computer Engineering at the University of Padova in 2016 and in 2019, respectively. His research interests lie in the intersection of Robotics, Computer Vision and Machine Learning.

Elisa Maiettini is a Post Doctoral researcher in the Humanoid Sensing and Perception (HSP) research line. She graduated in Software and Electronics Engineering at the University of Perugia, Italy, in 2013 and she obtained an M.D. with honors in Software and Automation Engineering at the same university in 2016. She received the Ph. D. in Bioengineering and Robotics from 2016 to 2020, at the Istituto Italiano di Tecnologia. The main fields of her research are: computer vision, machine learning and humanoid robotics.

Giulia Pasquale is Senior Technician at the Istituto Italiano di Tecnologia, Humanoid Sensing and Perception (IIT HSP) research line. She received her B.Sc. degree in Biomedical Engineering (with honors) and M.Sc. Degree in Bioengineering (with honors), in 2010 and 2013, at the University of Genoa. In 2013 she was research fellow at the National Research Council of Italy. She then obtained a Ph.D. in Bioengineering and Robotics within a collaboration between IIT and the University of Genoa from 2014 to 2017. She was later Postdoctoral Researcher at IIT HSP until 2019. Her research focuses on the development of visual object recognition and localization systems for robotic platforms, standing at the intersection between machine/deep learning, computer vision and robotics.

Giacomo Meanti is a second-year Ph.D. student in Computer Science. After his BSc at the University of Southampton, he received a MSc degree from ETH Zurich in 2018. He worked as a modelling expert at a startup in Switzerland, and is now working on his PhD at the University of Genova under the supervision of Lorenzo Rosasco.

Lorenzo Rosasco is full professor at the University of Genova, Italy. He is also affiliated with the Massachusetts Institute of Technology (MIT), where is a visiting professor, and with the Istituto Italiano di Tecnologia (IIT), where he is an external collaborator. He is leading the efforts to establish the Laboratory for Computational and Statistical Learning (LCSL), born from a collaborative agreement between IIT and MIT. He received his PhD from the University of Genova in 2006. Between 2006 and 2009 he was a postdoctoral fellow at Center for Biological and Computational Learning (CBCL) at MIT working with Tomaso Poggio. His research focuses on studying theory and algorithms for machine learning. He has developed and analyzed methods to learn from small as well as large samples of high dimensional data, using analytical and probabilistic tools, within a multidisciplinary approach drawing concepts and techniques primarily from computer science but also from statistics, engineering and applied mathematics.

Lorenzo Natale has a degree in Electronic Engineering and Ph.D. in Robotics. He was later postdoctoral researcher at the MIT Computer Science and Artificial Intelligence Laboratory. At the moment he is seniore researcher in the Istituto Italiano di Tecnologia, Genova, Italy, where he leads the Humanoid Sensing and Perception group. His research interests include artificial perception and software architectures for robotics. Dr. Natale served as the Program Chair of ICDL-Epirob 2014 and HAI 2017. He is Specialty Chief Editor for the Humanoid Robotics Section of Frontiers in Robotics and AI, associate editor for IEEE-Transactions on Robotics. He is also Ellis Fellow and Core Faculty of the Ellis Genoa Unit.

K-Planes: Explicit Radiance Fields in Space, Time, and Appearance

Sara Fridovich-Keil*
UC Berkeley
sfk@berkeley.edu

Giacomo Meanti*
Istituto Italiano di Tecnologia
giacomo.meanti@iit.it

Frederik Rahbæk Warburg
Technical University of Denmark
frwa@dtu.dk

Benjamin Recht
UC Berkeley
brecht@berkeley.edu

Angjoo Kanazawa
UC Berkeley
kanazawa@berkeley.edu

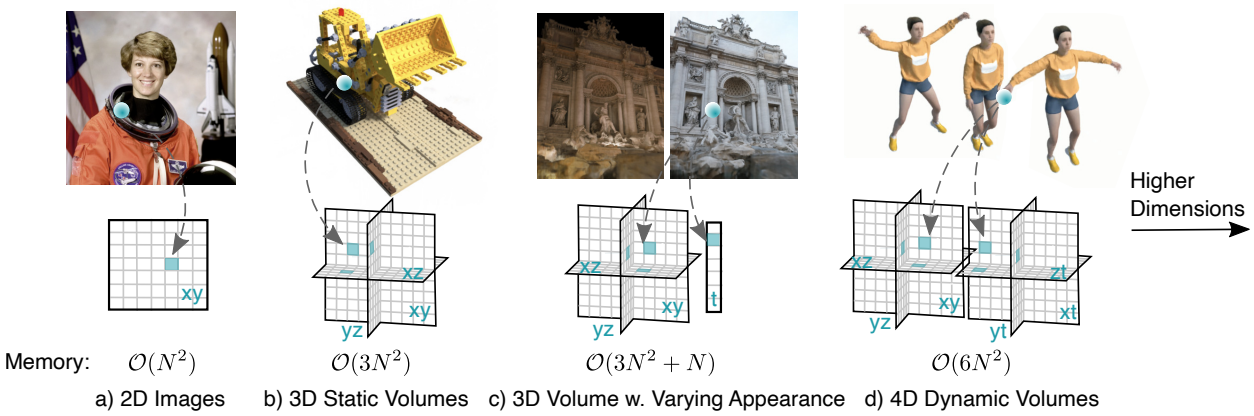


Figure 1. **Planar factorization of d-dimensional spaces.** We propose a simple planar factorization for volumetric rendering that naturally extends to arbitrary-dimensional spaces, and that scales gracefully with dimension in both optimization time and model size. We show the advantages of our approach on 3D static volumes, 3D photo collections with varying appearances, and 4D dynamic videos.

Abstract

We introduce k-planes, a white-box model for radiance fields in arbitrary dimensions. Our model uses $\binom{d}{2}$ ("d-choose-2") planes to represent a d-dimensional scene, providing a seamless way to go from static ($d = 3$) to dynamic ($d = 4$) scenes. This planar factorization makes adding dimension-specific priors easy, e.g. temporal smoothness and multi-resolution spatial structure, and induces a natural decomposition of static and dynamic components of a scene. We use a linear feature decoder with a learned color basis that yields similar performance as a nonlinear black-box MLP decoder. Across a range of synthetic and real, static and dynamic, fixed and varying appearance scenes, k-planes yields competitive and often state-of-the-art reconstruction fidelity with low memory usage, achieving 1000× *compression over a full 4D grid, and fast optimization with a pure PyTorch implementation. For video results and code, please see* https://sarafridov.github.io/K-Planes.

* equal contribution

1. Introduction

Recent interest in dynamic radiance fields demands representations of 4D volumes. However, storing a 4D volume directly is prohibitively expensive due to the curse of dimensionality. Several approaches have been proposed to factorize 3D volumes for static radiance fields, but these do not easily extend to higher dimensional volumes.

We propose a factorization of 4D volumes that is simple, interpretable, compact, and yields fast training and rendering. Specifically, we use six planes to represent a 4D volume, where the first three represent space and the last three represent space-time changes, as illustrated in Fig. 1(d). This decomposition of space and space-time makes our model interpretable, *i.e.* dynamic objects are clearly visible in the space-time planes, whereas static objects only appear in the space planes. This interpretability enables dimension-specific priors in time and space.

More generally, our approach yields a straightforward, prescriptive way to select a factorization of any dimension

with 2D planes. For a d-dimensional space, we use $k = \binom{d}{2}$ ("d-*choose*-2") *k-planes*, which represent every pair of dimensions — for example, our model uses $\binom{4}{2} = 6$ *hex-planes* in 4D and reduces to $\binom{3}{2} = 3$ *tri-planes* in 3D. Choosing any other set of planes would entail either using more than k planes and thus occupying unnecessary memory, or using fewer planes and thereby forfeiting the ability to represent some potential interaction between two of the d dimensions. We call our model k-planes; Fig. 1 illustrates its natural application to both static and dynamic scenes.

Most radiance field models entail some black-box components with their use of MLPs. Instead, we seek a simple model whose functioning can be inspected and understood. We find two design choices to be fundamental in allowing k-planes to be a white-box model while maintaining reconstruction quality competitive with or better than previous black-box models [16, 30]: (1) Features from our k-planes are *multiplied* together rather than added, as was done in prior work [5, 6], and (2) our linear feature decoder uses a learned basis for view-dependent color, enabling greater adaptivity including the ability to model scenes with variable appearance. We show that an MLP decoder can be replaced with this linear feature decoder only when the planes are multiplied, suggesting that the former is involved in both view-dependent color and determining spatial structure.

Our factorization of 4D volumes into 2D planes leads to a high compression level without relying on MLPs, using 200 MB to represent a 4D volume whose direct representation at the same resolution would require more than 300 GB, a compression rate of three orders of magnitude. Furthermore, despite not using any custom CUDA kernels, k-planes trains orders of magnitude faster than prior implicit models and on par with concurrent hybrid models.

In summary, we present the first white-box, interpretable model capable of representing radiance fields in arbitrary dimensions, including static scenes, dynamic scenes, and scenes with variable appearance. Our k-planes model achieves competitive performance across reconstruction quality, model size, and optimization time across these varied tasks, without any custom CUDA kernels.

2. Related Work

K-planes is an interpretable, explicit model applicable to static scenes, scenes with varying appearances, and dynamic scenes, with compact model size and fast optimization time. Our model is the first to yield all of these attributes, as illustrated in Tab. 1. We further highlight that k-planes satisfies this in a simple framework that naturally extends to arbitrary dimensions.

Spatial decomposition. NeRF [24] proposed a fully implicit model with a large neural network queried many times during optimization, making it slow and essentially a black-box. Several works have used geometric representations to reduce the optimization time. Plenoxels [10] proposed a fully explicit model with trilinear interpolation in a 3D grid, which reduced the optimization time from hours to a few minutes. However, their explicit grid representation of 3D volumes, and that of DVGO [33], grows exponentially with dimension, making it challenging to scale to high resolution and completely intractable for 4D dynamic volumes.

Hybrid methods [6, 25, 33] retain some explicit geometric structure, often compressed by a spatial decomposition, alongside a small MLP feature decoder. Instant-NGP [25] proposed a multiresolution voxel grid encoded implicitly via a hash function, allowing fast optimization and rendering with a compact model. TensoRF [6] achieved similar model compression and speed by replacing the voxel grid with a tensor decomposition into planes and vectors. In a generative setting, EG3D [5] proposed a similar spatial decomposition into three planes, whose values are added together to represent a 3D volume.

Our work is inspired by the explicit modeling of Plenoxels as well as these spatial decompositions, particularly the triplane model of [5], the tensor decomposition of [6], and the multiscale grid model of [25]. We also draw inspiration from Extreme MRI [26], which uses a multiscale low-

	Static	Appearance	Dynamic	Fast	Compact	Explicit
NeRF	✓	✗	✗	✗	✓	✗
NeRF-W	✓	✓	✗	✗	✓	✗
DVGO	✓	✗	✗	✓	✗	✗
Plenoxels	✓	✗	✗	✓	✗	✓
Instant-NGP, TensoRF	✓	✗	✗	✓	✓	✗[1]
DyNeRF, D-NeRF	–	✗	✓	✗	✓	✗
TiNeuVox, Tensor4D	–	✗	✓	✓	✓	✗
MixVoxels, V4D	–	✗	✓	✓	✗	✗
NeRFPlayer	–	✗	✓	✓	✓[2]	✗
K-planes hybrid (Ours)	✓	✓	✓	✓	✓	✗
K-planes explicit (Ours)	✓	✓	✓	✓	✓	✓

[1] TensoRF offers both hybrid and explicit versions, with a small quality gap [2] NerfPlayer offers models at different sizes, the smallest of which has $<$ 100 million parameters but the largest of which has $>$ 300 million parameters

Table 1. **Related work overview.** The k-planes model works for a diverse set of scenes and tasks (static, varying appearance, and dynamic). It has a low memory usage (compact) and fast training and inference time (fast). Here "fast" includes any model that can optimize within a few ($<$ 6) hours on a single GPU, and "compact" denotes models that use less than roughly 100 million parameters. "Explicit" denotes white-box models that do not rely on MLPs.

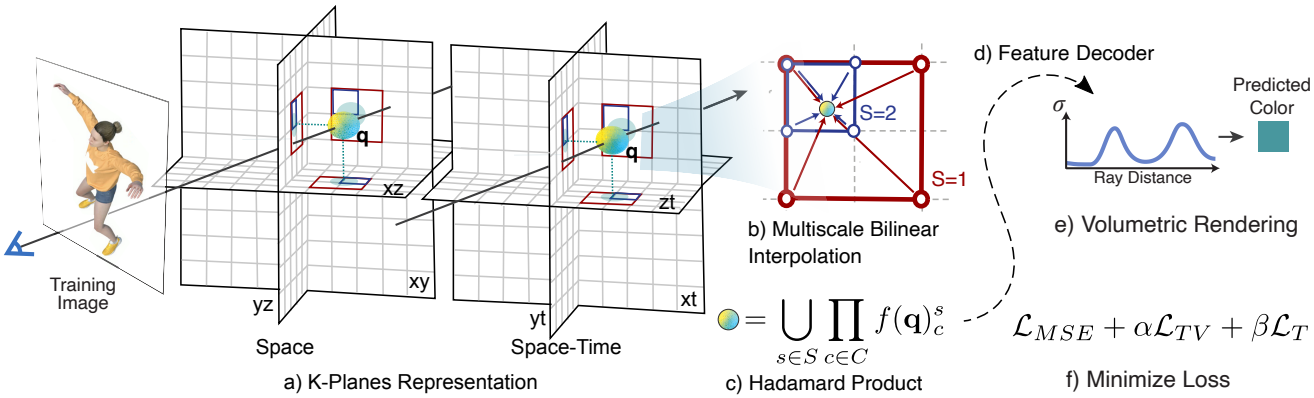


Figure 2. **Method overview.** (a) Our k-planes representation factorizes 4D dynamic volumes into six planes, three for space and three for spatiotemporal variations. To obtain the value of a 4D point $\mathbf{q} = (x, y, z, t)$, we first project the point into each plane, in which we (b) do multiscale bilinear interpolation. (c) The interpolated values are multiplied and then concatenated over the S scales. (d) These features are decoded either with a small MLP or our explicit linear decoder. (e) We follow the standard volumetric rendering formula to predict ray color and density. The model is optimized by (f) minimizing the reconstruction loss with simple regularization in space and time.

rank decomposition to represent 4D dynamic volumes in magnetic resonance imaging. These spatial decomposition methods have been shown to offer a favorable balance of memory efficiency and optimization time for static scenes. However, it is not obvious how to extend these factorizations to 4D volumes in a memory-efficient way. K-planes defines a unified framework that enables efficient and interpretable factorizations of 3D and 4D volumes and trivially extends to even higher dimensional volumes.

Dynamic volumes. Applications such as Virtual Reality (VR) and Computed Tomography (CT) often require the ability to reconstruct 4D volumes. Several works have proposed extensions of NeRF to dynamic scenes. The two most common schemes are (1) modeling a deformation field on top of a static *canonical* field [8, 9, 17, 27, 30, 36, 43], or (2) directly learning a radiance field conditioned on time [12, 16, 17, 28, 41]. The former makes decomposing static and dynamic components easy [40, 43], but struggles with changes in scene topology (e.g. when a new object appears), while the latter makes disentangling static and dynamic objects hard. A third strategy is to choose a representation of 3D space and repeat it at each timestep (e.g. NeRFPlayer [32]), resulting in a model that ignores space-time interactions and can become impractically large for long videos.

Further, some of these models are fully implicit [16, 30] and thus suffer from extremely long training times (e.g. DyNeRF used 8 GPUs for 1 week to train a single scene), as well as being completely black-box. Others use partially explicit decompositions for video [9, 11, 14, 18, 19, 31, 32, 37], usually combining some voxel or spatially decomposed feature grid with one or more MLP components for feature decoding and/or representing scene dynamics. Most closely related to k-planes is Tensor4D [31], which uses 9 planes to decompose 4D volumes. K-planes is less redundant (*e.g.* Tensor4D includes two yt planes), does not rely on multiple MLPs, and offers a simpler factorization that naturally generalizes to static and dynamic scenes. Our method combines a fully explicit representation with a built-in decomposition of static and dynamic components, the ability to handle arbitrary topology and lighting changes over time, fast optimization, and compactness.

Appearance embedding. Reconstructing large environments from photographs taken with varying illumination is another domain in which implicit methods have shown appealing results, but hybrid and explicit approaches have not yet gained a foothold. NeRF-W [20] was the first to demonstrate photorealistic view synthesis in such environments. They augment a NeRF-based model with a learned global appearance code per frame, enabling it to explain away changes in appearance, such as time of day. With Generative Latent Optimization (GLO) [4], these appearance codes can further be used to manipulate the scene appearance by interpolation in the latent appearance space. Block-NeRF [34] employs similar appearance codes.

We show that our k-planes representation can also effectively reconstruct these unbounded environments with varying appearance. We similarly extend our model – either the learned color basis in the fully explicit version, or the MLP decoder in the hybrid version – with a global appearance code to disentangle global appearance from a scene without affecting geometry. To the best of our knowledge, ours is both the first fully explicit and the first hybrid method to successfully reconstruct these challenging scenes.

3. K-planes model

We propose a simple and interpretable model for representing scenes in arbitrary dimensions. Our representation yields low memory usage and fast training and rendering. The k-planes factorization, illustrated in Fig. 2, models a d-dimensional scene using $k = \binom{d}{2}$ planes representing every

combination of two dimensions. For example, for static 3D scenes, this results in *tri-planes* with $\binom{3}{2} = 3$ planes representing xy, xz, and yz. For dynamic 4D scenes, this results in *hex-planes*, with $\binom{4}{2} = 6$ planes including the three space-only planes and three space-time planes xt, yt, and zt. Should we wish to represent a 5D space, we could use $\binom{5}{2} = 10$ *deca-planes*. In the following section, we describe the 4D instantiation of our k-planes factorization.

3.1. Hex-planes

The hex-planes factorization uses six planes. We refer to the space-only planes as $\mathbf{P}_{xy}$, $\mathbf{P}_{xz}$, and $\mathbf{P}_{yz}$, and the space-time planes as $\mathbf{P}_{xt}$, $\mathbf{P}_{yt}$, and $\mathbf{P}_{zt}$. Assuming symmetric spatial and temporal resolution N for simplicity of illustration, each of these planes has shape $N \times N \times M$, where M is the size of stored features that capture the density and view-dependent color of the scene.

We obtain the features of a 4D coordinate $\boldsymbol{q} = (i, j, k, \tau)$ by normalizing its entries between $[0, N)$ and projecting it onto these six planes

$$f(\boldsymbol{q})_c = \psi\big(\mathbf{P}_c, \pi_c(\boldsymbol{q})\big), \tag{1}$$

where π_c projects $\boldsymbol{q}$ onto the c'th plane and ψ denotes bilinear interpolation of a point into a regularly spaced 2D grid. We repeat Eq. (1) for each plane $c \in C$ to obtain feature vectors $f(\boldsymbol{q})_c$. We combine these features over the six planes using the Hadamard product (elementwise multiplication) to produce a final feature vector of length M

$$f(\boldsymbol{q}) = \prod_{c \in C} f(\boldsymbol{q})_c. \tag{2}$$

These features will be decoded into color and density using either a linear decoder or an MLP, described in Sec. 3.3.

Why Hadamard product? In 3D, k-planes reduces to the tri-plane factorization, which is similar to [5] except that the elements are multiplied. A natural question is why we multiply rather than add, as has been used in prior work with tri-plane models [5, 29]. Fig. 3 illustrates that combining the planes by multiplication allows k-planes to produce spatially localized signals, which is not possible with addition.

This selection ability of the Hadamard product produces substantial rendering improvements for linear decoders and modest improvement for MLP decoders, as shown in Tab. 2. This suggests that the MLP decoder is involved in both view-dependent color and determining spatial structure. The Hadamard product relieves the feature decoder of this extra task and makes it possible to reach similar performance using a linear decoder solely responsible for view-dependent color.

3.2. Interpretability

The separation of space-only and space-time planes makes the model interpretable and enables us to incorporate dimension-specific priors. For example, if a region of the scene never moves, its temporal component will always be 1 (the multiplicative identity), thereby just using the features from the space planes. This offers compression benefits since a static region can easily be identified and compactly represented. Furthermore, the space-time separation improves interpretability, *i.e.* we can track the changes in time by visualizing the elements in the time-space planes that are not 1. This simplicity, separation, and interpretability make adding priors straightforward.

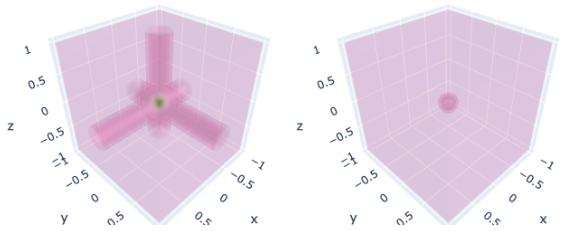


Figure 3. **Addition versus Hadamard product.** Elementwise addition of plane features (left) compared to multiplication (right), in a triplane example. A single entry in each plane is positive and the rest are zero, selecting a single 3D point by multiplication but producing intersecting lines by addition. This selection ability of multiplication improves the expressivity of our explicit model.

Plane Combination	Explicit	Hybrid	# params ↓
Multiplication	35.29	35.75	33M
Addition	28.78	34.80	33M

Table 2. **Ablation study over Hadamard product.** Multiplication of plane features yields a large improvement in PSNR ↑ for our explicit model, whereas our hybrid model can use its MLP decoder to partially compensate for the less expressive addition of planes. This experiment uses the static *Lego* scene [24] with 3 scales: 128, 256, and 512, and 32 features per scale.

Multiscale planes. To encourage spatial smoothness and coherence, our model contains multiple copies at different spatial resolutions, for example 64, 128, 256, and 512. Models at each scale are treated separately, and the M-dimensional feature vectors from different scales are concatenated together before being passed to the decoder. The red and blue squares in Fig. 2a-b illustrate bilinear interpolation with multiscale planes. Inspired by the multiscale hash mapping of Instant-NGP [25], this representation efficiently encodes spatial features at different scales, allowing us to reduce the number of features stored at the highest resolution and thereby further compressing our model. Empirically, we do not find it necessary to represent our time dimension at multiple scales.

Total variation in space. Spatial total variation regularization encourages sparse gradients (with L1 norm) or smooth gradients (with L2 norm), encoding priors over

edges being either sparse or smooth in space. We encourage this in 1D over the spatial dimensions of each of our space-time planes and in 2D over our space-only planes:

$$\mathcal{L}_{TV}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,j} \left(\|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i-1,j}\|_2^2 + \|\mathbf{P}_c^{i,j} - \mathbf{P}_c^{i,j-1}\|_2^2 \right), \tag{3}$$

where i, j are indices on the plane's resolution. Total variation is a common regularizer in inverse problems and was used in Plenoxels [10] and TensoRF [6]. We use the L2 version in our results, though we find that either L2 or L1 produces similar quality.

Smoothness in time. We encourage smooth motion with a 1D Laplacian (second derivative) filter

$$\mathcal{L}_{smooth}(\mathbf{P}) = \frac{1}{|C|n^2} \sum_{c,i,t} \|\mathbf{P}_c^{i,t-1} - 2\mathbf{P}_c^{i,t} + \mathbf{P}_c^{i,t+1}\|_2^2, \tag{4}$$

to penalize sharp "acceleration" over time. We only apply this regularizer on the time dimension of our space-time planes. Please see the appendix for an ablation study.

Sparse transients. We want the static part of the scene to be modeled by the space-only planes. We encourage this separation of space and time by initializing the features in the space-time planes as 1 (the multiplicative identity) and using an ℓ_1 regularizer on these planes during training:

$$\mathcal{L}_{sep}(\mathbf{P}) = \sum_{c} \|\mathbf{1} - \mathbf{P}_c\|_1, \qquad c \in \{xt, yt, zt\}. \tag{5}$$

In this way, the space-time plane features of the k-planes decomposition will remain fixed at 1 if the corresponding spatial content does not change over time.

3.3. Feature decoders

We offer two methods to decode the M-dimensional temporally- and spatially-localized feature vector $f(\boldsymbol{q})$ from Eq. (2) into density, σ, and view-dependent color, $\boldsymbol{c}$.

Learned color basis: a linear decoder and explicit model. Plenoxels [10], Plenoctrees [42], and TensoRF [6] proposed models where spatially-localized features are used as coefficients of the spherical harmonic (SH) basis, to describe view-dependent color. Such SH decoders can give both high-fidelity reconstructions and enhanced interpretability compared to MLP decoders. However, SH coefficients are difficult to optimize, and their expressivity is limited by the number of SH basis functions used (often limited 2nd degree harmonics, which produce blurry specular reflections).

Instead, we replace the SH functions with a learned basis, retaining the interpretability of treating features as coefficients for a linear decoder yet increasing the expressivity of the basis and allowing it to adapt to each scene, as was proposed in NeX [39]. We represent the basis using a small MLP that maps each view direction $\boldsymbol{d}$ to red $b_R(\boldsymbol{d}) \in \mathbb{R}^M$, green $b_G(\boldsymbol{d}) \in \mathbb{R}^M$, and blue $b_B(\boldsymbol{d}) \in \mathbb{R}^M$ *basis vectors*. The MLP serves as an adaptive drop-in replacement for the spherical harmonic basis functions repeated over the three color channels. We obtain the color values

$$\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d}) = \bigcup_{i \in \{R,G,B\}} f(\boldsymbol{q}) \cdot b_i(\boldsymbol{d}), \tag{6}$$

where $\cdot$ denotes the dot product and $\cup$ denotes concatenation. Similarly, we use a learned basis $b_\sigma \in \mathbb{R}^M$, independent of the view direction, as a linear decoder for density:

$$\sigma(\boldsymbol{q}) = f(\boldsymbol{q}) \cdot b_\sigma. \tag{7}$$

Predicted color and density values are finally forced to be in their valid range by applying the sigmoid to $\boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d})$, and the exponential (with truncated gradient) to $\sigma(\boldsymbol{q})$.

MLP decoder: a hybrid model. Our model can also be used with an MLP decoder like that of Instant-NGP [25] and DVGO [33], turning it into a hybrid model. In this version, features are decoded by two small MLPs, one g_σ that maps the spatially-localized features into density σ and additional features $\hat{f}$, and another g_{RGB} that maps $\hat{f}$ and the embedded view direction $\gamma(\boldsymbol{d})$ into RGB color

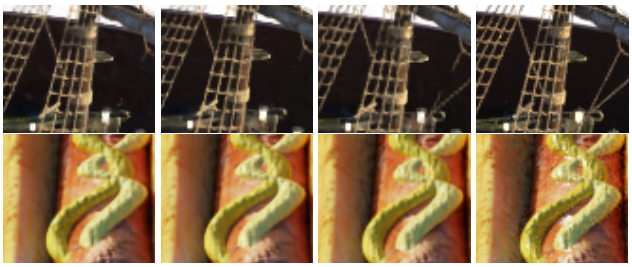
$$\begin{aligned} \sigma(\boldsymbol{q}), \hat{f}(\boldsymbol{q}) &= g_\sigma(f(\boldsymbol{q})) \\ \boldsymbol{c}(\boldsymbol{q}, \boldsymbol{d}) &= g_{RGB}(\hat{f}(\boldsymbol{q}), \gamma(\boldsymbol{d})). \end{aligned} \tag{8}$$

As in the linear decoder case, the predicted density and color values are finally normalized via exponential and sigmoid, respectively.

Global appearance. We also show a simple extension of our k-planes model that enables it to represent scenes with consistent, static geometry viewed under varying lighting or appearance conditions. Such scenes appear in the Phototourism [15] dataset of famous landmarks photographed at different times of day and in different weather. To model this variable appearance, we augment k-planes with an M-dimensional vector for each training image $1, \ldots, T$. Similar to NeRF-W [20], we optimize this per-image feature vector and pass it as an additional input to either the MLP learned color basis b_R, b_G, b_B, in our explicit version, or to the MLP color decoder g_{RGB}, in our hybrid version, so that it can affect color but not geometry.

3.4. Optimization details

Contraction and normalized device coordinates. For forward-facing scenes, we apply normalized device coordinates (NDC) [24] to better allocate our resolution while enabling unbounded depth. We also implement an ℓ_∞ version (rather than ℓ_2) of the scene contraction proposed in Mip-NeRF 360 [2], which we use on the unbounded Phototourism scenes.



(a) Ours-explicit (b) Ours-hybrid (c) TensoRF (d) Ground truth

Figure 4. **Zoomed qualitative results on static NeRF scenes.** Visual comparison of k-planes, TensoRF [6], and the ground truth, on *ship* (top) and *hotdog* (bottom).

(a) Ours-explicit (b) Ours-hybrid (c) TensoRF (d) Ground truth

Figure 5. **Zoomed qualitative results on static LLFF scenes.** Visual comparison of k-planes, TensoRF [6], and the ground truth, on *orchids* (top) and *T-rex* (bottom).

Proposal sampling. We use a variant of the proposal sampling strategy from Mip-NeRF 360 [2], with a small instance of k-planes as density model. Proposal sampling works by iteratively refining density estimates along a ray, to allocate more points in the regions of higher density. We use a two-stage sampler, resulting in fewer samples that must be evaluated in the full model and in sharper details by placing those samples closer to object surfaces. The density models used for proposal sampling are trained with the histogram loss [2].

Importance sampling. For multiview dynamic scenes, we implement a version of the importance sampling based on temporal difference (IST) strategy from DyNeRF [16]. During the last portion of optimization, we sample training rays proportionally to the maximum variation in their color within 25 frames before or after. This results in higher sampling probabilities in the dynamic region. We apply this strategy after the static scene has converged with uniformly sampled rays. In our experiments, IST has only a modest impact on full-frame metrics but improves visual quality in the small dynamic region. Note that importance sampling cannot be used for monocular videos or datasets with moving cameras.

4. Results

We demonstrate the broad applicability of our planar decomposition via experiments in three domains: static scenes (both bounded 360° and unbounded forward-facing), dynamic scenes (forward-facing multi-view and bounded 360° monocular), and Phototourism scenes with variable appearance. For all experiments, we report the metrics PSNR (pixel-level similarity) and SSIM[1] [38] (structural similarity), as well as approximate training time and number of parameters (in millions), in Tab. 3. Blank entries in Tab. 3 denote baseline methods for which the corresponding information is not readily available. Full per-scene results may be found in the appendix.

[1]Note that among prior work, some evaluate using an implementation of SSIM from MipNeRF [1] whereas others use the scikit-image implementation, which tends to produce higher values. For fair comparison on each dataset we make a best effort to use the same SSIM implementation as the relevant prior work.

4.1. Static scenes

We first demonstrate our triplane model on the bounded, 360°, synthetic scenes from NeRF [24]. We use a model with three symmetric spatial resolutions $N \in \{128, 256, 512\}$ and feature length $M = 32$ at each scale; please see the appendix for ablation studies over these hyperparameters. The explicit and hybrid versions of our model perform similarly, within the range of recent results on this benchmark. Fig. 4 shows zoomed-in visual results on a small sampling of scenes. We also present results of our triplane model on the unbounded, forward-facing, real scenes from LLFF [23]. Our results on this dataset are similar to the synthetic static scenes; both versions of our model match or exceed the prior state-of-the-art, with the hybrid version achieving slightly higher metrics than the fully explicit version. Fig. 5 shows zoomed-in visual results on a small sampling of scenes.

4.2. Dynamic scenes

We evaluate our hexplane model on two dynamic scene datasets: a set of synthetic, bounded, 360°, monocular videos from D-NeRF [30] and a set of real, unbounded, forward-facing, multiview videos from DyNeRF [16].

The D-NeRF dataset contains eight videos of varying duration, from 50 frames to 200 frames per video. Each timestep has a single training image from a different viewpoint; the camera "teleports" between adjacent timestamps [13]. Standardized test views are from novel camera positions at a range of timestamps throughout the video. Both our explicit and hybrid models outperform D-NeRF in both quality metrics and training time, though they do not surpass very recent hybrid methods TiNeuVox [9] and V4D [11], as shown in Fig. 7.

The DyNeRF dataset contains six 10-second videos recorded at 30 fps simultaneously by 15-20 cameras from a range of forward-facing view directions; the exact number of cameras varies per scene because a few cameras produced miscalibrated videos. A central camera is reserved

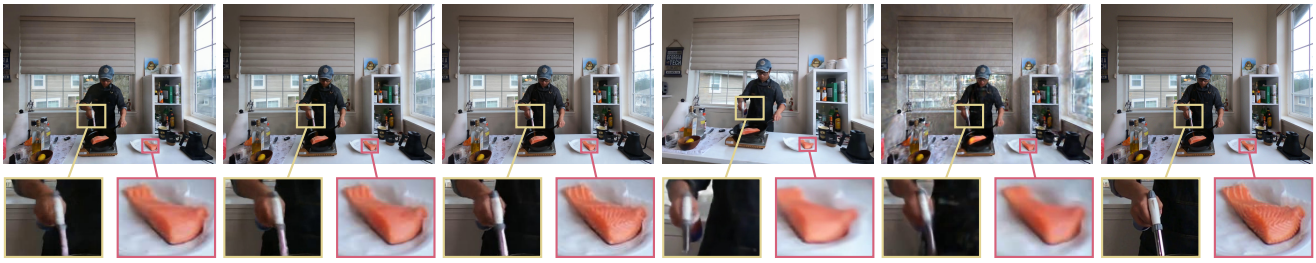
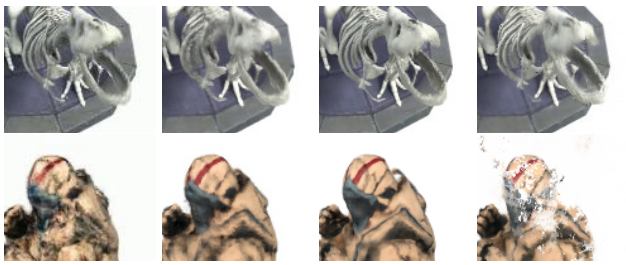


Figure 6. **Qualitative video results.** Our hexplane model rivals the rendering quality of state-of-the-art neural rendering methods. Our renderings were obtained after at most 4 hours of optimization on a single GPU whereas DyNeRF trained for a week on 8 GPUs. MixVoxels frame comes from a slightly different video rendering, and is thus slightly shifted.

	PSNR ↑	SSIM ↑	Train Time ↓	# Params ↓
NeRF [24] (static, synthetic)				
Ours-explicit	32.21	0.960	38 min	33M
Ours-hybrid	32.36	0.962	38 min	33M
Plenoxels [10]	31.71	0.958	11 min	∼500M
TensoRF [6]	33.14	0.963	17 min	18M
I-NGP [25]	33.18	-	5 min	∼ 16M
LLFF [23] (static, real)				
Ours-explicit	26.78	0.841	33 min	19M
Ours-hybrid	26.92	0.847	33 min	19M
Plenoxels	26.29	0.839	24 min	∼500M
TensoRF	26.73	0.839	25 min	45M
D-NeRF [30] (dynamic, synthetic)				
Ours-explicit	31.05	0.97	52 min	37M
Ours-hybrid	31.61	0.97	52 min	37M
D-NeRF	29.67	0.95	48 hrs	1-3M
TiNeuVox [9]	32.67	0.97	30 min	∼12M
V4D [11]	33.72	0.98	4.9 hrs	275M
DyNeRF [16] (dynamic, real)				
Ours-explicit	30.88	0.960	3.7 hrs	51M
Ours-hybrid	31.63	0.964	1.8 hrs	27M
DyNeRF [16]	[1]29.58	-	1344 hrs	7M
LLFF [23]	[1]23.24	-	-	-
MixVoxels-L [37]	30.80	0.960	1.3 hrs	125M
Phototourism [15] (variable appearance)				
Ours-explicit	22.25	0.859	35 min	36M
Ours-hybrid	22.92	0.877	35 min	36M
NeRF-W [20]	27.00	0.962	384 hrs	∼2M
NeRF-W (public)[2]	19.70	0.764	164 hrs	∼2M
LearnIt [35]	19.26	-	-	-

[1] DyNeRF and LLFF only report metrics on the *flame salmon* video (the first 10 seconds); average performance may be higher as this is one of the more challenging videos. [2] Open-source version https://github.com/kwea123/nerf_pl/tree/nerfw where we re-implemented test-time optimization as for *k*-planes.

Table 3. **Results.** Averaged metrics over all scenes in the respective datasets. Note that Phototourism scenes use MS-SSIM (multiscale structural similarity) instead of SSIM. K-planes timings are based on a single NVIDIA A30 GPU. Please see the appendix for per-scene results and the website for video reconstructions.



(a) Ours-explicit (b) TiNeuVox (c) V4D (d) D-NeRF

Figure 7. **Zoomed qualitative results on scenes from D-NeRF [30].** Visual comparison of *k*-planes, D-NeRF [30], TiNeuVox [9] and V4D [11], on *t-rex* (top) and *hook* (bottom).

for evaluation, and training uses frames from the remaining cameras. Both our methods again produce similar quality metrics to prior state-of-the-art, including recent hybrid method MixVoxels [37], with our hybrid method achieving higher quality metrics. See Fig. 6 for a visual comparison.

4.2.1 Decomposing time and space

One neat consequence of our planar decomposition of time and space is that it naturally disentangles dynamic and static portions of the scene. The static-only part of the scene can be obtained by setting the three time planes to one (the multiplicative identity). Subtracting the static-only rendered image from the full rendering (i.e. with the time plane parameters not set to 1), we can reveal the dynamic part of the scene. Fig. 9 shows this decomposition of time and space. This natural volumetric disentanglement of a scene into static and dynamic regions may enable many applications across augmented and virtual reality [3].

We can also visualize the time planes to better understand where motion occurs in a video. Fig. 8 shows the averaged features learned by the xt plane in our model for the *flame salmon* and *cut beef* DyNeRF videos, in which we can identify the motions of the hands in both space and time. The xt plane learns to be sparse, with most entries equal to the multiplicative identity, due to a combination of our sparse transients prior and the true sparsity of motion in the video. For example, in the left side of Fig. 8 one of the

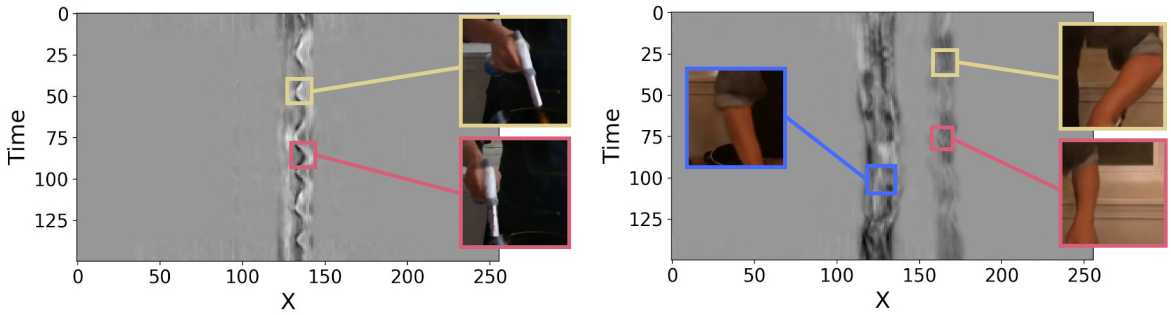


Figure 8. **Visualization of a time plane.** The xt plane highlights the dynamic regions in the scene. The wiggly patterns across time correspond to the motion of the person's hands and cooking tools, in the *flame salmon* scene (left) where only one hand moves and the *cut beef* scene (right) where both hands move.

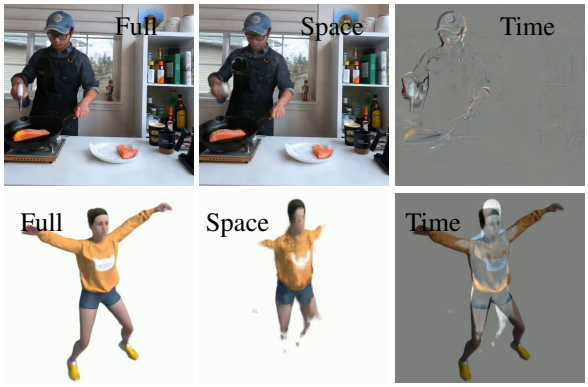


Figure 9. **Decomposition of space and time.** K-planes (left) naturally decomposes a 3D video into static and dynamic components. We render the static part (middle) by setting the time planes to the identity, and the remainder (right) is the dynamic part. Top shows the *flame salmon* multiview video [16] and bottom shows the *jumping jacks* monocular video [30].

cook's arms contains most of the motion, while in the right side both arms move. Having access to such an explicit representation of time allows us to add time-specific priors.

4.3. Variable appearance

Our variable appearance experiments use the Phototourism dataset [15], which includes photos of well-known landmarks taken by tourists with arbitrary view directions, lighting conditions, and transient occluders, mostly other tourists. Our experimental conditions parallel those of NeRF-W [20]: we train on more than a thousand tourist photographs and test on a standard set that is free of transient occluders. Like NeRF-W, we evaluate on test images by optimizing our per-image appearance feature on the left half of the image and computing metrics on the right half. Visual comparison to prior work is shown in the appendix.

Also similar to NeRF-W [4, 20], we can interpolate in the appearance code space. Since only the color decoder (and not the density decoder) takes the appearance code as input, our approach is guaranteed not to change the geometry, regardless of whether we use our explicit or our hybrid

Figure 10. **Appearance interpolation**. Like NeRF-W [20], we can interpolate our appearance code to alter the visual appearance of landmarks. We show three test views from the *Trevi fountain* with appearance codes corresponding to day and night.

model. Fig. 10 shows that our planar decomposition with a 32-dimensional appearance code is sufficient to accurately capture global appearance changes in the scene.

5. Conclusions

We introduced a simple yet versatile method to decompose a d-dimensional space into $\binom{d}{2}$ planes, which can be optimized directly from indirect measurements and scales gracefully in model size and optimization time with increasing dimension, without any custom CUDA kernels. We demonstrated that the proposed k-planes decomposition applies naturally to reconstruction of static 3D scenes as well as dynamic 4D videos, and with the addition of a global appearance code can also extend to the more challenging task of unconstrained scene reconstruction. K-planes is the first explicit, simple model to demonstrate competitive performance across such varied tasks.

Acknowledgments. Many thanks to Matthew Tancik, Ruilong Li, and other members of KAIR for helpful discussion and pointers. We also thank the DyNeRF authors for their response to our questions about their method.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, pages 5835–5844. IEEE, 2021. 6

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, pages 5460–5469. IEEE, 2022. 5, 6

[3] Sagie Benaim, Frederik Warburg, Peter Ebert Christensen, and Serge Belongie. Volumetric disentanglement for 3d scene manipulation, 2022. 7

[4] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks, 2017. 3, 8

[5] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, pages 16102–16112, 2022. 2, 4

[6] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *ECCV*, 2022. 2, 5, 6, 7, 13

[7] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020. 13

[8] Yilun Du, Yinan Zhang, Hong-Xing Yu, Joshua B. Tenenbaum, and Jiajun Wu. Neural radiance flow for 4d view synthesis and video processing. In *ICCV*, pages 14304–14314, 2021. 3

[9] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*. ACM, nov 2022. 3, 6, 7, 14

[10] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, pages 5491–5500. IEEE, 2022. 2, 5, 7, 13

[11] Wanshui Gan, Hongbin Xu, Yi Huang, Shifeng Chen, and Naoto Yokoya. V4d: Voxel for 4d novel view synthesis, 2022. 3, 6, 7, 14

[12] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *ICCV*, pages 5712–5721, 2021. 3

[13] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. In *NeurIPS*, 2022. 6

[14] Xiang Guo, Guanying Chen, Yuchao Dai, Xiaoqing Ye, Jiadai Sun, Xiao Tan, and Errui Ding. Neural deformable voxel grid for fast optimization of dynamic view synthesis. In *ACCV*, 2022. 3

[15] Yuhe Jin, Dmytro Mishkin, Anastasiia Mishchuk, Jiri Matas, Pascal Fua, Kwang Moo Yi, and Eduard Trulls. Image matching across wide baselines: From paper to practice. *Int. J. Comput. Vis.*, 129(2):517–547, 2021. 5, 7, 8

[16] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. Neural 3d video synthesis from multi-view video. In *CVPR*, pages 5511–5521, 2022. 2, 3, 6, 7, 8, 14

[17] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *CVPR*, 2021. 3

[18] Jia-Wei Liu, Yan-Pei Cao, Weijia Mao, Wenqiao Zhang, David Junhao Zhang, Jussi Keppo, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. Devrf: Fast deformable voxel radiance fields for dynamic scenes. *arxiv:2205.15723*, 2022. 3

[19] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM TOG*, 38(4):65:1–65:14, July 2019. 3

[20] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*, 2021. 3, 5, 7, 8, 15

[21] N. Max. Optical models for direct volume rendering. *IEEE TVCG*, 1(2):99–108, 1995. 11

[22] Moustafa Meshry, Dan B. Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *CVPR*, pages 6878–6887, 2019. 12

[23] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: practical view synthesis with prescriptive sampling guidelines. *ACM TOG*, 38(4):29:1–29:14, 2019. 6, 7, 13, 14

[24] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421. Springer, 2020. 2, 4, 5, 6, 7, 11, 13

[25] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM TOG*, 41(4), 2022. 2, 4, 5, 7, 13

[26] Frank Ong, Xucheng Zhu, Joseph Y. Cheng, Kevin M. Johnson, Peder E. Z. Larson, Shreyas S. Vasanawala, and Michael Lustig. Extreme mri: Large-scale volumetric dynamic imaging from continuous non-gated acquisitions. *Magnetic Resonance in Medicine*, 84(4):1763–1780, 2020. 2

[27] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, 2021. 3

[28] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM TOG*, 40(6), dec 2021. 3

[29] Songyou Peng, Michael Niemeyer, Lars M. Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *ECCV*, Lecture Notes in Computer Science, pages 523–540, 2020. 4

[30] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *CVPR*, 2021. 2, 3, 6, 7, 8, 11, 14

[31] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d : Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering, 2022. 3, 14

[32] Liangchen Song, Anpei Chen, Zhong Li, Zhang Chen, Lele Chen, Junsong Yuan, Yi Xu, and Andreas Geiger. Nerfplayer: A streamable dynamic scene representation with decomposed neural radiance fields, 2022. 3

[33] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 2, 5, 13

[34] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P. Srinivasan, Jonathan T. Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis, 2022. 3

[35] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, pages 2846–2855. Computer Vision Foundation / IEEE, 2021. 7, 15

[36] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *ICCV*. IEEE, 2021. 3

[37] Feng Wang, Sinan Tan, Xinghang Li, Zeyue Tian, and Huaping Liu. Mixed neural voxels for fast multi-view video synthesis, 2022. 3, 7, 14

[38] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 13(4):600–612, 2004. 6

[39] Suttisak Wizadwongsa, Pakkapon Phongthawee, Jiraphon Yenphraphai, and Supasorn Suwajanakorn. Nex: Real-time view synthesis with neural basis expansion, 2021. 5

[40] Tianhao Wu, Fangcheng Zhong, Andrea Tagliasacchi, Forrester Cole, and Cengiz Oztireli. D^2nerf: Self-supervised decoupling of dynamic and static objects from a monocular video, 2022. 3

[41] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *CVPR*, pages 9421–9431, 2021. 3

[42] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *ICCV*, pages 5732–5741, 2021. 5

[43] Wentao Yuan, Zhaoyang Lv, Tanner Schmidt, and Steven Lovegrove. STaR: Self-supervised tracking and reconstruction of rigid objects in motion with neural rendering. In *CVPR*, 2021. 3

6. Appendix

6.1. Volumetric rendering

We use the same volume rendering formula as NeRF [24], originally from [21], where the color of a pixel is represented as a sum over samples taken along the corresponding ray through the volume:

$$\sum_{i=1}^{N} \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \left(1 - \exp(-\sigma_i \delta_i)\right) \mathbf{c}_i \qquad (9)$$

where the first exp represents ray transmission to sample i, $1-\exp(-\sigma_i\delta_i)$ is the absorption by sample i, σ_i is the (post-activation) density of sample i, and $\mathbf{c}_i$ is the color of sample i, with distance δ_i to the next sample.

6.2. Per-scene results

Fig. 11 provides a qualitative comparison of methods for the Phototourism dataset, on the *Trevi fountain* scene. We also provide quantitative metrics for each of the three tasks we study, for each scene individually. Tab. 7 reports metrics on the static synthetic scenes, Tab. 8 reports metrics on the static real forward-facing scenes, Tab. 9 reports metrics on the dynamic synthetic monocular "teleporting camera" scenes, Tab. 10 reports metrics on the dynamic real forward-facing multiview scenes, and Tab. 11 reports metrics on the Phototourism scenes.

6.3. Ablation studies

Multiscale. In Tab. 4, we ablate our model on the static *Lego* scene [24] with respect to our multiscale planes, to assess the value of including copies of our model at different scales.

Feature length. In Tab. 5, we ablate our model on the static *Lego* scene with respect to the feature dimension M learned at each scale.

Time smoothness regularizer. Sec. 3.2 describes our temporal smoothness regularizer based on penalizing the norm of the second derivative over the time dimension, to encourage smooth motion and discourage acceleration. Tab. 6 illustrates an ablation study of this regularizer on the *Jumping Jacks* scene from D-NeRF [30].

7. Model hyperparameters

Our full hyperparameter settings are available in the config files in our released code, at https://github.com/sarafridov/K-Planes.

Scales (32 Feat. Each)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
$64, 128, 256, 512$	35.26	35.79	34M
$128, 256, 512$	35.29	35.75	33M
$256, 512$	34.52	35.37	32M
512	32.93	33.60	25M
$64, 128, 256$	34.26	35.07	8M

Scales (96 Feat. Total)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
$64, 128, 256, 512$	35.16	35.67	25M
$128, 256, 512$	35.29	35.75	33M
$256, 512$	34.50	35.16	47M
512	33.12	34.09	76M
$64, 128, 256$	34.26	35.07	8M

Table 4. **Ablation study over scales.** Including even a single lower scale improves performance, for both our explicit and hybrid models, even when holding the total feature dimension constant. Using lower scales only (excluding resolution 512^3) substantially reduces model size and yields quality much better than using high resolution alone, though slightly worse than including both low and high resolutions. This experiment uses the static *Lego* scene; in the top table each scale is allocated 32 features and in the bottom table a total of 96 features are allocated evenly among all scales.

Feature Length (M)	Explicit PSNR ↑	Hybrid PSNR ↑	# params ↓
2	30.66	32.05	2M
4	32.27	34.18	4M
8	33.80	35.12	8M
16	34.80	35.44	17M
32	35.29	35.75	33M
64	35.38	35.88	66M
128	35.45	35.99	132M

Table 5. **Ablation study over feature length M.** Increasing the feature length M learned at each scale consistently improves quality for both our models, with a corresponding linear increase in model size and optimization time. Our experiments in the main text use a mixture of $M = 16$ and $M = 32$; for specific applications it may be beneficial to vary M along this tradeoff between quality and model size. This experiment uses the static *Lego* scene with 3 scales: 128, 256, and 512.

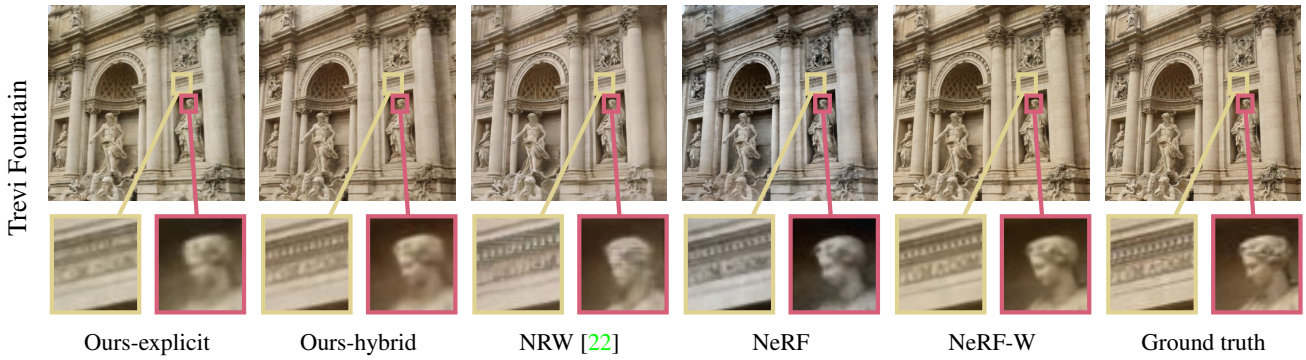


Figure 11. **Qualitative results from Phototourism dataset**. We compare our model with strong baselines. Our method captures the geometry and appearance of the scene, but produces slightly lower resolution results than NeRF-W. Note that our model optimizes in just 35 minutes on a single GPU compared to NeRF-W, which takes 2 days on 8 GPUs.

Time Smoothness Weight (λ)	Explicit PSNR ↑	Hybrid PSNR ↑
0.0	30.45	30.86
0.001	31.61	32.23
0.01	32.00	32.64
0.1	31.96	32.58
1.0	31.36	32.22
10.0	30.45	31.63

Table 6. **Ablation study over temporal smoothness regularization.** For both models, a temporal smoothness weight of 0.01 is best, with PSNR degrading gradually with over- or under-regularization. This experiment uses the *Jumping Jacks* scene with 4 scales: 64, 128, 256, and 512, and 32 features per scale.

	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
PSNR ↑									
Ours-explicit	34.82	25.72	31.2	36.65	35.29	29.49	34.00	30.51	32.21
Ours-hybrid	34.99	25.66	31.41	36.78	35.75	29.48	34.05	30.74	32.36
INGP [25]	35.00	26.02	33.51	37.40	36.39	29.78	36.22	31.10	33.18
TensoRF [6]	35.76	26.01	33.99	37.41	36.46	30.12	34.61	30.77	33.14
Plenoxels [10]	33.98	25.35	31.83	36.43	34.10	29.14	33.26	29.62	31.71
JAXNeRF [7, 24]	34.20	25.27	31.15	36.81	34.02	30.30	33.72	29.33	31.85
SSIM ↑									
	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Mean
Ours-explicit	0.981	0.937	0.975	0.982	0.978	0.949	0.988	0.892	0.960
Ours-hybrid	0.983	0.938	0.975	0.982	0.982	0.950	0.988	0.897	0.962
INGP	-	-	-	-	-	-	-	-	-
TensoRF	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963
Plenoxels	0.977	0.933	0.976	0.980	0.975	0.949	0.985	0.890	0.958
JAXNeRF	0.975	0.929	0.970	0.978	0.970	0.955	0.983	0.868	0.954

Table 7. **Full results on static synthetic scenes [24].** Dashes denote values that were not reported in prior work.

	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Mean
PSNR ↑									
Ours-explicit	32.72	24.87	21.07	31.34	19.89	28.37	27.54	28.40	26.78
Ours-hybrid	32.64	25.38	21.30	30.44	20.26	28.67	28.01	28.64	26.92
NeRF [24]	32.70	25.17	20.92	31.16	20.36	27.40	26.80	27.45	26.50
Plenoxels [10]	30.22	25.46	21.41	31.09	20.24	27.83	26.48	27.58	26.29
TensoRF (L) [6]	32.35	25.27	21.30	31.36	19.87	28.60	26.97	28.14	26.73
DVGOv2 [33]	-	-	-	-	-	-	-	-	26.34
SSIM ↑									
	Room	Fern	Leaves	Fortress	Orchids	Flower	T-Rex	Horns	Mean
Ours-explicit	0.955	0.809	0.738	0.898	0.665	0.867	0.909	0.884	0.841
Ours-hybrid	0.957	0.828	0.746	0.890	0.676	0.872	0.915	0.892	0.847
NeRF [24]	0.948	0.792	0.690	0.881	0.641	0.827	0.880	0.828	0.811
Plenoxels [10]	0.937	0.832	0.760	0.885	0.687	0.862	0.890	0.857	0.839
TensoRF (L) [6]	0.952	0.814	0.752	0.897	0.649	0.871	0.900	0.877	0.839
DVGOv2 [33]	-	-	-	-	-	-	-	-	0.838

Table 8. **Full results on static forward-facing scenes [23].** Dashes denote values that were not reported in prior work.

PSNR ↑

	Hell Warrior	Mutant	Hook	Balls	Lego	T-Rex	Stand Up	Jumping Jacks	Mean
Ours-explicit	25.60	33.56	28.21	38.99	25.46	31.28	33.27	32.00	31.05
Ours-hybrid	25.70	33.79	28.50	41.22	25.48	31.79	33.72	32.64	31.61
D-NeRF [30]	25.02	31.29	29.25	32.80	21.64	31.75	32.79	32.80	29.67
T-NeRF [30]	23.19	30.56	27.21	32.01	23.82	30.19	31.24	32.01	28.78
Tensor4D [31]	-	-	-	-	26.71	-	36.32	34.43	-
TiNeuVox [9]	28.17	33.61	31.45	40.73	25.02	32.70	35.43	34.23	32.67
V4D [11]	27.03	36.27	31.04	42.67	25.62	34.53	37.20	35.36	33.72

SSIM ↑

	Hell Warrior	Mutant	Hook	Balls	Lego	T-Rex	Stand Up	Jumping Jacks	Mean
Ours-explicit	0.951	0.982	0.951	0.989	0.947	0.980	0.980	0.974	0.969
Ours-hybrid	0.952	0.983	0.954	0.992	0.948	0.981	0.983	0.977	0.971
D-NeRF [30]	0.95	0.97	0.96	0.98	0.83	0.97	0.98	0.98	0.95
T-NeRF [30]	0.93	0.96	0.94	0.97	0.90	0.96	0.97	0.97	0.95
Tensor4D [31]	-	-	-	-	0.953	-	0.983	0.982	-
TiNeuVox [9]	0.97	0.98	0.97	0.99	0.92	0.98	0.99	0.98	0.97
V4D [11]	0.96	0.99	0.97	0.99	0.95	0.99	0.99	0.99	0.98

Table 9. **Full results on monocular "teleporting-camera" dynamic scenes.** We use the synthetic scenes from D-NeRF [30], which we refer to as monocular "teleporting-camera" because although there is a single training view per timestep, the camera can move arbitrarily between adjacent timesteps. Dashes denote unreported values. TiNeuVox trains in 30 minutes, V4D in 4.9 hours, D-NeRF in 2 days, and Tensor4D for an unspecified duration (Tensor4D reports iterations rather than time). Our reported results were obtained after roughly 1 hour of optimization on a single GPU. Like D-NeRF and TiNeuVox, we train and evaluate using half-resolution images (400 by 400 pixels).

PSNR ↑

	Coffee Martini	Spinach	Cut Beef	Flame Salmon[1]	Flame Steak	Sear Steak	Mean
Ours-explicit	28.74	32.19	31.93	28.71	31.80	31.89	30.88
Ours-hybrid	29.99	32.60	31.82	30.44	32.38	32.52	31.63
LLFF [23]	-	-	-	23.24	-	-	-
DyNeRF [16]	-	-	-	29.58	-	-	-
MixVoxels-L† [37]	29.36	31.61	31.30	29.92	31.21	31.43	30.80

SSIM ↑

	Coffee Martini	Cook Spinach	Cut Beef	Flame Salmon[1]	Flame Steak	Sear Steak	Mean
Ours-explicit	0.943	0.968	0.965	0.942	0.970	0.971	0.960
Ours-hybrid	0.953	0.966	0.966	0.953	0.970	0.974	0.964
LLFF	-	-	-	0.848	-	-	-
DyNeRF	-	-	-	0.961	-	-	-
MixVoxels-L	0.946	0.965	0.965	0.945	0.970	0.971	0.960

† Very recent/concurrent work. MixVoxels was released in December 2022. [1]Using the first 10 seconds of the 30 second long video.

Table 10. **Full results on multiview dynamic scenes [16].** Dashes denote unreported values. Note that our method optimizes in less than 4 GPU hours, whereas DyNeRF trains on 8 GPUs for a week, approximately 1344 GPU hours.

PSNR ↑				
	Brandenburg Gate	Sacre Coeur	Trevi Fountain	Mean
Ours-explicit	24.85	19.90	22.00	22.25
Ours-hybrid	25.49	20.61	22.67	22.92
NeRF-W [20]	29.08	25.34	26.58	27.00
NeRF-W (public)†	21.32	19.17	18.61	19.70
LearnIt [35]	19.11	19.33	19.35	19.26
MS-SSIM ↑				
	Brandenburg Gate	Sacre Coeur	Trevi Fountain	Mean
Ours-explicit	0.912	0.821	0.845	0.859
Ours-hybrid	0.924	0.852	0.856	0.877
NeRF-W	0.962	0.939	0.934	0.945
Nerf-W (public)†	0.845	0.752	0.694	0.764
LearnIt	-	-	-	-

† Open-source version https://github.com/kwea123/nerf_pl/tree/nerfw where we implement the test-time optimization ourselves exactly as for k-planes. NeRF-W code is not public.

Table 11. **Full results on phototourism scenes.** Note that our results were obtained after about 35 GPU minutes, whereas NeRF-W trains with 8 GPUs for two days, approximately 384 GPU hours.

Estimating Koopman operators with sketching to provably learn large scale dynamical systems

Giacomo Meanti[1,*] giacomo.meanti@iit.it
Antoine Chatalic[2,*] antoine.chatalic@dibris.unige.it
Vladimir R. Kostic[1,3] vladimir.kostic@iit.it
Pietro Novelli[1] pietro.novelli@iit.it
Massimiliano Pontil[1,5] massimiliano.pontil@iit.it
Lorenzo Rosasco[1,2,4] lrosasco@mit.edu

Abstract

The theory of Koopman operators allows to deploy non-parametric machine learning algorithms to predict and analyze complex dynamical systems. Estimators such as principal component regression (PCR) or reduced rank regression (RRR) in kernel spaces can be shown to provably learn Koopman operators from finite empirical observations of the system's time evolution. Scaling these approaches to very long trajectories is a challenge and requires introducing suitable approximations to make computations feasible. In this paper, we boost the efficiency of different kernel-based Koopman operator estimators using random projections (sketching). We derive, implement and test the new "sketched" estimators with extensive experiments on synthetic and large-scale molecular dynamics datasets. Further, we establish non asymptotic error bounds giving a sharp characterization of the trade-offs between statistical learning rates and computational efficiency. Our empirical and theoretical analysis shows that the proposed estimators provide a sound and efficient way to learn large scale dynamical systems. In particular our experiments indicate that the proposed estimators retain the same accuracy of PCR or RRR, while being much faster. Code is available at `https://github.com/Giodiro/NystromKoopman`.

1 Introduction

In the physical world, temporally varying phenomena are everywhere, from biological processes in the cell to fluid dynamics to electrical fields. Correspondingly, they generate large amounts of data both through experiments and simulations. This data is often analyzed in the framework of dynamical systems, where the state of a system $\boldsymbol{x}$ is observed at a certain time t, and the dynamics is described by a function f which captures its evolution in time

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t).$$

The function f must capture the whole dynamics, and as such it may be non-linear and even stochastic for instance when modeling stochastic differential equations, or simply noisy processes. Applications of this general formulation arise in fields ranging from robotics, atomistic simulations, epidemiology, and many more. Along with a recent increase in the availability of simulated data, data-driven techniques for learning the dynamics underlying physical systems have become commonplace. The typical approach of such techniques is to acquire a dataset of training pairs $(\boldsymbol{x}_t, \boldsymbol{y}_t = \boldsymbol{x}_{t+1})$ sampled

[*]Equal contribution
[1]Istituto Italiano di Tecnologia
[2]MaLGa – DIBRIS, Università di Genova
[3]University of Novi Sad
[4]MIT, CBMM
[5]University College London

Preprint. Under review.

in time, and use them to learn a model for f which minimizes a forecasting error. Since dynamical systems stem from real physical processes, forecasting is not the only goal and the ability to interpret the dynamics is paramount. One particularly important dimension for interpretation is the separation of dynamics into multiple temporal scales: fast fluctuations can e.g. be due to thermodynamical noise or electrical components in the system, while slow dynamics describe important conformational changes in molecules or mechanical effects.

Koopman operator theory [24, 25] provides an elegant framework in which the potentially non-linear dynamics of the system can be studied via the Koopman operator

$$(\mathcal{K}\psi)(\boldsymbol{x}) = \mathbf{E}\big[\psi(f(\boldsymbol{x}))\big], \tag{1}$$

which has the main advantage of being linear but is defined on a typically infinite-dimensional set of observable functions. The expectation in (1) is taken with respect to the potential stochasticity of f. Thanks to its linearity, the operator $\mathcal{K}$ can e.g. be applied twice to get two-steps-ahead forecasts, and one can compute its spectrum (beware however that $\mathcal{K}$ is not self-adjoint, unless the dynamical process is time-reversible). Accurately approximating the Koopman operator and its spectral properties is of high interest for the practical analysis of dynamical systems. However doing so efficiently for long temporal trajectories remains challenging. In this paper we are interested in designing estimators which are both theoretically accurate and computationally efficient.

Related works Learning the spectral properties of the Koopman operator directly from data has been considered for at least 3 decades [36], resulting in a large body of previous work. Among the different approaches proposed over time (see Mezić [37] for a recent review) it is most common to search for finite dimensional approximations to the operator. DMD [52, 59], tICA [38, 45] and many subsequent extensions [28] for example can be seen as minimizers of the forecasting error when ψ is restricted to be a linear function of the states [48]. eDMD [62, 22] and VAC [41, 42] instead allow for a (potentially learnable, as in recent deep learning algorithms [29, 34, 65, 58]) dictionary of non-linear functions ψ. KernelDMD [63, 23] and kernel tICA [53] are further generalizations which again approximate the Koopman operator but using an infinite dimensional space of features ψ, encoded by the feature map of a reproducing kernel. While often slow from a computational point of view, kernel methods are highly expressive and can be analyzed theoretically, to prove convergence and derive learning rates of the resulting estimators [26]. Approximate kernel methods which are much faster to run have been recently used for Koopman operator learning by Baddoo et al. [6] where an iterative procedure is used to identify the best approximation to the full kernel, but no formal learning rates are demonstrated, and by Ahmad et al. [3] who derive learning rates in Hilbert-Schmidt norm (while we consider operator norm) for the Nyström KRR estimator (one of the three considered in this paper).

Contributions In this paper we adopt the kernel learning approach. Starting from the problem of approximating the Koopman operator in a reproducing kernel Hilbert space, we derive three different estimators based on different inductive biases: kernel ridge regression (KRR) which comes from Tikhonov regularization, principal component regression (PCR) which is equivalent to dynamic mode decompositin (DMD) and its extensions, and reduced rank regression (RRR) which comes from a constraint on the maximum rank of the estimator [21]. We show how to overcome the computational scalability problems inherent in full kernel methods using an approximation based on random projections which is known as the Nyström method [54, 61]. The approximate learning algorithms scale very easily to the largest datasets, with a computational complexity which goes from $O(n^3)$ for the exact algorithm to $O(n^2)$ for the approximate one. We can further show that the Nyström KRR, PCR and RRR estimators have the same convergence rates as theirs exact, slow counterparts – which are known to be optimal under our assumptions. We provide learning bounds in operator norm, which are known to translate to bounds for dynamic mode decomposition and are thus of paramount importance for applications. Finally, we thoroughly validate the approximate PCR and RRR estimators on synthetic dynamical systems, comparing efficiency and accuracy against their exact counterparts [26], as well as recently proposed fast Koopman estimator streaming KAF [18]. To showcase a realistic scenario, we train on a molecular dynamics simulation of the fast-folding Trp-cage protein [32].

Structure of the paper We introduce the setting in Section 2, and define our three estimators in Section 3. In Section 4 we provide bounds on the excess risk of our estimators, and extensive experiments on synthetic as well as large-scale molecular dynamics datasets in Section 5.

2 Background and related work

Notation We consider a measurable space $(\mathcal{X}, \mathcal{B})$ where $\mathcal{X}$ corresponds to the state space, and denote $L^2_\pi := L^2(\mathcal{X}, \mathcal{B}, \pi)$ the L^2 space of functions on $\mathcal{X}$ w.r.t. to a probability measure π, and L^∞_π the space of measurable functions bounded almost everywhere. We denote $\mathrm{HS}(\mathcal{H})$ the space of Hilbert-Schmidt operators on a space $\mathcal{H}$.

Setting The setting we will consider is that of Markovian, time-homogeneous stochastic process $\{X_t\}_{t\in\mathbb{N}}$ on $\mathcal{X}$. By definition of a Markov process, X_t only depends on X_{t-1} and not on any previous states. Time-homogeneity ensures that the transition probability $\mathbb{P}\big[X_{t+1} \in B | X_t = \boldsymbol{x}\big]$ for any measurable set B does not depend on t, and can be denoted with $p(\boldsymbol{x}, B)$. This implies in particular that the distribution of (X_t, X_{t+1}) does not depend on t, and we denote it ρ in the following. We further assume the existence of the *invariant* density π which satisfies $\pi(B) = \int_{\mathcal{X}} \pi(\boldsymbol{x}) p(\boldsymbol{x}, B)\,\mathrm{d}\boldsymbol{x}$. This classical assumption allows one to study large class of stochastic dynamical systems, but also deterministic systems on the attractor, see e.g. [12]. The Koopman operator $\mathcal{K}_\pi : L^2_\pi(\mathcal{X}) \to L^2_\pi(\mathcal{X})$ is a bounded linear operator, defined by

$$(\mathcal{K}_\pi g)(\boldsymbol{x}) = \int_{\mathcal{X}} p(\boldsymbol{x}, \boldsymbol{y}) g(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} = \mathbf{E}\big[g(X_{t+1}) | X_t = \boldsymbol{x}\big], \quad g \in L^2_\pi(\mathcal{X}), \boldsymbol{x} \in \mathcal{X}. \tag{2}$$

We are in particular interested in the eigenpairs $(\lambda_i, \varphi_i) \in \mathbb{C} \times L^2_\pi$, that satisfy

$$\mathcal{K}_\pi \varphi_i = \lambda_i \varphi_i. \tag{3}$$

Through this decomposition it is possible to interpret the system by separating fast and slow processes, or projecting the states onto fewer dimensions [13, 17, 7]. In particular, the Koopman mode decomposition (KMD) allows to propagate the system state in time. Given an observable $g : \mathcal{X} \to \mathbb{R}^d$ such that $g \in \operatorname{span}\{\varphi_i | i \in \mathbb{N}\}$, the modes allow to reconstruct $g(\boldsymbol{x})$ with a Koopman eigenfunction basis. The modes $\boldsymbol{\eta}^g_i \in \mathbb{C}^d$ are the coefficients of this basis expansion:

$$(\mathcal{K}_\pi g)(\boldsymbol{x}) = \mathbf{E}\big[g(X_t) | X_0 = \boldsymbol{x}\big] = \sum_i \lambda_i \varphi_i(\boldsymbol{x}) \boldsymbol{\eta}^g_i. \tag{4}$$

This decomposition describes the system's dynamics in terms of a stationary component (the Koopman modes), a temporal component (the eigenvalues λ_i) and a spatial component (eigenfunctions φ_i).

Kernel-based learning In this paper we approximate $\mathcal{K}_\pi$ with kernel-based algorithms, using operators in reproducing kernel Hilbert spaces (RKHS) $\mathcal{H}$ associated with kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and feature map $\phi : \mathcal{X} \to \mathcal{H}$. We wish to find an operator $A : \mathcal{H} \to \mathcal{H}$ which minimizes the risk

$$\mathcal{R}_{\mathrm{HS}}(A) = \mathbf{E}_\rho\big[\ell(A, (\boldsymbol{x}, \boldsymbol{y}))\big] \quad \text{where} \quad \ell(A, (\boldsymbol{x}, \boldsymbol{y})) := \|\phi(\boldsymbol{y}) - A\phi(\boldsymbol{x})\|^2. \tag{5}$$

The operator A^* should thus be understood as an estimator of the Koopman operator $\mathcal{K}_\pi$ in $\mathcal{H}$ as will be clarified in (15). In practice π and ρ are unknown, and one typically has access to a dataset $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ sampled from ρ, where each pair $(\boldsymbol{x}_i, \boldsymbol{y}_i = f(\boldsymbol{x}_i))$ may equivalently come from a single long trajectory or multiple shorter ones concatenated together. We thus use the empirical risk

$$\hat{\mathcal{R}}_{\mathrm{HS}}(A) = \frac{1}{n} \sum_{i=1}^n \ell(A, (\boldsymbol{x}_i, \boldsymbol{y}_i)) \tag{6}$$

as a proxy for (5). In practice, minimizing eq. (6) may require finding the solution to a very badly conditioned linear system. To avoid this potential pitfall, different regularization methods (such as Tikhonov or truncated SVD) can be applied on top of the empirical risk.

Remark 2.1 (Connections to other learning problems): *The problem of minimizing eqs.* (5) *and* (6) *has strong connections to learning conditional mean embeddings [55, 40, 30] where the predictors and targets are embedded in different RKHSs, and to structured prediction [10, 11] which is an even more general framework. On the other hand, the most substantial difference from the usual kernel regression setting [8] is the embedding of both targets and predictors into a RKHS, instead of just targets.*

We denote the input and cross covariance $C = \mathbf{E}_\pi[\phi(\boldsymbol{x}) \otimes \phi(\boldsymbol{x})]$ and $C_{YX} = \mathbf{E}_\rho[\phi(\boldsymbol{y}) \otimes \phi(\boldsymbol{x})]$, and their empirical counterparts as $\hat{C} = \frac{1}{n}\sum_{i=1}^n [\phi(\boldsymbol{x}_i) \otimes \phi(\boldsymbol{x}_i)]$ and $\hat{C}_{YX} = \frac{1}{n}\sum_{i=1}^n \phi(\boldsymbol{y}_i) \otimes \phi(\boldsymbol{x}_i)]$.

We also use the abbreviation $C_\lambda := C + \lambda I$. Minimizing the empirical risk (6) with Tikhonov regularization [8] yields the following KRR estimator

$$\hat{A}_\lambda = \operatorname*{arg\,min}_{A \in \mathrm{HS}(\mathcal{H})} \hat{\mathcal{R}}_{\mathrm{HS}}(A) + \lambda \|A\|^2_{\mathrm{HS}} = \hat{C}_{YX}(\hat{C} + \lambda I)^{-1}. \tag{7}$$

Eq. (7) can be computed by transforming its expression with the kernel trick [20], to arrive at a form where one must invert the kernel matrix – a $n \times n$ matrix whose i, j-th entry is $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This operation requires $O(n^3)$ time and $O(n^2)$ memory, severely limiting the scalability of KRR to $n \lesssim 100\,000$ points. Improving the scalability of kernel methods is a well-researched topic, with the most important solutions being random features [46, 47, 64, 19] and random projections [54, 61, 19]. In this paper we use the latter approach, whereby the kernel matrix is assumed to be approximately low-rank and is *sketched* to a lower dimensionality. In particular we will use the Nyström method to approximate the kernel matrix projecting it onto a small set of inducing points, chosen among the training set. The sketched estimators are much more efficient than the exact ones, increasingly so as the training trajectories become longer. For example, the state of the art complexity for solving (non vector valued) approximate kernel ridge regression is $O(n\sqrt{n})$ time instead of $O(n^3)$ [35, 1]. Furthermore, when enough inducing points are used (typically on the order of $\sqrt{n}$), the learning rates of the exact and approximate estimators are the same, and optimal [5, 49]. Hence it is possible – and in this paper we show it for learning the Koopman operator – to obtain large efficiency gains, without losing anything in terms of theoretical guarantees of convergence.

3 Nyström estimators for Koopman operator regression

In this section, we introduce three efficient approximations of the KRR, PCR and RRR estimators of the Koopman operator. Our estimators rely on the Nyström approximation, i.e. on random projections onto low-dimensional subspaces of $\mathcal{H}$ spanned by the feature-embeddings of subsets of the data. We thus consider two sets of $m \ll n$ inducing points $\{\tilde{\boldsymbol{x}}_j\}_{j=1}^m \subset \{\boldsymbol{x}_t\}_{t=1}^n$ and $\{\tilde{\boldsymbol{y}}_j\}_{j=1}^m \subset \{\boldsymbol{y}_t\}_{t=1}^n$ sampled respectively from the input and output data. The choice of these inducing points (also sometimes called Nyström centers) is important to obtain a good approximation. Common choices include uniform sampling, leverage score sampling [15, 51], and iterative procedures such as the one used in [6] to identify the most relevant centers. In this paper we focus on uniform sampling for simplicity, but we stress that our theoretical results in Section 4 can easily be extended to leverage scores sampling by means of [49, Lemma 7]. To formalize the Nyström estimators, we define operators $\widetilde{\Phi}_X, \widetilde{\Phi}_Y : \mathbb{R}^m \to \mathcal{H}$ as $\widetilde{\Phi}_X w = \sum_{j=1}^m w_j \phi(\tilde{\boldsymbol{x}}_j)$ and $\widetilde{\Phi}_Y w = \sum_{j=1}^m w_j \phi(\tilde{\boldsymbol{y}}_j)$, and denote P_X and P_Y the orthogonal projections onto $\operatorname{span} \widetilde{\Phi}_X$ and $\operatorname{span} \widetilde{\Phi}_Y$ respectively.

In the following paragraphs we apply the projection operators to three estimators corresponding to different choices of regularization. For each of them a specific proposition (proven in Appendix C) states an efficient way of computing it based on the kernel trick. For this purpose we introduce the kernel matrices $K_{\tilde{X},X}, K_{\tilde{Y},Y} \in \mathbb{R}^{m \times n}$ between training set and inducing points with entries $(K_{\tilde{X},X})_{ji} = k(\tilde{\boldsymbol{x}}_j, \boldsymbol{x}_i)$, $(K_{\tilde{Y},Y})_{ji} = k(\tilde{\boldsymbol{y}}_j, \boldsymbol{y}_i)$, and the kernel matrices of the inducing points $K_{\tilde{X},\tilde{X}}, K_{\tilde{Y},\tilde{Y}} \in \mathbb{R}^{m \times m}$ with entries $(K_{\tilde{X},X})_{jk} = k(\tilde{\boldsymbol{x}}_j, \tilde{\boldsymbol{x}}_k)$ and $(K_{\tilde{X},X})_{jk} = k(\tilde{\boldsymbol{y}}_j, \tilde{\boldsymbol{y}}_k)$.

Kernel Ridge Regression (KRR) The cost of computing $\hat{A}_\lambda$ defined in Eq. (7) is $O(n^3)$ [26] which is prohibitive for datasets containing long trajectories. However, applying the projection operators to each side of the empirical covariance operators, we obtain an estimator which additionally depends on the m inducing points:

$$\hat{A}^{\mathrm{KRR}}_{m,\lambda} := P_Y \hat{C}_{YX} P_X (P_X \hat{C} P_X + \lambda I)^{-1} : \mathcal{H} \to \mathcal{H}. \tag{8}$$

If $\mathcal{H}$ is infinite dimensional, Eq. (8) cannot be computed directly. Proposition 3.1 (proven in Appendix C) provides a computable version of the estimator.

Proposition 3.1 (Nyström KRR): *The Nyström KRR estimator* (8) *can be expressed as*

$$\hat{A}^{\mathit{KRR}}_{m,\lambda} = \widetilde{\Phi}_Y K^{\dagger}_{\tilde{Y},\tilde{Y}} K_{\tilde{Y},Y} K_{X,\tilde{X}} (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^{\dagger} \widetilde{\Phi}^*_X. \tag{9}$$

The computational bottlenecks are the inversion of an $m \times m$ matrix and a large matrix multiplication, which overall need $O(2m^3 + 2m^2 n)$ operations. In particular, in Section 4 we will show that

$m \asymp \sqrt{n}$ is sufficient to guarantee optimal rates even with minimal assumptions, leading to a final cost of $O(n^2)$. Note that a similar estimator was derived in [3].

Please note that the $O(n^2)$ cost is for a straightforward implementation, and can indeed be reduced via iterative linear solvers (possibly preconditioned, to further reduce the practical running time), and randomized linear algebra techniques. In particular, we could leverage results from Rudi et al. [50] to reduce the computational cost to $O(n\sqrt{n})$.

Principal Component Regression (PCR) Typical settings in which Koopman operator theory is used focus on the decomposition of a dynamical system into a small set of components, obtained from the eigendecomposition of the operator itself. For this reason, a good prior on the Koopman estimator is for it to be low rank. The kernel PCR estimator $\hat{A}^{\mathrm{PCR}} = \hat{C}_{YX}[\![\hat{C}]\!]_r^\dagger$ formalizes this concept [26, 63], where here $[\![\cdot]\!]_r$ denotes the truncation to the first r components of the spectrum. Again this is expensive to compute when n is large, but the estimator can be sketched as follows:

$$\hat{A}_m^{\mathrm{PCR}} = P_Y \hat{C}_{YX} [\![P_X \hat{C} P_X]\!]_r^\dagger. \tag{10}$$

The next proposition provides an efficiently implementable version of this estimator.

Proposition 3.2 (Nyström PCR): *The sketched PCR estimator* (10) *satisfies*

$$\hat{A}_m^{PCR} = \widetilde{\Phi}_Y K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} [\![K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} K_{X,\tilde{X}}]\!]_r \widetilde{\Phi}_X^* \tag{11}$$

requiring $O(2m^3 + 2m^2 n)$ operations, i.e. optimal rates can again be obtained at a cost of at most $O(n^2)$ operations.

Note that with $m = n$, $\hat{A}_m^{\mathrm{PCR}}$ is equivalent to the kernel DMD estimator [63], also known as kernel analog forecasting (KAF) [4]. The sketched estimator of Proposition 3.2 was also recently derived in [6], albeit without providing theoretical guarantees.

Reduced Rank Regression (RRR) Another way to promote low-rank estimators is to add an explicit rank constraint when minimizing the empirical risk. Combining such a constraint with Tikhonov regularization corresponds to the reduced rank regression [21, 26] estimator:

$$A_\lambda^{\mathrm{RRR}} = \operatorname*{arg\,min}_{A \in \mathrm{HS}:\mathrm{rk}(A) \leq r} \hat{\mathcal{R}}_{\mathrm{HS}}(A) + \lambda \|A\|_{\mathrm{HS}}^2. \tag{12}$$

Minimizing Eq. (12) requires solving a $n \times n$ generalized eigenvalue problem. The following proposition introduces the sketched version of this estimator, along with a procedure to compute it which instead requires the solution of a $m \times m$ eigenvalue problem. For $m \asymp \sqrt{n}$, which is enough to guarantee optimal learning rates with minimal assumptions (see Section 4), this represents a reduction from $O(n^3)$ to $O(n\sqrt{n})$ time.

Proposition 3.3 (Nyström RRR): *The Nyström RRR estimator can be written as*

$$\hat{A}_{m,\lambda}^{RRR} = [\![P_Y \hat{C}_{YX} P_X (P_X \hat{C} P_X + \lambda I)^{-1/2}]\!]_r (P_X \hat{C} P_X + \lambda I)^{-1/2}. \tag{13}$$

To compute it, solve the $m \times m$ eigenvalue problem

$$(K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} K_{Y,\tilde{Y}} K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} w_i = \sigma_i^2 w_i$$

for the first r eigenvectors $W_r = [w_1, \ldots, w_r]$, appropriately normalized. Then denoting $D_r := K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} W_r$ and $E_r := (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} K_{Y,\tilde{Y}} D_r$ it holds

$$\hat{A}_{m,\lambda}^{RRR} = \widetilde{\Phi}_Y D_r E_r^* \widetilde{\Phi}_X^*. \tag{14}$$

4 Learning bounds in operator norm for the sketched estimators

In this section, we state the main theoretical results showing that optimal rates for operator learning with KRR, PCR and RRR can be reached with Nyström estimators.

Assumptions We first make two assumptions on the space $\mathcal{H}$ used for the approximation, via its reproducing kernel k.

Assumption 4.1 (Bounded kernel): *There exists $K < \infty$ such that* $\operatorname{ess\,sup}_{\boldsymbol{x}\sim\pi}\|\phi(\boldsymbol{x})\| \leq K$.

Assumption 4.1 ensures that $\mathcal{H}$ is compactly embedded in L^2_π [57, Lemma 2.3], and we denote $\Phi_X^* : \mathcal{H} \to L^2_\pi$ the embedding operator which maps any function in $\mathcal{H}$ to its equivalence class π-almost everywhere in L^2_π.

Assumption 4.2 (Universal kernel): *The kernel k is universal, i.e.* $\operatorname{cl}(\operatorname{ran}(\Phi_X^*)) = L^2_\pi$.

We refer the reader to [56, Definition 4.52] for a definition of a universal kernel. The third assumption on the RKHS is related to the embedding property from Fischer and Steinwart [16], connected to the embedding of interpolation spaces. For a detailed discussion see Appendix A.3.

Assumption 4.3 (Embedding property): *There exists $\tau \in]0, 1]$ and $c_\tau > 0$ such that* $\operatorname{ess\,sup}_{\boldsymbol{x}\sim\pi}\|C_\lambda^{-1/2}\phi(\boldsymbol{x})\|^2 \leq c_\tau \lambda^{-\tau}$.

Next, we make an assumption on the decay of the spectrum of the covariance operator that is of paramount importance for derivation of optimal learning bounds. In the following, $\lambda_i(A)$ and $\sigma_i(A)$ always denote the eigenvalues and singular values of an operator A (in decreasing order).

Assumption 4.4 (Spectral decay): *There exists $\beta \in]0, \tau]$ and $c > 0$ such that $\lambda_i(C) \leq ci^{-1/\beta}$.*

This assumption is common in the literature, and we will see that the optimal learning rates depend on β. It implies the bound $d_{\text{eff}}(\lambda) := \operatorname{tr}(C_\lambda^{-1}C) \lesssim \lambda^{-\beta}$ on the effective dimension, which is a key quantity in the analysis (both statements are actually equivalent, see Appendix E.2). Note that $d_{\text{eff}}(\lambda) = \mathbf{E}_{\boldsymbol{x}\sim\pi}\|C_\lambda^{-1/2}\phi(\boldsymbol{x})\| \leq \operatorname{ess\,sup}_{\boldsymbol{x}\sim\pi}\|C_\lambda^{-1/2}\phi(\boldsymbol{x})\|$, and thus it necessarily holds $\beta \leq \tau$. For a Gaussian kernel, both β and τ can be chosen arbitrarily close to zero.

Finally, we make an assumption about the regularity of the problem itself. A common assumption occurring in the literature is that $\mathbf{E}[f(X_1) \,|\, X_0 = \cdot] \in \mathcal{H}$ for every $f \in \mathcal{H}$, meaning that one can define the Koopman operator directly on the space $\mathcal{H}$, i.e. the learning problem is *well-specified*. However, this assumption is often too strong. Following [27, D.1] we make a different assumption on the cross-covariance remarking that, irrespectively of the choice of RKHS, it holds true whenever the Koopman operator is self-adjoint (i.e. the dynamics is time-reversible).

Assumption 4.5 (Regularity of $\mathcal{K}_\pi$): *There exists $a > 0$ such that $C_{XY}C_{XY}^* \preccurlyeq a^2C^2$.*

Rates The risk can be decomposed as $\mathcal{R}_{\text{HS}}(A) = \mathcal{E}_{\text{HS}}(A) + \mathcal{R}_{\text{HS},0}$ where $\mathcal{R}_{\text{HS},0}$ is a constant and $\mathcal{E}_{\text{HS}}(A) := \|\mathcal{K}_\pi\Phi_X^* - \Phi_X^*A^*\|^2_{\text{HS}}$ corresponds to the excess risk (more details in Appendix B). Optimal learning bounds for the KRR estimator in the context of CME (i.e. in Hilbert-Schmidt norm) have been developed in [30] under Assumptions 4.1 to 4.4 in well-specified and misspecified settings. On the other hand, in the context of dynamical systems, Kostic et al. [26, Theorem 1] report the importance of *reduced rank estimators* that have a small excess risk in operator norm

$$\mathcal{E}(A) := \|\mathcal{K}_\pi\Phi_X^* - \Phi_X^*A^*\|^2_{\mathcal{H}\to L^2_\pi}. \tag{15}$$

The rationale behind considering the operator norm is that it allows to control the error of the eigenvalues approximation and thus of the KMD (3), (4) as discussed below. Optimal learning bounds in operator norm for KRR, PCR and RRR are established in [27]. In this work we show that the same optimal rates remain valid for the *Nyström* KRR, PCR and RRR estimators. According to [26] and [27] these operator norm bounds lead to reliable approximation of the Koompan mode decomposition of Eq. (4).

We now provide our main result.

Theorem 4.6 (Operator norm error for KRR, i.i.d. data): *Let assumptions 4.1 to 4.5 hold. Let $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{1\leq i\leq n}$ be i.i.d. samples, and let $P_Y = P_X$ be the projection induced by m Nyström landmarks drawn uniformly from $(\boldsymbol{x}_i)_{1\leq i\leq n}$ without replacement. Let $\lambda = c_\lambda n^{-1/(1+\beta)}$ where c_λ is a constant given in the proof, and assume $n \geq (c_\lambda/K^2)^{1+\beta}$. Then it holds with probability at least $1-\delta$*

$$\mathcal{E}(\hat{A}^{KRR}_{m,\lambda})^{1/2} \lesssim n^{-\frac{1}{2(1+\beta)}} \qquad \textit{provided} \qquad m \gtrsim \max(1, n^{\tau/(1+\beta)})\log(n/\delta).$$

The proof is provided in Appendix E.2, but essentially relies on a decomposition involving the terms $\|C_\lambda^{-1/2}(C_{YX} - \hat{C}_{YX})\|$, $\|C_\lambda^{-1/2}(C - \hat{C})\|$, $\|C_\lambda^{-1/2}(C - \hat{C})C_\lambda^{-1/2}\|$, as well as bounding

the quantity $\|P_X^{\perp} C^{1/2}\|$ where $P_X^{\perp}$ denotes the projection on the orthogonal of $\mathrm{ran}(P_X)$. All these terms are bounded using two variants of the Bernstein inequality. Note that our results can easily be extended to leverage score sampling of the landmarks by bounding term $\|P_X^{\perp} C^{1/2}\|$ by means of [49, Lemma 7]; the same rate could then be obtained using a smaller number m of Nyström points.

The rate $n^{-1/(2(1+\beta))}$ is known to be optimal (up to the log factor) in this setting by assuming an additional lower bound on the decay of the covariance's eigenvalues of the kind $\lambda_i(C) \gtrsim i^{-1/\beta}$, see [27, Theorem 7 in D.4]. One can see that without particular assumptions ($\beta = \tau = 1$), we only need the number m of inducing points to be of the order of $\Omega(\sqrt{n})$ in order to get an optimal rates. For τ fixed, this number increases when β decreases (faster decay of the covariance's spectrum), however note that the optimal rate depends on β and also improves in this case. The dependence in τ is particularly interesting, as for instance with a Gaussian kernel it is known that τ can be chosen arbitrarily closed to zero [30, 16]. In that case, the number m of inducing points can be taken on the order of $\Omega(\log n)$.

Note that a bound for the Nyström KRR estimator has been derived in Hilbert-Schmidt norm by Ahmad et al. [3]. Using the operator norm however allows to derive bounds on the eigenvalues (see discussion below), which is of paramount importance for practical applications. Moreover, we now provide a bound on the error of PCR and RRR estimators, which are not covered in [3].

Lemma 4.7 (Operator norm error for PCR and RRR, i.i.d. data): *Under the assumptions of Theorem 4.6, taking $\lambda = c_\lambda n^{-1/(1+\beta)}$ with c_λ as in Theorem 4.6, $n \geq (c_\lambda/K^2)^{1+\beta}$, and provided*

$$m \gtrsim \max(1, n^{\tau/(1+\beta)}) \log(n/\delta),$$

it holds with probability at least $1-\delta$

$$\mathcal{E}(\hat{A}^{RRR}_{m,\lambda})^{1/2} \lesssim c_{\mathrm{RRR}}\, n^{-\frac{1}{2(1+\beta)}}, \;\; \textit{for } r \textit{ s.t. } \sigma_{r+1}(\Phi_{Y|X}) < \min(\sigma_r(\Phi_{Y|X}), n^{-\frac{1}{2(1+\beta)}})$$
$$\textit{and} \quad \mathcal{E}(\hat{A}^{PCR}_{m})^{1/2} \lesssim c_{\mathrm{PCR}}\, n^{-\frac{1}{2(1+\beta)}}, \;\; \textit{for } r > n^{\frac{1}{\beta(1+\beta)}},$$

where $c_{\mathrm{RRR}} = (\sigma_r^2(\Phi_{Y|X}) - \sigma_{r+1}^2(\Phi_{Y|X}))^{-1}$ and $c_{\mathrm{PCR}} = (\sigma_r(\Phi_X) - \sigma_{r+1}(\Phi_X))^{-1}$ are the problem dependant constants.

Note that when rank of $\mathcal{K}_\pi$ is r, then there is no restriction on r for the RRR estimator, while for PCR the choice of r depends on the spectral decay property of the kernel. In general, if $r > n^{\frac{1}{\beta(1+\beta)}}$, then $\sigma_{r+1}(\Phi_{Y|X}) \leq \sigma_{r+1}(\Phi_X) \lesssim n^{-1/(2(1+\beta))}$, which implies that RRR estimator can achieve the same rate of PCR but with smaller rank. Again the rate is sharp (up to the log factor) in this setting [27].

Koopman mode decomposition According to [26, Theorem 1], working in operator norm allows us to bound the error of our estimators for dynamic mode decomposition, as well as to quantify how close the eigenpairs $(\hat{\lambda}_i, \hat{\varphi}_i)$ of an estimator $\hat{A}^*$ are to being eigenpairs of the Koopman operator. Namely, recalling that for function $\hat{\varphi}_i$, the corresponding candidate for Koopman eigenfunction in L^2_π space is $\Phi_X^* \hat{\varphi}_i$, one has $\|\mathcal{K}_\pi(\Phi_X^*\hat{\varphi}_i) - \hat{\lambda}_i(\Phi_X^*\hat{\varphi}_i)\|/\|\Phi_X^*\hat{\varphi}_i\| \leq \mathcal{E}(\hat{A})^{1/2}\|\hat{\varphi}_i\|/\|\Phi_X^*\hat{\varphi}_i\|$. While eigenvalue and eigenfunction learning rates were studied, under additional assumptions, in [27], where the operator norm error rates were determinant, here, in Section 5, we empirically show that the proposed estimators accurately learn the Koopman spectrum. We refer the reader to Appendix D for the details on computation of eigenvalues, eigenfunctions and KMD of an estimator in practice.

Dealing with non-i.i.d. data The previous results hold for i.i.d. data, which is not a very realistic assumption when learning from sampled trajectories. Our results can however easily be extended to β-mixing processes by considering random variables $Z_i = \sum_{j=1}^{k} X_{i+j}$ (thus representing portions of the trajectory) sufficiently separated in time to be nearly independent. We now consider a trajectory $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n+1}$ with $\boldsymbol{x}_1 \sim \pi$ and $\boldsymbol{x}_{t+1} \sim p(\boldsymbol{x}_t, \cdot)$ for $t \in [1, n]$, and use Lemma J.8 (re-stated from [26]) which allows to translate concentration results on the Z_i to concentration on the X_i by means of the β-mixing coefficients defined as $\beta_X(k) := \sup_{B \in \mathcal{B}\otimes\mathcal{B}} \big|\rho_k(B) - (\pi \times \pi)(B)\big|$ where ρ_k denotes the joint probability of (X_t, X_{t+k}). Using this result the concentration results provided in appendix can thus be generalied to the β-mixing setting, and apart from logarithmic dependencies we essentially obtain similar results to the i.i.d. setting except that the sample size n is replaced by $p \approx n/(2k)$.

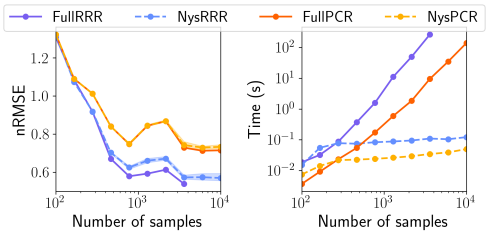


Figure 1: Full and Nyström estimators trained on L63 with increasing n. Error (*left*) and running time (*right*) are plotted to show efficiency gains without accuracy loss with the Nyström approximation. RBF($\sigma = 3.5$) kernel, $r = 25$ principal components and $m = 250$ inducing points.

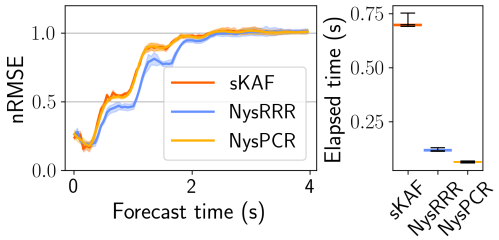


Figure 2: Nyström and sKAF estimators trained on L63 for increasing forecast horizons; the error (*left*) and overall running times (*right)* are shown. We used a RBF kernel with $\sigma = 3.5$, $r = 50$, $m = 250$ (for Nyström methods) and $\sqrt{n} \log n$ random features (for sKAF).

5 Experimental validation

In this section we show how the estimators proposed in section 3 perform in various scenarios, ranging from synthetic low dimensional ODEs to large-scale molecular dynamics simulations. The code for reproducing all experiments is available online. Our initial aim is to demonstrate the speed of NysPCR and NysRRR, compared to the recently proposed alternative Streaming KAF (sKAF) [18]. Then we show that their favorable scaling properties make it possible to train on large molecular dynamics datasets without any subsampling. In particular we run a metastability analysis of the alanine dipeptide and the Trp-cage protein, showcasing the accuracy of our models' eigenvalue and eigenfunction estimates, as well as their efficiency on massive datasets ($> 500\,000$ points)

Efficiency Benchmarks on Lorenz '63 The chaotic Lorenz '63 system [33] consists of 3 ODEs with no measurement noise. With this toy dynamical system we can easily compare the Nyström estimators to two alternatives: 1. the corresponding *exact* estimators and 2. the sKAF algorithm which also uses randomized linear algebra to improve the efficiency of PCR. In this setting we sample long trajectories from the system, keeping the first points for training (the number of training points varies for the first experiment, and is fixed to $10\,000$ for the second, see fig. 2), and the subsequent ones for testing. In Figure 1 we compare the run-time and accuracy with of NysPCR and NysRRR versus their full counterparts. To demonstrate the different scaling regimes we fix the number of inducing points to 250 and increase the number of data points n. The accuracy of the two solvers (as measured with the normalized RMSE metric (nRMSE) [18] on the first variable) is identical for PCR and close for RRR, but the running time of the approximate solvers increases much slower with n than that of the exact solvers. Each experiment is repeated 20 times to display error bars over the choice of Nyström centers. In the second experiment, shown in fig. 2, we reproduce the setting of [18] by training at increasingly long forecast horizons. Plotting the nRMSE we verify that sKAF and NysPCR converge to very similar accuracy values, although NysPCR is approximately 10 times faster. NysRRR instead offers slightly better accuracy, at the expense of a higher running time compared to NysPCR. Error bars are the standard deviation of nRMSE over 5 successive test sets with $10\,000$ points each.

Molecular dynamics datasets An important application of Koopman operator theory is in the analysis of molecular dynamics (MD) datasets, where the evolution of a molecule's atomic positions as they evolve over time is modelled. Interesting systems are very high dimensional, with hundreds or thousands of atoms. Furthermore, trajectories are generated at very short time intervals (< 1 ns) but interesting events (e.g. protein folding/unfolding) occur at timescales on the order of at least $10\,\mu$s, so that huge datasets are needed to have a few samples of the rare events. The top eigenfunctions of the Koopman operator learned on such trajectories can be used to project the high-dimensional state space onto low-dimensional coordinates which capture the long term, slow dynamics.

We take three 250 ns long simulations sampled at 1 ps of the alanine dipeptide [60], which is often taken as a model system for molecular dynamics [43, 42]. We use the pairwise distances between heavy atoms as features, yielding a 45-dimensional space. We train a NysRRR model with $10\,000$ centers on top of the full dataset ($449\,940$ points are used for training, the rest for validation and testing) with lag time 100 ps, and recover a 2-dimensional representation which correlates well

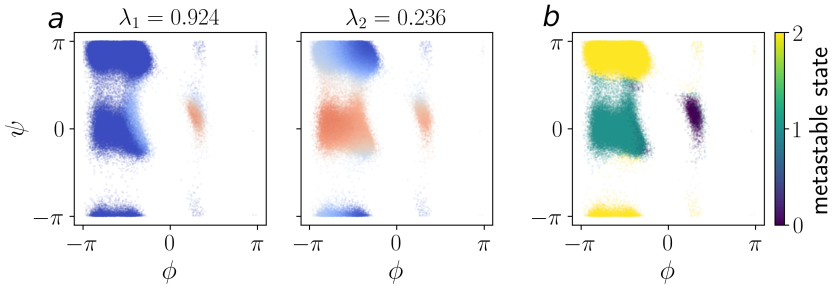


Figure 3: Dynamics of the alanine dipeptide (lag-time 100), Nyström RRR model. On the left the first two non-constant eigenfunctions, overlaid in color on the Ramachandran plot which fully describes the metastable states. On the right the three states of a PCCA+ model trained on the eigenfunctions.

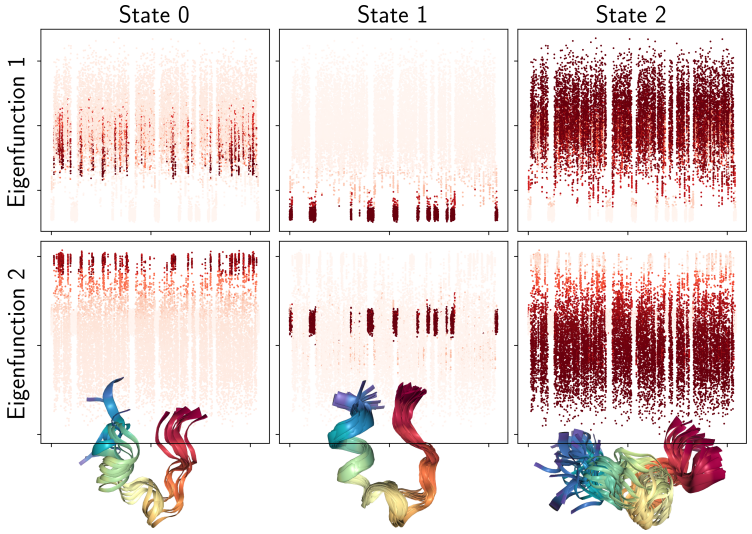


Figure 4: First eigenfunctions for Trp-cage dynamics, colored according to the membership probability for each state in a PCCA+ model. The bottom insets show a few overlaid structures from each state. The first eigenfunction exhibits a strong linear separation between state 1 (folded) and the other states. The second separates between state 0 (partially folded) ant the rest. NysRRR model trained with $m = 5000$, $r = 10$, RBF($\sigma = 0.02$) kernel, $\lambda = 10^{-10}$.

with the ϕ, ψ backbone dihedral angles of the molecule, known to capture all relevant long-term dynamics. Figure 3a shows the top two eigenfunctions overlaid onto ϕ, ψ, the first separates the slowest transition between low and high ϕ; the second separates low and high ψ. The implied time-scales from the first two non-trivial eigenvalues are $1262\,\mathrm{ps}$ and $69\,\mathrm{ps}$, which are close to the values reported by Nüske et al. [43] ($1400\,\mathrm{ps}$ and $70\,\mathrm{ps}$) who used a more complex post-processing procedure to identify time-scales. We then train a PCCA+ [14] model on the first three eigenfunctions to obtain three states, as shown in fig. 3b. PCCA+ acts on top of a fine clustering (in our case obtained with k-means, $k = 50$), to find the set of maximally stable states by analyzing transitions between the fine clusters. The coarse clusters clearly correspond to the two transitions described above.

Finally we take a $208\,\mu\mathrm{s}$ long simulation of the fast-folding Trp-cage protein [32], sampled every $0.2\,\mathrm{ns}$. Again, the states are the pairwise distances between non-hydrogen atoms belonging to the protein, in $10\,296$ dimensions. A NysRRR model is trained on $626\,370$ points, using 5000 centers in approximately 10 minutes. Note that without sketching this would be a completely intractable problem. Using a lag-time of $10\,\mathrm{ns}$ we observe a spectral gap between the third and fourth eigenvalues, hence we train a PCCA+ model on the first 3 eigenfunctions to obtain the states shown in fig. 4. The first non-trivial Koopman eigenvector effectively distinguishes between the folded (state 1) and unfolded states as is evident from the first row of fig. 4. The second one instead can be used to identify a partially folded state of the protein (state 0), as can be seen from the insets in fig. 4.

6 Conclusions

We introduced three efficient kernel-based estimators of the Koopman operator relying on random projections, and provided a bound on their excess risk in operator norm – which is of paramount importance to control the accuracy of Koopman mode decomposition. Random projections allow to process efficiently even the longest trajectories, and these gains come for free as our estimators still enjoy optimal theoretical learning rates. We leave for future work the refinement our analysis under e.g. an additional source condition assumption or in the misspecified setting. Another future research direction shall be to devise ways to further reduce the computational complexity of the estimators.

7 Acknowledgements

This paper is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819789). L. R. acknowledges the financial support of the European Research Council (grant SLING 819789), the AFOSR projects FA9550-18-1-7009, FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), the EU H2020-MSCA-RISE project NoMADS - DLV-777826, and the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. M. P., V. K. and P. N. acknowledge financial support from PNRR MUR project PE0000013-FAIR and the European Union (Projects 951847 and 101070617).

References

[1] Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models, 2023. `arXiv:2302.02605 [cs.LG]`.

[2] Thomas N. E. Greville (auth.) Adi Ben-Israel. *Generalized Inverses: Theory and Applications*. CMS Books in Mathematics. Springer, 2 edition, 2003.

[3] Tamim El Ahmad, Luc Brogat-Motte, Pierre Laforgue, and d'Alché-Buc Florence. Sketch In, Sketch Out: Accelerating both Learning and Inference for Structured Prediction with Kernels, 2023. `arxiv:2302.10128`.

[4] Romeo Alexander and Dimitrios Giannakis. Operator-theoretic framework for forecasting nonlinear time series with kernel analog techniques. *Physica D: Nonlinear Phenomena*, 409, 2020. doi: https://doi.org/10.1016/j.physd.2020.132520.

[5] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. *Journal of Machine Learning Research*, 30, 2013.

[6] Peter J. Baddoo, Benjamin Herrmann, Beverley J. McKeon, and Steven L. Brunton. Kernel learning for robust dynamic mode decomposition: linear and nonlinear disambiguation optimization. *Proceedings of the Royal Society A*, 2022. doi: http://doi.org/10.1098/rspa.2021.0830.

[7] Steven L. Brunton, Marko Budišić, Eurika Kaiser, and J. Nathan Kutz. Modern Koopman theory for dynamical systems, 2021. `arxiv:2102.12086`.

[8] Andrea Caponnetto and Ernesto De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7(3):331 – 368, 2007. doi: 10.1007/s10208-006-0196-8.

[9] Antoine Chatalic, Nicolas Schreuder, Lorenzo Rosasco, and Alessandro Rudi. Nyström Kernel Mean Embeddings. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3006–3024. PMLR, 2022.

[10] Carlo Ciliberto, Lorenzo Rosasco, and Alessandro Rudi. A Consistent Regularization Approach for Structured Prediction. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[11] Carlo Ciliberto, Lorenzo Rosasco, and Alessandro Rudi. A general framework for consistent structured prediction with implicit loss embeddings. *Journal of Machine Learning Research*, 21 (1):98:3852–98:3918, 2022.

[12] G. Da Prato and J. Zabczyk. *Ergodicity for Infinite Dimensional Systems*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1996. doi: 10.1017/CBO9780511662829.

[13] Michael Dellnitz and Oliver Junge. On the approximation of complicated dynamical behavior. *SIAM Journal on Numerical Analysis*, 36(2):491–515, 1999. doi: 10.1137/S0036142996313002.

[14] Peter Deuflhard and Marcus Weber. Robust perron cluster analysis in conformation dynamics. *Linear Algebra and its Applications*, 398:161–184, 2005. doi: https://doi.org/10.1016/j.laa.2004.10.026.

[15] Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13(1), 2012.

[16] Simon Fischer and Ingo Steinwart. Sobolev norm learning rates for regularized least-squares algorithms. *Journal of Machine Learning Research*, 21(1):8464–8501, 2020.

[17] Gary Froyland, Georg A. Gottwald, and Andy Hammerlindl. A computational method to extract macroscopic variables and their dynamics in multiscale systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1816–1846, 2014. doi: 10.1137/130943637.

[18] Dimitris Giannakis, Amelia Henriksen, Joel A. Tropp, and Rachel Ward. Learning to Forecast Dynamical Systems from Streaming Data, 2021. `arxiv:2109.09703`.

[19] Alex Gittens and Michael W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. *Journal of Machine Learning Research*, 17:3977–4041, 2016.

[20] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171 – 1220, 2008. doi: 10.1214/009053607000000677.

[21] Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5(2):248–264, 1975. doi: https://doi.org/10.1016/0047-259X(75)90042-1.

[22] Stefan Klus, Péter Koltai, and Christof Schütte. On the numerical approximation of the Perron-Frobenius and Koopman operator. *Journal of Computational Dynamics*, 3(1):51–79, 2016. ISSN 2158-2491. doi: 10.3934/jcd.2016003.

[23] Stefan Klus, Ingmar Schuster, and Krikamol Muandet. Eigendecompositions of transfer operators in reproducing kernel hilbert spaces. *Journal of Nonlinear Science*, 30(1):283–315, 2020. doi: 10.1007/s00332-019-09574-z.

[24] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931. doi: 10.1073/pnas.17.5.315.

[25] B. O. Koopman and J. v. Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences*, 18(3):255–263, 1932. doi: 10.1073/pnas.18.3.255.

[26] Vladimir Kostic, Pietro Novelli, Andreas Maurer, Carlo Ciliberto, Lorenzo Rosasco, and Massimiliano Pontil. Learning Dynamical Systems via Koopman Operator Regression in Reproducing Kernel Hilbert Spaces, 2022. `arXiv:2205.14027 [cs.LG]`.

[27] Vladimir Kostic, Karim Lounici, Pietro Novelli, and Massimiliano Pontil. Koopman operator learning: Sharp spectral rates and spurious eigenvalues, 2023. `arXiv:2302.02004 [cs.LG]`.

[28] J. Nathan Kutz, Steven L. Brunton, Binghi W. Brunton, and Joshua Proctor. *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*. SIAM, 2016.

[29] Qianxiao Li, Felix Dietrich, Erik M. Bollt, and Ioannis G. Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10), 2017. doi: 10.1063/1.4993854.

[30] Zhu Li, Dimitri Meunier, Mattes Mollenhauer, and Arthur Gretton. Optimal Rates for Regularized Conditional Mean Embedding Learning, 2022. `arXiv:2208.01711`.

[31] Junhong Lin and Volkan Cevher. Optimal convergence for distributed learning with stochastic gradient methods and spectral algorithms. *Journal of Machine Learning Research*, 21(147): 1–63, 2020.

[32] Kresten Lindorff-Larsen, Stefano Piana, Ron O. Dror, and David E. Shaw. How fast-folding proteins fold. *Science*, 334(6055):517–520, 2011. doi: 10.1126/science.1208351.

[33] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20(2): 130 – 141, 1963. doi: https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[34] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1), 2018. doi: 10.1038/s41467-018-07210-0.

[35] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. In *Advances in Neural Information Processing Systems 32*, 2020.

[36] Igor Mezič. *On the geometrical and statistical properties of dynamical systems: Theory and applications*. PhD thesis, ProQuest LLC, California Institute of Technology, 1994.

[37] Igor Mezić. Koopman operator, geometry, and learning of dynamical systems. *Notices of the American Mathematical Society*, 68(7):1087–1105, 2021.

[38] L. Molgedey and H. G. Schuster. Separation of a mixture of independent signals using time delayed correlations. *Phys. Rev. Lett.*, 72:3634–3637, 1994. doi: 10.1103/PhysRevLett.72.3634.

[39] Mattes Mollenhauer, Ingmar Schuster, Stefan Klus, and Christof Schütte. Singular value decomposition of operators on reproducing kernel hilbert spaces. In *Advances in Dynamics, Optimization and Computation*, pages 109–131, 2020.

[40] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, and Bernhard Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1-2):1–141, 2017. doi: 10.1561/2200000060.

[41] Frank Noé and Feliks Nüske. A variational approach to modeling slow processes in stochastic dynamical systems. *Multiscale Modeling & Simulation*, 11(2):635–655, 2013. doi: 10.1137/110858616.

[42] Feliks Nüske, Bettina G. Keller, Guillermo Pérez-Hernández, Antonia S. J. S. Mey, and Frank Noé. Variational approach to molecular kinetics. *Journal of Chemical Theory and Computation*, 10(4):1739 – 1752, 2014. doi: 10.1021/ct4009156.

[43] Feliks Nüske, Hao Wu, Jan-Hendrik Prinz, Christoph Wehmeyer, Cecilia Clementi, and Frank Noé. Markov state models from short non-equilibrium simulations — analysis and correction of estimation bias. *The Journal of Chemical Physics*, 146(9), 2017. doi: 10.1063/1.4976518.

[44] I. F. Pinelis and A. I. Sakhanenko. Remarks on inequalities for large deviation probabilities. *Theory of Probability & Its Applications*, 30(1):143–148, 1986. doi: 10.1137/1130013.

[45] Guillermo Pérez-Hernández, Fabian Paul, Toni Giorgino, Gianni De Fabritiis, and Frank Noé. Identification of slow molecular order parameters for markov model construction. *The Journal of Chemical Physics*, 139(1), 07 2013. doi: 10.1063/1.4811489.

[46] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *NeurIPS 20*, 2008.

[47] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems 21*, 2009.

[48] Clarence W. Rowley, Igor Mezič, Shervin Bagheri, Philipp Schlatter, and Dan S. Henningson. Spectral analysis of nonlinear flows. *Journal of Fluid Mechanics*, pages 115–127, 2009. doi: 10.1017/S0022112009992059.

[49] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. Less is more: Nyström computational regularization. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 1657–1665, 2015.

[50] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. FALKON: An optimal large scale kernel method. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.

[51] Alessandro Rudi, Daniele Calandriello, Luigi Carratino, and Lorenzo Rosasco. On fast leverage score sampling and optimal learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[52] Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010. doi: 10.1017/S0022112010001217.

[53] Christian R. Schwantes and Vijay S. Pande. Modeling molecular kinetics with tICA and the kernel trick. *Journal of Chemical Theory and Computation*, 11(2):600–608, 2015. doi: 10.1021/ct5007357.

[54] Alex J. Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In *ICML 17*, 2000.

[55] Le Song, Jonathan Huang, Alex Smola, and Kenji Fukumizu. Hilbert space embeddings of conditional distributions with applications to dynamical systems. In *Proceedings of the 26th Annual International Conference on Machine Learning*, page 961–968, 2009. doi: 10.1145/1553374.1553497.

[56] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Science & Business Media, 2008. URL `https://link.springer.com/book/10.1007/978-0-387-77242-4`.

[57] Ingo Steinwart and Clint Scovel. Mercer's theorem on general domains: On the interaction between measures, kernels, and RKHSs. *Constructive Approximation*, 35(3):363–417, 2012.

[58] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, page 1130–1140, 2017.

[59] Jonathan H. Tu, Clarence W. Rowley, Dirk M. Luchtenburg, Steven L. Brunton, and J. Nathan Kutz. On dynamic mode decomposition: Theory and applications. *Journal of Computational Dynamics*, 1(2):391–421, 2014. doi: 10.3934/jcd.2014.1.391.

[60] Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of Chemical Physics*, 148(24), 2018. doi: 10.1063/1.5011399.

[61] Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *NeurIPS 13*, 2001.

[62] Mattew O. Williams, Ioannis G. Kevrekidis, and Clarence W. Rowley. A data–driven approximation of the Koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307 – 1346, 2015. doi: 10.1007/s00332-015-9258-5.

[63] Matthew O. Williams, Clarence W. Rowley, and Ioannis G. Kevrekidis. A kernel-based method for data-driven Koopman spectral analysis. *Journal of Computational Dynamics*, 2(2):247–265, 2015. ISSN 2158-2491. doi: 10.3934/jcd.2015005.

[64] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs Random Fourier Features: A theoretical and empirical comparison. In *Advances in Neural Information Processing Systems 24*, 2012.

[65] Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, 2019. doi: 10.23919/ACC.2019.8815339.

[66] Vadim Yurinsky. *Sums and Gaussian Vectors*. Lecture Notes in Mathematics 1617. Springer-Verlag Berlin Heidelberg, 1 edition, 1995.

A Setting and notations

A.1 Operators and notations

We define the following operators:

- $\Phi_X : L^2_\pi \to \mathcal{H}$, defined by $\Phi_X f = \int_{\mathcal{X}} f(x)\phi(x)\,\mathrm{d}\pi(x)$ for any $f \in L^2_\pi$.
- $\Phi^*_X : \mathcal{H} \to L^2_\pi$, defined by $\Phi^*_X h = \langle h, \phi(\cdot)\rangle_{\mathcal{H}}$ for any $h \in \mathcal{H}$ (i.e. the embedding operator mapping a function to its π-equivalence class in L^2_π).
- $\Phi_{Y|X} : L^2_\pi \to \mathcal{H}$, defined by $\Phi_{Y|X} = \Phi_X \mathcal{K}^*_\pi$.
- $\Phi^*_{Y|X} : \mathcal{H} \to L^2_\pi$, defined by $\Phi^*_{Y|X} = \mathcal{K}_\pi \Phi^*_X$.
- $C : \mathcal{H} \to \mathcal{H}$ defined as $C = \mathbf{E}_{x\sim\pi}\phi(x) \otimes \phi(x) = \Phi_X \Phi^*_X$, satisfying $\mathrm{tr}(C) \leq K^2$. Note that under our assumptions, this also corresponds to the covariance of Y.
- $C_{XY} := \mathbf{E}_{(x,y)\sim\rho}\phi(x) \otimes \phi(y) = \Phi_X \Phi^*_{Y|X}$.

As well as the following discretized variants:

- $\hat{\Phi}_X : \mathbb{R}^n \to \mathcal{H}$, defined by $\hat{\Phi}_X v = \sum_{i=1}^n v_i \phi(x_i)$ for any $v = [v_1, \ldots, v_n] \in \mathbb{R}^n$
- $\hat{\Phi}^*_X : \mathcal{H} \to \mathbb{R}^n$, defined by $\hat{\Phi}^*_X h = [\langle \phi(x_1), h\rangle_{\mathcal{H}}, \ldots, \langle \phi(x_n), h\rangle_{\mathcal{H}}]^T$ for any $h \in \mathcal{H}$
- $\hat{\Phi}_{Y|X} : \mathbb{R}^n \to \mathcal{H}$, defined by $\hat{\Phi}_{Y|X} v = \sum_{i=1}^n v_i \phi(y_i)$ for any $v = [v_1, \ldots, v_n] \in \mathbb{R}^n$.
- $\hat{\Phi}^*_{Y|X} : \mathcal{H} \to \mathbb{R}^n$, defined by $\hat{\Phi}^*_{Y|X} h = [\langle \phi(y_1), h\rangle_{\mathcal{H}}, \ldots, \langle \phi(y_n), h\rangle_{\mathcal{H}}]^T$ for any $h \in \mathcal{H}$.
- $\hat{C} = \frac{1}{n}\hat{\Phi}_X \hat{\Phi}^*_X = \frac{1}{n}\sum_{i=1}^n \phi(x_i) \otimes \phi(x_i) \in \mathcal{L}(\mathcal{H})$ is the empirical covariance.

The Nyström discretized operators are obtained by applying the kernel map to $m \ll n$ inducing points $\{\tilde{\boldsymbol{x}}_j\}_{j=1}^m \subset \{\boldsymbol{x}_j\}_{j=1}^n$ and $\{\tilde{\boldsymbol{y}}_j\}_{j=1}^m \subset \{\boldsymbol{y}_j\}_{j=1}^n$:

- $\widetilde{\Phi}_X : \mathbb{R}^m \to \mathcal{H}$ such that $\widetilde{\Phi}_X w = \sum_{j=1}^m w_j \phi(\tilde{\boldsymbol{x}}_j)$.
- $\widetilde{\Phi}_Y : \mathbb{R}^m \to \mathcal{H}$ such that $\widetilde{\Phi}_Y w = \sum_{j=1}^m w_j \phi(\tilde{\boldsymbol{y}}_j)$.

Furthermore denote by P_X and P_Y the orthogonal projections onto $\mathrm{span}\,\widetilde{\Phi}_X$ and $\mathrm{span}\,\widetilde{\Phi}_Y$ respectively.

One important quantity to derive the rates is the so-called effective dimension, defined as

$$d_{\text{eff}}(\lambda) := \mathrm{tr}(C_\lambda^{-1} C).$$

where $C_\lambda := C + \lambda I$.

A.2 Conditional mean embedding

For any $x \in \mathcal{X}$, we denote $\mu_p(x)$ the conditional mean embedding associated to the transition kernel defined as

$$\mu_p(x) := \mathbf{E}\big[\phi(X_{t+1}) | X_t = x\big] = \int \phi(y) p(x, \mathrm{d}y)$$

The following lemma provides a characterization of $\Phi^*_{Y|X}$ in terms of the conditional mean embedding.

Lemma A.1: *We have the following relations:*

$$\Phi_{Y|X} f = \int_{\mathcal{X}} f(x)\mu_p(x)\,\mathrm{d}\pi(x), \quad f \in L^2_\pi \tag{16}$$

$$(\Phi^*_{Y|X} f)(x) = \langle f, \mu_p(x)\rangle, \quad f \in \mathcal{H} \tag{17}$$

$$\Phi_{Y|X}\Phi^*_{Y|X} = \mathbf{E}_{x\sim\pi}\mu_p(x) \otimes \mu_p(x) \tag{18}$$

Proof of Lemma A.1: For the first property:

$$(\Phi^*_{Y|X} f)(x) = (\mathcal{K}_\pi(\Phi^*_X f))(x) \tag{19}$$

$$= \int (\Phi^*_X f)(y) p(x, \mathrm{d}y) \tag{20}$$

$$= \int f(y) p(x, \mathrm{d}y) \tag{21}$$

$$= \langle f, \int \phi(y) p(x, \mathrm{d}y) \rangle = \langle f, \mu_p(x) \rangle \tag{22}$$

where we used that f and $\Phi^*_X f$ coincide π-almost everywhere. The second property is a direct consequence of the definition of the adjoint. For (18), we simply use (17) and the definition of $\Phi_{Y|X}$ to get

$$\Phi_{Y|X}(\Phi^*_{Y|X} f) = \int \langle f, \mu_p(z) \rangle \mu_p(z) \, \mathrm{d}\pi(z) = \left(\int \mu_p(z) \mu_p(z)^* \, \mathrm{d}\pi(z) \right) f.$$

□

A.3 Power spaces

We now define the α-power space $[\mathcal{H}]^\alpha_\pi$ in order to provide some intuition regarding Assumption 4.3. By Assumption 4.1, $\mathrm{tr}(C) = \int \mathrm{tr}(\phi(x) \otimes \phi(x)) \, \mathrm{d}\pi(x) \leq K^2$ and thus C is trace-class (and compact). By [16], there exists a non-increasing summable sequence $(\mu_i)_{i \in I}$ for an at most countable index set I, a family $(e_i)_{i \in I} \in \mathcal{H}$ s.t. $(\Phi^*_X e_i)_{i \in I}$ is an orthonormal basis of $\overline{\mathrm{span}\, \Phi^*_X} \subseteq L^2_\pi$ and $(\mu_i^{1/2} e_i)_{i \in I}$ is an orthonormal basis of $(\ker \Phi^*_X)^\perp \subseteq \mathcal{H}$ such that

$$C = \sum_{i \in I} \mu_i \langle \cdot, \mu_i^{1/2} e_i \rangle_{\mathcal{H}} \mu_i^{1/2} e_i.$$

For $\alpha \geq 0$, we now define the α-power space as

$$[\mathcal{H}]^\alpha_\pi := \left\{ \sum_{i \in I} a_i \mu_i^{\alpha/2} \Phi^*_X e_i \;\middle|\; (a_i)_{i \in I} \in \ell_2(I) \right\} \subseteq L^2_\pi$$

equipped with norm

$$\left\| \sum_{i \in I} a_i \mu_i^{\alpha/2} \Phi^*_X e_i \right\|_{[\mathcal{H}]^\alpha_\pi} := \|(a_i)_{i \in I}\|_{\ell_2(I)}.$$

We can now make the following assumption regarding the embedding of the power spaces into L^∞_π.

Assumption A.2 (Embedding): *There exists $\tau \in [\beta, 1]$ such that $c_\tau := \|[\mathcal{H}]^\tau_\pi \hookrightarrow L^\infty_\pi\|^2 < \infty$.*

We stress that Assumption A.2 implies in particular Assumption 4.3, and is a common assumption in the literature, see for instance [16].

B Expression of the risk

We have the following risk decomposition.

Lemma B.1: *The risk can alternatively be written*

$$\begin{aligned}
\mathcal{R}_{\text{HS}}(A) &= \mathbf{E}_{(x,y)\sim\rho}\|\phi(y) - A\phi(x)\|^2 \\
&= \mathcal{R}_{\text{HS},0} + \mathcal{E}_{\text{HS}}(A) \\
\textit{where} \quad \mathcal{R}_{\text{HS},0} &:= \|\Phi_X\|_{\text{HS}}^2 - \|\Phi_{Y|X}\|_{\text{HS}}^2 \\
&= \int \|\mu_p(x) - \phi(y)\|^2 \,\mathrm{d}\rho(x,y) \\
\textit{and} \quad \mathcal{E}_{\text{HS}}(A) &:= \|\Phi_{Y|X} - A\Phi_X\|_{\text{HS}}^2 \\
&= \int \|\mu_p(x) - A\phi(x)\|^2 \,\mathrm{d}\pi(x).
\end{aligned}$$

where $\inf_{A\in\text{HS}(\mathcal{H})} \mathcal{E}_{\text{HS}}(A) = 0$*, and thus we interpret* $\mathcal{E}_{\text{HS}}$ *as the excess risk.*

Proof of Lemma B.1: Let $(h_i)_{i\in\mathbb{N}}$ be an orthonormal basis of $\mathcal{H}$. Then

$$\begin{aligned}
\mathcal{E}_{\text{HS}}(A) &:= \|\Phi_{Y|X} - A\Phi_X\|_{\text{HS}}^2 \\
&= \sum_{i\in\mathbb{N}} \|\Phi_{Y|X}^* h_i - \Phi_X^* A^* h_i\|_{L_\pi^2}^2 \\
&= \sum_{i\in\mathbb{N}} \int ((\Phi_{Y|X}^* h_i)(x) - \langle A^* h_i, \phi(x)\rangle_{\mathcal{H}})^2 \,\mathrm{d}\pi(x) \\
\text{(by (17))} \quad &= \sum_{i\in\mathbb{N}} \int (\langle h_i, \mu_p(x)\rangle_{\mathcal{H}} - \langle h_i, A\phi(x)\rangle_{\mathcal{H}})^2 \,\mathrm{d}\pi(x) \\
&= \int \|\mu_p(x) - A\phi(x)\|^2 \,\mathrm{d}\pi(x).
\end{aligned}$$

It holds

$$\begin{aligned}
\mathcal{R}_{\text{HS},0} &= \int \|\mu_p(x) - \phi(y)\|^2 \,\mathrm{d}\rho(x,y) \\
&= \int \Big(\|\mu_p(x)\|^2 - 2\big\langle \mu_p(x), \phi(y)\big\rangle_{\mathcal{H}} + \operatorname{tr} \phi(y)\phi(y)^* \Big) \,\mathrm{d}\rho(x,y) \\
&= \int \|\mu_p(x)\|^2 \,\mathrm{d}\pi(x) - 2\int \left\langle \mu_p(x), \int \phi(y) p(x,\mathrm{d}y) \right\rangle_{\mathcal{H}} \mathrm{d}\pi(x) + \int\int \operatorname{tr} \phi(y)\phi(y)^* p(x,\mathrm{d}y) \,\mathrm{d}\pi(x) \\
&\overset{(i)}{=} -\int \|\mu_p(x)\|^2 \,\mathrm{d}\pi(x) + \int \operatorname{tr}\big(\phi(y)\phi(y)^*\big) \,\mathrm{d}\pi(y) \\
&= -\operatorname{tr}\left(\int \mu_p(x)\mu_p(x)^* \,\mathrm{d}\pi(x) \right) + \operatorname{tr}(C) \\
&= -\operatorname{tr}\Big(\Phi_{Y|X}\Phi_{Y|X}^* \Big) + \operatorname{tr}(\Phi_X \Phi_X^*)
\end{aligned}$$

where we used the invariance property of π in (i) and Lemma A.1 for the last inequality. Then one can easily check that the sum of both corresponds to the full risk defined in (5):

$$\begin{aligned}
\mathcal{R}_{\text{HS},0} + \mathcal{E}_{\text{HS}}(A) &= \int \|\mu_p(x) - \phi(y)\|^2 \,\mathrm{d}\rho(x,y) + \int \|\mu_p(x) - A\phi(x)\|^2 \,\mathrm{d}\pi(x) \\
&= \int \Big(\|\mu_p(x)\|^2 - 2\langle \mu_p(x), \int \phi(y) p(x, \mathrm{d}y)\rangle + \int \|\phi(y)\|^2 p(x, \mathrm{d}y) \Big) \,\mathrm{d}\pi(x) \\
&\quad + \int \Big(\|\mu_p(x)\|^2 - 2\langle \mu_p(x), A\phi(x)\rangle + \|A\phi(x)\|^2 \Big) \,\mathrm{d}\pi(x) \\
&= \int \Big(\int \|\phi(y)\|^2 p(x, \mathrm{d}y) - 2\langle \int \phi(y) p(x, \mathrm{d}y), A\phi(x)\rangle + \|A\phi(x)\|^2 \Big) \,\mathrm{d}\pi(x) \\
&= \int \Big(\|\phi(y)\|^2 - 2\langle \phi(y), A\phi(x)\rangle + \|A\phi(x)\|^2 \Big) \,\mathrm{d}\rho(x,y) \\
&= \int \|\phi(y) - A\phi(x)\|^2 \,\mathrm{d}\rho(x,y) = \mathcal{R}_{\text{HS}}(A).
\end{aligned}$$

□

C Expression of the estimators

In this section we give proofs of propositions 3.1 to 3.3 on how to efficiently compute the Nyström estimators.

For all three – KRR, PCR and RRR – estimators, the starting point is their respective *full* estimator which can be derived by following the first-order optimality criterion for the following minimization problems

$$\textbf{Full KRR:} \qquad \hat{A}_\lambda^{\text{KRR}} = \operatorname*{arg\,min}_{A \in \mathcal{H} \to \mathcal{H}} \|\hat{\Phi}_{Y|X} - A\hat{\Phi}_X\|_{\text{HS}}^2 + \lambda \|A\|_{\text{HS}}^2 \tag{23}$$

$$\textbf{Full PCR:} \qquad \hat{A}^{\text{PCR}} = \operatorname*{arg\,min}_{A \in \mathcal{H} \to \mathcal{H}} \|\hat{\Phi}_{Y|X} - A\Pi_r\hat{\Phi}_X\|_{\text{HS}}^2 \tag{24}$$

$$\textbf{Full RRR:} \qquad \hat{A}_\lambda^{\text{RRR}} = \operatorname*{arg\,min}_{A \in \mathcal{H} \to \mathcal{H} : \mathrm{rk}(A) \leq r} \|\hat{\Phi}_{Y|X} - A\hat{\Phi}_X\|_{\text{HS}}^2 + \lambda \|A\|_{\text{HS}}^2 \tag{25}$$

where Π_r is the orthogonal projection onto the top-r eigenvectors of $\hat{C}$.

To derive the Nyström estimators, we project the embedded data $\hat{\Phi}_X$, $\hat{\Phi}_{Y|X}$ onto the span of the embedded inducing points – $P_X\hat{\Phi}_X$, $P_Y\hat{\Phi}_{Y|X}$ – and then express the resulting estimators as $\widetilde{\Phi}_Y W \widetilde{\Phi}_X^*$ with $W \in \mathbb{R}^{m \times m}$. This form is particularly useful for later computing forecasts, eigenfunctions and Koopman modes with the estimator. In particular the following equalities for the projection (shown here for P_X but equivalently exist for P_Y)

$$P_X = P_X P_X = \widetilde{\Phi}_X (\widetilde{\Phi}_X^* \widetilde{\Phi}_X)^\dagger \widetilde{\Phi}_X^* = \widetilde{\Phi}_X^{*\dagger} \widetilde{\Phi}_X^* = \widetilde{\Phi}_X \widetilde{\Phi}_X^\dagger,$$

and the characterization of P_X through the SVD of $\widetilde{\Phi}_X = U\Sigma V^*$, such that $P_X = UU^*$.

C.1 Nyström KRR

We begin with the Nyström KRR estimator, providing an alternative but equivalent description in lemma C.1.

Lemma C.1 (Expression of the KRR regularization): *Let U be such that $P_X = UU^*$, $U^*U = I$. Then it holds*

$$g_{\text{KRR}}(\hat{C}) := P_X (P_X \hat{C} P_X + \lambda I)^{-1} = U(U^*\hat{C}U + \lambda I)^{-1} U^*. \tag{26}$$

Proof of Lemma C.1: Using $U^*U = I$, it holds $(U^*\hat{C}U + \lambda I)U^* = U^*(UU^*\hat{C}UU^* + \lambda UU^*)$ and thus $U^*(UU^*\hat{C}UU^* + \lambda UU^*)^{-1} = (U^*\hat{C}U + \lambda I)^{-1}U^*$. As a consequence,

$$\begin{aligned} g_{\text{KRR}}(\hat{C}) &= P_X(P_X\hat{C}P_X + \lambda I)^{-1} \\ &= UU^*(UU^*\hat{C}UU^* + \lambda I)^{-1} \\ &= U(U^*\hat{C}U + \lambda I)^{-1}U^*. \end{aligned}$$

□

Then we can provide the computatable formulas for Nyström KRR

Proposition C.2 (Nyström KRR): *The Nyström KRR estimator, obtained by projection of eq.* (23) *is*

$$\begin{aligned} \hat{A}^{KRR}_{m,\lambda} &= P_Y\hat{C}_{YX}P_X(P_X\hat{C}P_X + \lambda I)^{-1} \\ &= \widetilde{\Phi}_Y K^{\dagger}_{\tilde{Y},\tilde{Y}}K_{\tilde{Y},Y}K_{X,\tilde{X}}(K_{\tilde{X},X}K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^{\dagger}\widetilde{\Phi}^*_X. \end{aligned}$$

Proof of Proposition C.2: Using the definition in eq. (26), and lemma C.1, we have

$$\begin{aligned} \hat{A}^{\text{KRR}}_{m,\lambda} &= P_Y\hat{C}_{YX}g_{\text{KRR}}(\hat{C}) \\ &= P_Y\hat{C}_{YX}U(U^*\hat{C}U + \lambda I)^{-1}U^* \\ &= P_Y\hat{C}_{YX}U\Sigma V^*V\Sigma^{-1}(U^*\hat{C}U + \lambda I)^{-1}\Sigma^{-1}V^*V\Sigma U^* \end{aligned}$$

Now using the fact that Σ, V, V^* and $U^*\hat{C}U + \lambda I$ are full-rank, it holds [2, eq. (20)]

$$\begin{aligned} \hat{A}^{\text{KRR}}_{m,\lambda} &= P_Y\hat{C}_{YX}\widetilde{\Phi}_X(V^*)^{\dagger}(\Sigma U^*\hat{C}U\Sigma + \lambda\Sigma^2)^{\dagger}V^{\dagger}\widetilde{\Phi}^*_X \\ &= P_Y\hat{C}_{YX}\widetilde{\Phi}_X(V\Sigma U^*\hat{C}U\Sigma V^* + \lambda V\Sigma^2V^*)^{\dagger}\widetilde{\Phi}^*_X. \end{aligned}$$

Finally, by definition of P_Y, $\hat{C}_{YX}$ and $\hat{C}$,

$$\begin{aligned} \hat{A}^{\text{KRR}}_{m,\lambda} &= \widetilde{\Phi}_Y(\widetilde{\Phi}^*_Y\widetilde{\Phi}_Y)^{\dagger}\widetilde{\Phi}^*_Y\hat{\Phi}_{Y|X}\hat{\Phi}^*_X\widetilde{\Phi}_X(\widetilde{\Phi}^*_X\hat{\Phi}_X\hat{\Phi}^*_X\widetilde{\Phi}_X + n\lambda\widetilde{\Phi}^*_X\widetilde{\Phi}_X)^{\dagger}\widetilde{\Phi}^*_X \\ &= \widetilde{\Phi}_Y K^{\dagger}_{\tilde{Y},\tilde{Y}}K_{\tilde{Y},Y}K_{X,\tilde{X}}(K_{X,\tilde{X}}K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^{\dagger}\widetilde{\Phi}^*_X. \end{aligned}$$

□

Remark C.1 (Alternative derivation of the Nyström KRR estimator): *Note that the Nyström KRR estimator can equivalently be derived as the solution to a variational problem similar to eq.* (23)*, where the operator A is restricted to operate between spaces $\mathcal{H}_{\widetilde{X}} := \operatorname{span}\widetilde{\Phi}_X$ and $\mathcal{H}_{\widetilde{Y}} := \operatorname{span}\widetilde{\Phi}_Y$.*

C.2 Nyström PCR

Define the following filter on the spectrum of $P_X\hat{C}P_X$: $g_{\text{PCR}}(\hat{C}) = [\![P_X\hat{C}P_X]\!]^{\dagger}_r$, which truncates it to the first r components before taking the pseudo-inverse. The Nyström PCR estimator, obtained by projection of eq. (24) is

$$\hat{A}^{\text{PCR}}_m = P_Y\hat{C}_{YX}g_{\text{PCR}}(\hat{C}). \tag{27}$$

The next proposition provides an efficiently implementable version of the PCR estimator.

Proposition C.3 (Nyström PCR): *The sketched PCR estimator eq.* (27) *satisfies*

$$\hat{A}^{PCR}_m = \widetilde{\Phi}_Y K^{\dagger}_{\tilde{Y},\tilde{Y}}K_{\tilde{Y},Y}K_{X,\tilde{X}}[\![K^{\dagger}_{\tilde{X},\tilde{X}}K_{\tilde{X},X}K_{X,\tilde{X}}]\!]_r\widetilde{\Phi}^*_X \tag{28}$$

Proof of Proposition C.3: We begin by computing the decomposition of $P_X \hat{C} P_X$ which is necessary to obtain $g_{\text{PCR}}(\hat{C})$. The following expressions are equivalent [39, Proposition 3] for determining its eigenvectors $\tilde{h}$ and eigenvalues λ:

$$\begin{aligned} UU^* \hat{C} UU^* \tilde{h} &= \lambda \tilde{h} \\ U^* \hat{C} U h &= \lambda h, \qquad \tilde{h} = Uh. \end{aligned}$$

Let the truncated eigenvalues be $\Lambda_r = \operatorname{diag}[\lambda_1, \dots, \lambda_r]$ and the eigenvectors be $H_r = [h_1, \dots, h_r]$. Then $\tilde{H}_r = UH_r$ must be normalized such that $\tilde{H}_r^* \tilde{H}_r = H_r^* U^* U H_r = I$. The rank-r truncation $[\![P_X \hat{C} P_X]\!]_r$ is a projection onto $\tilde{H}_r \tilde{H}_r^*$:

$$[\![P_X \hat{C} P_X]\!]_r^\dagger = (UU^* \hat{C} UU^* (UH_r)(UH_r)^*)^\dagger = (UH\Lambda H^* H_r H_r^* U^*)^\dagger = UH_r \Lambda_r^{-1} H_r^* U^*$$

where we used that $U^* \hat{C} U = H \Lambda H^*$.
Now substitute $U = \widetilde{\Phi}_X V \Sigma^{-1}$ to simplify the eigendecomposition of $U^* \hat{C} U$:

$$\begin{aligned} \Sigma^{-1} V^* K_{\tilde{X},X} K_{X,\tilde{X}} V \Sigma^{-1} h &= \lambda h \\ V \Sigma^{-2} V^* K_{\tilde{X},X} K_{X,\tilde{X}} d &= \lambda d, \qquad h = \Sigma^{-1} V^* K_{\tilde{X},X} K_{X,\tilde{X}} d. \end{aligned} \tag{29}$$

where $V \Sigma^{-2} V^* = K_{\tilde{X},\tilde{X}}^\dagger$. Denote by $D_r = [d_1, \dots, d_r]$ the truncated eigenvectors such that $H_r = \Sigma^{-1} V^* K_{\tilde{X},X} K_{X,\tilde{X}} D_r$, normalized such that $H^* H = D^* K_{\tilde{X},X} K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} K_{X,\tilde{X}} D = I$,

$$\begin{aligned} UH_r \Lambda_r^{-1} H_r^* U^* &= \widetilde{\Phi}_X K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} K_{X,\tilde{X}} D_r \Lambda_r^{-1} D_r^* K_{\tilde{X},X} K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}}^\dagger \widetilde{\Phi}_X^* \\ &= \widetilde{\Phi}_X D_r \Lambda_r D_r^* \widetilde{\Phi}_X^* \\ &= \widetilde{\Phi}_X [\![K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} K_{X,\tilde{X}}]\!]_r \widetilde{\Phi}_X^*. \end{aligned}$$

Finally, we can plug the pieces together to get

$$P_Y \hat{C}_{YX} [\![P_X \hat{C} P_X]\!]_r^\dagger = \widetilde{\Phi}_Y K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} [\![K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} K_{X,\tilde{X}}]\!]_r \widetilde{\Phi}_X^*.$$

□

Remark C.2 (Variational problem for Nyström PCR): *Note that, unlike the NysKRR estimator, the variational problem for NysPCR where the operator is restricted to $A : \mathcal{H}_{\widetilde{X}} \to \mathcal{H}_{\widetilde{Y}}$ is not equivalent to the one obtained in proposition C.3 by projecting the covariance operator. In fact, the former does not take the full covariance into account when computing the low-rank projection, but just the Nyström points.*

C.3 Nyström RRR

The Nyström RRR estimator does not correspond to a specific spectral filter. We can nonetheless compute it starting from the expression of the exact empirical estimator [26], projecting the covariance operators, and rearranging the expression to result in a finite-dimensional procedure.

Proposition C.4 (Nyström RRR): *The sketched RRR estimator can be written as*

$$\hat{A}_{m,\lambda}^{RRR} = [\![P_Y \hat{C}_{YX} P_X (P_X \hat{C} P_X + \lambda I)^{-1/2}]\!]_r (P_X \hat{C} P_X + \lambda I)^{-1/2}. \tag{30}$$

To compute it, solve the $m \times m$ eigenvalue problem

$$(K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} K_{Y,\tilde{Y}} K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} w_i = \sigma_i^2 w_i$$

for the first r eigenvectors $W_r = [w_1, \dots, w_r]$, normalized such that $W_r^* K_{\tilde{X},X} K_{Y,\tilde{Y}} K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} W_r = I$. Then let $D_r := K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} W_r$ and $E_r := (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} K_{Y,\tilde{Y}} U_r$, such that the following holds

$$\hat{A}_{m,\lambda}^{RRR} = \widetilde{\Phi}_Y D_r E_r^* \widetilde{\Phi}_X^*. \tag{31}$$

Proof of Proposition C.4: Let $B := n^{1/2} P_Y \hat{C}_{YX} P_X (P_X \hat{C} P_X + \lambda I)^{-1/2}$. The computationally intensive part for this estimator is in evaluating the rank-r truncation $[\![B]\!]_r$. Its singular values and left singular vectors can be obtained by solving the symmetric eigenvalue problem $BB^* q_i = \sigma_i^2 q_i$. We rewrite BB^*

$$\begin{aligned}
BB^* &= P_Y \hat{\Phi}_{Y|X} \hat{\Phi}_X^* P_X (P_X \hat{\Phi}_X \hat{\Phi}_X^* P_X + n\lambda I)^{-1} P_X \hat{\Phi}_X \hat{\Phi}_{Y|X}^* P_Y \\
&= P_Y \hat{\Phi}_{Y|X} \hat{\Phi}_X^* P_X \hat{\Phi}_X (\hat{\Phi}_X^* P_X \hat{\Phi}_X + n\lambda I)^{-1} \hat{\Phi}_{Y|X}^* P_Y \\
&= P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} (K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}}^\dagger K_{\tilde{X},X} + n\lambda I)^{-1} \hat{\Phi}_{Y|X}^* P_Y \\
&= P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}}^\dagger (K_{\tilde{X},X} K_{X,\tilde{X}} K_{\tilde{X},\tilde{X}} + n\lambda I)^{-1} K_{\tilde{X},X} \hat{\Phi}_{Y|X}^* P_Y \\
&= P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} \hat{\Phi}_{Y|X}^* P_Y
\end{aligned}$$

where the second and fourth equalities are applications of the push-through identity, the third by definition of projections and kernel matrices, and the last by collecting $K_{\tilde{X},\tilde{X}}$. By construction, the non-trivial eigenfunctions of BB^* are in the range of $P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}}$, therefore we can set $q_i = P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} w_i$ for some $w_i \in \mathbb{R}^m$, and solve the following eigenvalue problem instead

$$\begin{aligned}
P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} \hat{\Phi}_{Y|X}^* P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} w_i &= \sigma_i^2 P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} w_i \\
(K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^\dagger K_{\tilde{X},X} K_{Y,\tilde{Y}} K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} w_i &= \sigma_i^2 w_i
\end{aligned}$$

where we have simplified the left term of both sides of the equation.
The eigenfunctions of BB^* are therefore $q_i = P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} w_i$, which must be normalized as

$$\|q_i\|^2 = w_i^\top K_{\tilde{X},X} K_{Y,\tilde{Y}} K_{\tilde{Y},\tilde{Y}}^\dagger K_{\tilde{Y},Y} K_{X,\tilde{X}} w_i = 1.$$

Thanks to this normalization, the projector onto the r leading left singular vectors of B is $Q_r Q_r^*$, where $Q_r = [q_1, \dots, q_r]$. Then the NysRRR estimator can be written as

$$Q_r Q_r^* B (P_X \hat{C} P_X + \lambda I)^{-1/2}$$

where

$$\begin{aligned}
B(P_X \hat{C} P_X + \lambda I)^{-1/2} &= P_Y \hat{C}_{YX} P_X (P_X \hat{C} P_X + \lambda I)^{-1} \\
&= P_Y \hat{\Phi}_{Y|X} K_{X,\tilde{X}} (K_{\tilde{X},X} K_{X,\tilde{X}} + n\lambda K_{\tilde{X},\tilde{X}})^{-1} \widetilde{\Phi}_X^*.
\end{aligned}$$

with the same techniques we used for rewriting BB^*. Finally, let D_r and E_r as in the statement. We can apply the projection to obtain

$$Q_r Q_r^* B (P_X \hat{\Phi}_X \hat{\Phi}_X^* P_X + n\lambda I)^{-1/2} = \widetilde{\Phi}_Y D_r E_r^* \widetilde{\Phi}_X^*.$$

□

D Forecasting & Koopman Modes

The three estimators considered in Appendix C are all of the form

$$\hat{A}_\lambda = \widetilde{\Phi}_Y W \widetilde{\Phi}_X^*, \qquad W \in \mathbb{R}^{m \times m}.$$

We will use this generic form to provide expressions for the following operations:

1. producing forecasts of the dynamical system at a future time,
2. computing the approximate eigenvalues and eigenfunctions of the Koopman operator,
3. computing the Koopman modes.

D.1 Forecasting

Given a new data-point $x \in \mathcal{X}$ and an observable function $g \in \mathcal{H}$ (note that this can simply be the identity function), we can approximate the one-step-ahead expectation $\mathbf{E}\big[g(X_{t+1}) | X_t = \boldsymbol{x}\big] =$

$(\mathcal{K}_\pi g)(\boldsymbol{x})$ by using the obtained estimators $\hat{A}^*$. Note that by the reproducing property $\widetilde{\Phi}_Y^* g = [g(\boldsymbol{y}_i), \dots, g(\boldsymbol{y}_m)]^\top =: g_m$, then

$$(\hat{A}^* g)(x) = (\widetilde{\Phi}_X W^\top \widetilde{\Phi}_Y^* g)(\boldsymbol{x}) = (\widetilde{\Phi}_X W^\top g_m)(\boldsymbol{x}) = \sum_{i=1}^m (W^\top g_m)_i k(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}).$$

D.2 Eigenfunctions and eigenvalues

We wish to compute the eigenfunctions $\xi, \psi \in \mathcal{H}$, as well as the eigenvalues λ_i of $\hat{A}$. The left eigenfunctions satisfy $\hat{A}^* \xi_i = \bar{\lambda}_i \xi_i$ and the right eigenfunctions satisfy $\hat{A}\psi_i = \lambda_i \psi_i$. In the following we will use Mollenhauer et al. [39, Proposition 3] to manipulate the eigendecomposition of operators in $\mathcal{H}$.

Consider the decomposition $W = U_r V_r^*$ with $U_r, V_r \in \mathbb{C}^{m \times r}$, which is available for all considered estimators with $r \leq m$. For example, in the Nyström RRR estimator of proposition C.4, we can simply take $U_r = D_r$ and $V_r = E_r$. For the Nyström KRR estimator instead, $r = m$ and we can take the whole of W as our U_r and $V_r = I$.

To compute the **right eigenfunctions** ψ_i, such that $(\widetilde{\Phi}_Y U_r V_r^* \widetilde{\Phi}_X^*)\psi_i = \lambda_i \psi_i$, consider the following equivalent eigendecomposition

$$V_r^* \widetilde{\Phi}_X^* \widetilde{\Phi}_Y U_r \tilde{g}_i = \lambda_i \tilde{g}_i, \quad \text{where } \psi_i = \widetilde{\Phi}_Y U_r \tilde{g}_i.$$

Note that $\widetilde{\Phi}_X^* \widetilde{\Phi}_Y = K_{\tilde{X},\tilde{Y}}$ is a finite-dimensional object which can easily be computed. The eigenfunctions ψ_i must be normalized such that $\psi_i^* \psi_i = 1$ for every i, so we must have

$$\tilde{g}_i^* U_r^* \widetilde{\Phi}_Y^* \widetilde{\Phi}_Y U_r \tilde{g}_i = 1.$$

A very similar process can be followed to obtain the **left eigenfunctions** ξ_i, such that $\widetilde{\Phi}_X V_r U_r^* \widetilde{\Phi}_Y^* \xi_i = \bar{\lambda}_i \xi_i$. Here we consider instead

$$U_r^* \widetilde{\Phi}_Y^* \widetilde{\Phi}_X V_r \tilde{h}_i = \bar{\lambda}_i \tilde{h}_i, \quad \text{where } \xi_i = \widetilde{\Phi}_X V_r \tilde{h}_i.$$

where once again, $\widetilde{\Phi}_Y^* \widetilde{\Phi}_X = K_{\tilde{X},\tilde{Y}}^\top$ and the eigenfunctions must be normalized such that $\tilde{h}_i^* V_r^* \widetilde{\Phi}_X^* \widetilde{\Phi}_X V_r \tilde{h}_i = 1$ for every i. Finally, ψ and ξ must be orthogonal to each other: we must have for $i, j \in [r]$ that $\langle \psi_i, \bar{\xi}_j \rangle_{\mathcal{H}} = \delta_{ij}$ (where δ_{ij} is a Dirac delta equals to 1 when $i = j$ and 0 otherwise). We can compute

$$\langle \psi_i, \bar{\xi}_j \rangle_{\mathcal{H}} = \tilde{h}_i^* V_r^* K_{\tilde{X},\tilde{Y}} U_r \tilde{g}_i = \lambda_j \tilde{h}_i^* \tilde{g}_j,$$

and note that $\tilde{h}_i^* \tilde{g}_j = \delta_{ij}$, but we must normalize ξ such that

$$\xi_i = \widetilde{\Phi}_X V_r \tilde{h}_i / \bar{\lambda}_i.$$

D.3 Koopman modes

Given the eigendecomposition of any estimator $\hat{A}$ as $\hat{A}_r = \sum_{i=1}^r \lambda_i \psi_i \otimes \bar{\xi}_i$, for an observable g we have the following

$$\hat{A}_r^* g = \sum_{i=1}^r \lambda_i \xi_i \langle g, \bar{\psi}_i \rangle_{\mathcal{H}}$$

where $\langle g, \bar{\psi}_i \rangle_{\mathcal{H}} = \gamma_i^g$ are the Koopman modes. Expanding the definition of ψ_i we get

$$\gamma_i^g = \langle g, \bar{\psi}_i \rangle_{\mathcal{H}} = \tilde{g}_i^* U_r^* \widetilde{\Phi}_Y^* g = \tilde{g}_i^* U_r^* g_m \in \mathbb{C}^m$$

which we can efficiently compute.

E Excess risk of the Nyström KRR estimator

E.1 Almost-sure decomposition of the KRR excess risk

Lemma E.1 (Excess risk decomposition in operator norm for KRR): *Let Assumptions 4.1 to 4.3 and 4.5 hold. Then the Nyström KRR estimator* (8) *satisfies almost surely*

$$\begin{aligned}\mathcal{E}(\hat{A}_{m,\lambda}^{KRR})^{1/2} \leq a\lambda^{1/2} + a\theta_1^2\|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \theta_1^2\|(C_{YX} - \hat{C}_{YX})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\ + a\theta_1\theta_2\theta_3\|P_X^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} + \theta_1^2\|P_Y^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\end{aligned}$$

where $\theta_1 := \|\hat{C}_\lambda^{-1/2}C_\lambda^{1/2}\|$, $\theta_2 := \|\hat{C}_\lambda^{1/2}C_\lambda^{-1/2}\|$, $\theta_3 := \|\hat{C}_\lambda^{-1}C_\lambda\|$, *and* a *is the constant of Assumption 4.5.*

Proof of Lemma E.1: Let $\theta_1 := \|\hat{C}_\lambda^{-1/2}C_\lambda^{1/2}\|$, $\theta_3 := \|\hat{C}_\lambda^{-1}C_\lambda\|$. As in Lemma C.1 define $g_{\text{KRR}}(\hat{C}) := U(U^*\hat{C}U + \lambda I)^{-1}U^*$. We have

$$\begin{aligned}\mathcal{E}(\hat{A}_{m,\lambda}^{\text{KRR}})^{1/2} &= \|\Phi_{Y|X} - \hat{A}_{m,\lambda}^{\text{KRR}}\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} \\ &\leq \|\Phi_{Y|X} - A_\lambda\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} + \|(A_\lambda - C_{YX}g_{\text{KRR}}(\hat{C}))\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} \\ &\quad + \|(C_{YX}g_{\text{KRR}}(\hat{C}) - \hat{A}_{m,\lambda}^{\text{KRR}})\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} \\ &\leq \underbrace{\|\Phi_{Y|X} - A_\lambda\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})}}_{A} + \underbrace{\|(A_\lambda - C_{YX}g_{\text{KRR}}(\hat{C}))C^{1/2}\|}_{B} \\ &\quad + \underbrace{\|(C_{YX}g_{\text{KRR}}(\hat{C}) - \hat{A}_{m,\lambda}^{\text{KRR}})C^{1/2}\|}_{C} \qquad (32)\end{aligned}$$

where we used the polar decomposition $\Phi_X^* = WC^{1/2}$ for some partial isometry $W : \mathcal{H} \to L^2_\pi$.
The first term is

$$\begin{aligned}\|\Phi_{Y|X} - A_\lambda\Phi_X\| &= \|\Phi_X\mathcal{K}_\pi^* - C_{YX}C_\lambda^{-1}\Phi_X\| \\ &\leq a\lambda^{1/2} + \|(I - P_\mathcal{H})\Phi_{Y|X}\|\end{aligned}$$

where we used the definition of $\Phi_{Y|X}$ and applied Lemma H.1.

The second term of our decomposition (32) can be bounded as follows:
It holds

$$\begin{aligned}B &= \|C_{YX}(C_\lambda^{-1} - g_{\text{KRR}}(\hat{C}))C^{1/2}\| \\ \text{(by Lemma H.2:)} \quad &\leq a\|C(C_\lambda^{-1} - g_{\text{KRR}}(\hat{C}))C^{1/2}\| \\ &\leq a\left(\underbrace{\|C(C_\lambda^{-1} - \hat{C}_\lambda^{-1})C^{1/2}\|}_{B_1} + \underbrace{\|C(\hat{C}_\lambda^{-1} - g_{\text{KRR}}(\hat{C}))C^{1/2}\|}_{B_2}\right)\end{aligned}$$

We now bound the terms B_1 and B_2 separately.

$$\begin{aligned}B_1 &= \|C(C_\lambda^{-1} - \hat{C}_\lambda^{-1})C^{1/2}\| \\ &= \|CC_\lambda^{-1}(\hat{C}_\lambda - C_\lambda)\hat{C}_\lambda^{-1}C^{1/2}\| \\ &\leq \|CC_\lambda^{-1}\|\|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1/2}\|\|C_\lambda^{1/2}\hat{C}_\lambda^{-1/2}\|\|\hat{C}_\lambda^{-1/2}C^{1/2}\| \\ &\leq \theta_1^2\|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1/2}\|\end{aligned}$$

Let $\hat{P}_\lambda := \hat{C}_\lambda^{1/2}g_{\text{KRR}}(\hat{C})\hat{C}_\lambda^{1/2}$. We recall that $g_{\text{KRR}}(\hat{C}) = P_X g_{\text{KRR}}(\hat{C})$, so that

$$\begin{aligned}\hat{P}_\lambda^2 &= \hat{C}_\lambda^{1/2}(g_{\text{KRR}}(\hat{C})\hat{C}_\lambda P_X)g_{\text{KRR}}(\hat{C})\hat{C}_\lambda^{1/2} \\ &= \hat{C}_\lambda^{1/2}P_X g_{\text{KRR}}(\hat{C})\hat{C}_\lambda^{1/2} \\ &= \hat{P}_\lambda.\end{aligned}$$

This implies $\hat{P}_\lambda^2 = \hat{P}_\lambda = \hat{P}_\lambda^*$. Hence $\hat{P}_\lambda$ is an orthogonal projection, and defining $\hat{P}_\lambda^\perp = I - \hat{P}_\lambda$ it holds $\|\hat{P}_\lambda^\perp\|_{\mathcal{B}(\mathcal{H})} \leq 1$. We can thus bound B_2 as follows:

$$\begin{aligned}
B_2 &= \|C(\hat{C}_\lambda^{-1} - g_{\text{KRR}}(\hat{C}))C^{1/2}\| \\
&= \|C\hat{C}_\lambda^{-1/2}(I - \hat{P}_\lambda)\hat{C}_\lambda^{-1/2}C^{1/2}\| \\
\text{(by Lemma I.1)} \qquad &= \|C\hat{C}_\lambda^{-1}P_X^\perp\hat{C}_\lambda^{1/2}(I - \hat{P}_\lambda)\hat{C}_\lambda^{-1/2}C^{1/2}\| \\
&= \|C\hat{C}_\lambda^{-1}\|_{\mathcal{B}(\mathcal{H})}\|P_X^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\|C_\lambda^{-1/2}\hat{C}_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\|I - \hat{P}_\lambda\|\|\hat{C}_\lambda^{-1/2}C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\
&\leq \theta_1\theta_2\theta_3\|P_X^\perp C_\lambda^{1/2}\|
\end{aligned}$$

For the third term, due to Lemma C.1:

$$\begin{aligned}
C &= \|(C_{YX} - P_Y\hat{C}_{YX})g_{\text{KRR}}(\hat{C})C^{1/2}\| && (33)\\
&\leq \|(C_{YX} - P_Y\hat{C}_{YX})C_\lambda^{-1/2}\|\|C_\lambda^{1/2}\hat{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})}\|\hat{P}_\lambda\|_{\mathcal{B}(\mathcal{H})}\|\hat{C}_\lambda^{-1/2}C^{1/2}\|_{\mathcal{B}(\mathcal{H})} && (34)\\
&\leq \theta_1^2\|(C_{YX} - P_Y\hat{C}_{YX})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} && (35)\\
&\leq \theta_1^2\Big(\|P_Y^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} + \|(C_{YX} - \hat{C}_{YX})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})}\Big) && (36)
\end{aligned}$$

where we used Lemma J.7 for the last inequality.
Starting again from (32) and putting everything together, we get

$$\begin{aligned}
\mathcal{E}(\hat{A}_{m,\lambda}^{\text{KRR}})^{1/2} \leq\ & a\lambda^{1/2} \\
&+ a\theta_1^2\|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\
&+ \theta_1^2\|(C_{YX} - \hat{C}_{YX})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\
&+ a\theta_1\theta_2\theta_3\|P_X^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\
&+ \theta_1^2\|P_Y^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}.
\end{aligned}$$

□

E.2 Excess risk rates for KRR

In order to control the terms appearing in our decomposition, we recall that Assumption 4.4 implies

$$d_{\text{eff}}(\lambda) \leq C_\beta\lambda^{-\beta} \text{ where } C_\beta := \begin{cases} \dfrac{c}{1-\beta} & , \beta < 1 \\ K^2 & , \beta = 1 \end{cases}, \tag{37}$$

where c is the constant of Assumption 4.4, see [8, Proposition 3 with $b \to 1/\beta$ and $\beta \to c$] and [16, Lemma 11] which shows that the existence of a constant C_β such that the first part of (37) holds implies in return $\lambda_i(C) \lesssim i^{-1/\beta}$.

Proof of Theorem 4.6: By Lemma E.1 taking $P_X = P_Y$, it holds almost surely

$$\begin{aligned}
\mathcal{E}(\hat{A}_{m,\lambda}^{\text{KRR}})^{1/2} \leq\ & a\lambda^{1/2} + a\theta_1^2\|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \theta_1^2\|(C_{YX} - \hat{C}_{YX})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\
&+ a\theta_1\theta_2\theta_3\|P_X^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} + \theta_1^2\|P_Y^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}
\end{aligned}$$

and we recall that $\theta_1 := \|\hat{C}_\lambda^{-1/2}C_\lambda^{1/2}\|$ and $\theta_3 := \|\hat{C}_\lambda^{-1}C_\lambda\|$. We bound separately the terms appearing in this expression.

Bound of θ_1 and θ_2. We control these term by bounding $\|C_\lambda^{-1/2}(\hat{C} - C)C_\lambda^{-1/2}\|$. By Lemma J.3 it holds for any $\delta' \in]0, 1[$ and any $\lambda \in]0, \|C\|_{\mathcal{B}(\mathcal{H})}]$ with probability $1 - \delta'$

$$\Big\|C_\lambda^{-1/2}(\hat{C} - C)C_\lambda^{-1/2}\Big\| \leq \frac{4c_\tau\beta}{3n\lambda^\tau} + \sqrt{\frac{2c_\tau\beta}{n\lambda^\tau}} \quad \text{where} \quad \beta = \log\Big(\tfrac{8K^2}{\delta'\lambda}\Big) \tag{38}$$

A sufficient condition to bound the right hand side of the previous expression by $1/4$ is to have $n\lambda^\tau > 32c_\tau\beta$ (in which cases both terms are bounded by $1/8$). Assuming this holds, $I - \|C_\lambda^{-1/2}(\hat{C} -$

$C)C_\lambda^{-1/2}\|$ is invertible and we also have

$$\begin{aligned}
\theta_2^2 = \|\hat{C}_\lambda^{1/2} C_\lambda^{-1/2}\|^2 &= \|C_\lambda^{-1/2}\hat{C}_\lambda C_\lambda^{-1/2}\| = \|I - C_\lambda^{-1/2}(C-\hat{C})C_\lambda^{-1/2}\| \\
&\leq 1 + \|C_\lambda^{-1/2}(C-\hat{C})C_\lambda^{-1/2}\| \\
&\leq 1.25 \\
\text{and thus} \quad \theta_2 &\leq 1.12 \\
\text{while} \quad \theta_1^2 = \|\hat{C}_\lambda^{-1/2} C_\lambda^{1/2}\|^2 &= \|(C_\lambda^{-1/2}\hat{C}_\lambda C_\lambda^{-1/2})^{-1}\| \\
&\overset{(i)}{\leq} (1 - \|C_\lambda^{-1/2}(\hat{C}-C)C_\lambda^{-1/2}\|)^{-1} \\
&\leq 1.34 \\
\text{and thus} \quad \theta_1 &\leq 1.16
\end{aligned}$$

where (i) can be obtained by taking the Neumann expansion of $I - \|C_\lambda^{-1/2}(\hat{C}-C)C_\lambda^{-1/2}\|$.

Bound for θ_3. By Lemma J.5 it holds with probability $1-\delta'$

$$\|(C-\hat{C})C_\lambda^{-1}\|_{\text{op}} \leq \frac{2K\sqrt{c_\tau}\log(2/\delta')}{\lambda^{(\tau+1)/2}n} + \sqrt{\frac{2K^2\operatorname{tr}(C_\lambda^{-2}C)\log(2/\delta')}{n}} \tag{39}$$

Both terms in the above rhs are bounded by $1/4$ provided

$$\begin{aligned}
\lambda^{(\tau+1)/2} n &\geq 8K\sqrt{c_\tau}\log(2/\delta') \\
n &\geq 32K^2\lambda^{-(1+\beta)}\log(2/\delta')
\end{aligned}$$

where we used $\operatorname{tr}(C_\lambda^{-2}C) = \sum \lambda_i(C)(\lambda_i(C)+\lambda)^{-2} \leq \lambda^{-1}\operatorname{tr}(C_\lambda^{-1}C) \leq C_\beta\lambda^{-(1+\beta)}$. When this is the case, we have $\|(C-\hat{C})C_\lambda^{-1}\|_{\text{op}} \leq 1/2 < 1$ and the operator $I - (\hat{C}_\lambda - C_\lambda)C_\lambda^{-1}$ is invertible.

$$\begin{aligned}
\theta_3 = \|(\hat{C}_\lambda C_\lambda^{-1})^{-1}\| &= \|(I - (\hat{C}_\lambda - C_\lambda)C_\lambda^{-1})^{-1}\| \\
&\overset{(i)}{\leq} (1 - \|(\hat{C}_\lambda - C_\lambda)C_\lambda^{-1}\|_{\mathcal{B}(\mathcal{H})})^{-1} \\
&\leq 2.
\end{aligned}$$

where (i) can be obtained by considering the Neumann expansion of $I - (\hat{C}_\lambda - C_\lambda)C_\lambda^{-1}$.

Bound for $\|P_X^\perp C_\lambda^{1/2}\|$. By Lemma J.6, provided $\lambda \in]0, \|C\|_{\mathcal{B}(\mathcal{H})}]$ it holds with probability $1-\delta'$

$$\|P_X^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} \leq \sqrt{3\lambda}$$

provided $m \geq \max(67, 5\operatorname{ess\,sup}_{x\sim\pi}\|C_\lambda^{-1/2}\phi(x)\|^2)\log\frac{4K^2}{\lambda\delta'}$, which by Lemma H.3 is ensured if $m \geq \max(67, 5\frac{c_\tau}{\lambda^\tau})\log\frac{4K^2}{\lambda\delta'}$.

Bound for $\|(C-\hat{C})C_\lambda^{-1/2}\|$ and $\|(C_{YX}-\hat{C}_{YX})C_\lambda^{-1/2}\|$. By Lemma J.4, for any $\delta' \in]0,1[$, each of the following events holds with probability $1-2\delta'$:

$$\max(\|(C-\hat{C})C_\lambda^{-1/2}\|, \|(C_{YX}-\hat{C}_{YX})C_\lambda^{-1/2}\|) \leq \frac{2K\sqrt{c_\tau}\log(2/\delta')}{\lambda^{\tau/2}n} + \sqrt{\frac{2K^2 d_{\text{eff}}(\lambda)\log(2/\delta')}{n}} \tag{40}$$

By Eq. (37) we have $d_{\text{eff}}(\lambda) \leq C_\beta\lambda^{-\beta}$.

Choosing $\delta' = \delta/5$, we get via a union bound with probability $1-\delta$ that $\theta_1\theta_2\theta_3 \leq 2.6$, $\theta_1^2 \leq 1.34$ and

$$\begin{aligned}
\mathcal{E}(\hat{A}_{m,\lambda}^{\text{KRR}})^{1/2} &\leq a\lambda^{1/2} + 1.34(a+1)\left(\frac{2K\sqrt{c_\tau}\log(2/\delta')}{\lambda^{\tau/2}n} + \sqrt{\frac{2K^2C_\beta\log(2/\delta')}{n\lambda^\beta}}\right) + (2.6a+1.34)\sqrt{3}\lambda^{1/2} \\
&\leq c_1\lambda^{1/2} + c_2\lambda^{-\tau/2}n^{-1} + c_3\lambda^{-\beta/2}n^{-1/2} \\
\text{where:} \quad c_1 &:= (5.5a + 2.33) \\
c_2 &:= 1.34(a+1)2K\sqrt{c_\tau}\log(2/\delta') \\
c_3 &:= 1.34(a+1)\sqrt{2K^2C_\beta\log(2/\delta')}
\end{aligned}$$

for any λ and m satisfying the constraints

$$\begin{cases} \lambda > n^{-1/\tau}(32c_\tau)^{1/\tau}\log\Big(\frac{8K^2}{\delta'\lambda}\Big)^{1/\tau} \\ \lambda \geq n^{-2/(\tau+1)}(8K\sqrt{c_\tau}\log(2/\delta'))^{2/(\tau+1)} \\ \lambda \geq n^{-1/(1+\beta)}(32K^2\log(2/\delta'))^{1/(1+\beta)} \\ \lambda \in]0, K^2]. \\ m \geq \max(67, 5\frac{c_\tau}{\lambda^\tau})\log\frac{4K^2}{\lambda\delta'} \quad \text{(uniform sampling)} \end{cases} \tag{41}$$

We pick $\boxed{\lambda := c_\lambda n^{-1/(1+\beta)}}$ which is asymptotically the saturating constraint (given that $1/(1+\beta) < 1 < 2/(\tau+1) \leq 1/\tau$), where c_λ is a constant choosen to enforce the following equations (which are sufficient conditions for eq. (41) to hold):

$$\begin{cases} c_\lambda^\tau n^{1-\tau/(1+\beta)} > (32c_\tau)\log\Big(\frac{8K^2 n^{1/(1+\beta)}}{\delta' c_\lambda}\Big) \\ c_\lambda^{(\tau+1)/2} n^{1-(\tau+1)/(2(1+\beta))} \geq 8K\sqrt{c_\tau}\log(2/\delta') \\ c_\lambda \geq (32K^2\log(2/\delta'))^{1/(1+\beta)} \\ c_\lambda n^{-1/(1+\beta)} \leq K^2 \end{cases} \tag{42}$$

As $1-(\tau+1)/(2(1+\beta)) > 0$, a sufficient condition for the second equation is

$$c_\lambda \geq (8K\sqrt{c_\tau}\log(2/\delta'))^{2/(\tau+1)}.$$

Assuming $c_\tau \geq 8K^2$, a sufficient condition to satisfy the first constraint is

$$c_\lambda^\tau n^{1-\tau/(1+\beta)} > (32c_\tau)2\max(\log\Big(n^{1/(1+\beta)}\Big), \log\Big((\delta')^{-1}\Big))$$

which is in particular ensured (noting that $\log(n)/n^\nu \leq 1/(\nu e)$ for any $n, \nu > 0$) whenever

$$c_\lambda > (64c_\tau \max((e(1+\beta-\tau))^{-1}, \log\big(1/\delta'\big)))^{1/\tau}$$

Noting that $1+\beta-\tau \leq 1$, we get that

$$c_\lambda > (64c_\tau(e(1+\beta-\tau))^{-1}\log\big(1/\delta'\big))^{1/\tau}$$

is also sufficient. We recall that $1/(1+\beta) < 1 < 2/(\tau+1) \leq 1/\tau$, so that we can choose

$$\boxed{c_\lambda := \log(2/\delta')^{1/\tau}\max((32K^2)^{1/(1+\beta)}, (8K\sqrt{c_\tau})^{2/(\tau+1)}, (64c_\tau(e(1+\beta-\tau))^{-1})^{1/\tau}, 8K^2)}$$

while the last constraint $n \geq (c_\lambda/K^2)^{1+\beta}$ is satisfied by assumption.

$$\begin{aligned} \mathcal{E}(\hat{A}^{\text{KRR}}_{m,\lambda})^{1/2} &\leq c_1\lambda^{1/2} + c_2\lambda^{-\tau/2}n^{-1} + c_3\lambda^{-\beta/2}n^{-1/2} \\ &\leq c_1 c_\lambda^{1/2} n^{-1/(2(1+\beta))} + c_2 c_\lambda^{-\tau/2} n^{\tau/(2(1+\beta))-1} + c_3 c_\lambda^{-\beta/2} n^{\beta/(2(1+\beta))-1/2} \\ &\leq c_1 c_\lambda^{1/2} n^{-1/(2(1+\beta))} + c_2 c_\lambda^{-\tau/2} n^{-(1+2\beta+(1-\tau))/(2(1+\beta))} + c_3 c_\lambda^{-\beta/2} n^{-1/(2(1+\beta))} \\ &\leq (c_1 c_\lambda^{1/2} + c_2 c_\lambda^{-\tau/2} + c_3 c_\lambda^{-\beta/2}) n^{-1/(2(1+\beta))}. \end{aligned}$$

which gives the claimed result. The last constraint (on m) is satisfied by the assumptions of the lemma. □

F Excess risk of the Nyström RRR estimator

Recalling (30), NyströmRRR estimator is of the form $\hat{A}^{\text{RRR}}_{m,\lambda} = [\![\tilde{B}]\!]_r(\tilde{C}_\lambda)^{-1/2}$, where $\tilde{B} := \tilde{C}_{YX}(\tilde{C}_\lambda)^{-1/2}$ for $\tilde{C}_{YX} := P_Y\hat{C}_{YX}P_X$ and $\tilde{C}_\lambda := P_X\hat{C}P_X + \lambda I$. While the population version is $A^{\text{RRR}}_\lambda := [\![B]\!]_r C_\lambda^{-1/2}$ where $B := C_{YX}(C_\lambda)^{-1/2}$.

In this section we follow the approach in [27] and decompose the operator norm excess risk in the following way:

$$\mathcal{E}(\hat{A}^{\text{RRR}}_{m,\lambda})^{1/2} = \|\Phi_{Y|X} - A_\lambda\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} + \|(A_\lambda - A^{\text{RRR}}_\lambda)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} + \|(A^{\text{RRR}}_\lambda - \hat{A}^{\text{RRR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})}$$

Then, recalling that $A_\lambda = C_{YX}C_\lambda^{-1}$ and $\hat{A}^{\text{KRR}}_{m,\lambda} = \tilde{B}\tilde{C}_\lambda^{-1/2}$, we also have $A^{\text{RRR}}_\lambda = P_B A_\lambda$ and $\hat{A}^{\text{RRR}}_{m,\lambda} = P_{\tilde{B}}\hat{A}^{\text{KRR}}_{m,\lambda}$, where P_B and $P_{\tilde{B}}$ are orthogonal projectors onto leading r left singular vectors of B and $\tilde{B}$, respectively.

Thus,

$$\begin{aligned}
\mathcal{E}(\hat{A}^{\text{RRR}}_{m,\lambda})^{1/2} &\le a\,\lambda^{1/2} + \sigma_{r+1}(\Phi_{Y|X}) + \|(A^{\text{RRR}}_\lambda - \hat{A}^{\text{RRR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(\mathcal{H})} \\
&= a\,\lambda^{1/2} + \sigma_{r+1}(\Phi_{Y|X}) + \|(P_B A_\lambda - P_{\tilde{B}}\hat{A}^{\text{KRR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(\mathcal{H})} \\
&\le a\,\lambda^{1/2} + \sigma_{r+1}(\Phi_{Y|X}) + \|((P_B - P_{\tilde{B}})A_\lambda\Phi_X\|_{\mathcal{B}(\mathcal{H})} + \|P_{\tilde{B}}(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(\mathcal{H})} \\
&\le a\,\lambda^{1/2} + \sigma_{r+1}(\Phi_{Y|X}) + K\,\frac{\|\tilde{B}\tilde{B}^* - BB^*\|_{\mathcal{B}(\mathcal{H})}}{\sigma_r^2(B) - \sigma_{r+1}^2(B)} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(\mathcal{H})}
\end{aligned}$$

where the last inequality is due to $\|A_\lambda\| \le a$ and [27, Proposition 4].

Recalling Lemma E.1, we observe that

$$\mathcal{E}(\hat{A}^{\text{RRR}}_{m,\lambda})^{1/2} \le \sigma_{r+1}(\Phi_{Y|X}) + K\,\frac{\|\tilde{B}\tilde{B}^* - BB^*\|}{\sigma_r^2(B) - \sigma_{r+1}^2(B)} + \underbrace{a\,\lambda^{1/2} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})\Phi_X\|}_{\le\text{r.h.s. of the bound in Lemma E.1}}$$

Therefore, to prove Lemma 4.7 for the RRR estimator we just need to bound $\|\tilde{B}\tilde{B}^* - BB^*\|_{\mathcal{B}(\mathcal{H})}$. To that end, observe that, after some algebra, one obtains

$$\tilde{B}\tilde{B}^* - BB^* = A_\lambda(\tilde{C}_{YX} - C_{YX})^* + (\tilde{C}_{YX} - C_{YX})A_\lambda^* - A_\lambda(\tilde{C}_\lambda - C_\lambda)A_\lambda^* + (\tilde{A}_\lambda - A_\lambda)\tilde{C}_\lambda(\tilde{A}_\lambda - A_\lambda)^*,$$

and, consequently,

$$\begin{aligned}
\|\tilde{B}\tilde{B}^* - BB^*\|_{\mathcal{B}(\mathcal{H})} \le 2a\|\tilde{C}_{YX} - C_{YX}\|_{\mathcal{B}(\mathcal{H})} + a^2\|P_X\hat{C}P_X - C\|_{\mathcal{B}(\mathcal{H})} \\
+ \|C_\lambda^{-1/2}\tilde{C}_\lambda C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})}\|(\tilde{A}_\lambda - A_\lambda)C_\lambda^{1/2}\|^2_{\mathcal{B}(\mathcal{H})},
\end{aligned}$$

follows using that $\|A_\lambda\|_{\mathcal{B}(\mathcal{H})} \le a$.

On the other hand,

$$\|\tilde{C}_{YX} - C_{YX}\|_{\mathcal{B}(\mathcal{H})} \le \|P_Y(\hat{C}_{YX} - C_{YX})P_X\|_{\mathcal{B}(\mathcal{H})} + \|P_Y^\perp C_{YX}P_X\|_{\mathcal{B}(\mathcal{H})} + \|C_{YX}P_X^\perp\|_{\mathcal{B}(\mathcal{H})},$$

which implies that

$$\|\tilde{C}_{YX} - C_{YX}\|_{\mathcal{B}(\mathcal{H})} \le \|\hat{C}_{YX} - C_{YX}\|_{\mathcal{B}(\mathcal{H})} + 2\,a\,K\,\varepsilon_1,$$

where $\varepsilon_1 := \max\{\|P_X^\perp C^{1/2}\|_{\mathcal{B}(\mathcal{H})},\ \|P_Y^\perp C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\}$. Similarly, we obtain

$$\|\tilde{C}_\lambda - C_\lambda\|_{\mathcal{B}(\mathcal{H})} \le \|\hat{C} - C\|_{\mathcal{B}(\mathcal{H})} + 2\,K\,\varepsilon_1. \tag{43}$$

But, ε_1 can be bounded by Lemma J.6. Indeed, provided $\lambda \in]0, \|C\|_{\mathcal{B}(\mathcal{H})}]$, it holds with probability $1-\delta'$

$$\varepsilon_1 \le \sqrt{3\lambda}$$

provided $m \ge \max(67, 5\,\text{ess}\sup_{x\sim\pi}\|C_\lambda^{-1/2}\phi(x)\|^2)\log\frac{4K^2}{\lambda\delta'}$, which by Lemma H.3 is ensured if $m \ge \max(67, 5\frac{c_\tau}{\lambda^\tau})\log\frac{4K^2}{\lambda\delta'}$.

Additionally,

$$\begin{aligned}\|C_\lambda^{-1/2}\tilde{C}_\lambda C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} &\le \|C_\lambda^{-1/2}P_X\hat{C}_\lambda P_X C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \lambda\|C_\lambda^{-1/2}P_X^\perp C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})}\\ &\le \theta_2^2\,\|\hat{C}_\lambda^{-1/2}P_X\hat{C}_\lambda P_X C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + 1\\ &\le \theta_2^2\,\|\hat{C}_\lambda^{1/2}P_X\hat{C}_\lambda^{-1}P_X C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} + 1\\ &\le \theta_2^2\,\|\hat{C}_\lambda^{1/2}P_X(P_X\hat{C}_\lambda P_X)^\dagger P_X C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} + 1,\end{aligned}$$

implies that

$$\|C_\lambda^{-1/2}\tilde{C}_\lambda C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \le \theta_2^2 + 1 \le 2.25, \tag{44}$$

provided, as above, that $n\lambda^\tau > 32c_\tau\beta$.

Therefore, setting $\varepsilon_0 := \max\{a\|\hat{C}_{YX} - C_{YX}\|_{\mathcal{B}(\mathcal{H})}, a^2\|\hat{C} - C\|_{\mathcal{B}(\mathcal{H})}\}$, for all $i \in [m]$ we have

$$\left|\sigma_i^2(\tilde{B}) - \sigma_i^2(B)\right| \le \|\tilde{B}\tilde{B}^* - BB^*\| \le 3\varepsilon_0 + 6.93\,K\,a^2\lambda^{1/2} + 2.25\,\varepsilon_2^2, \tag{45}$$

where $\varepsilon_2 := \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})C_\lambda^{1/2}\|$ is the variance of Nyström KRR estimator, and conclude that

$$\mathcal{E}(\hat{A}^{\text{RRR}}_{m,\lambda})^{1/2} \le \sigma_{r+1}(\Phi_{Y|X}) + K\,\frac{3\varepsilon_0 + 6.93\,K\,a^2\,\lambda^{1/2} + 2.25\,\varepsilon_2^2}{\sigma_r^2(B) - \sigma_{r+1}^2(B)} + a\,\lambda^{1/2} + \varepsilon_2.$$

Therefore, the proof of Lemma 4.7 for RRR estimator directly follows from the bound on $a\,\lambda^{1/2} + \varepsilon_2$ given in the proof of Theorem 4.6, and the fact that, see e.g. [26], $\varepsilon_0 \lesssim n^{-1/2} \lesssim \lambda^{1/2}$.

G Excess risk of the Nyström PCR estimator

Recalling Eq. (27), NyströmPCR estimator is of the form

$$\hat{A}^{\text{PCR}}_m = P_Y\hat{C}_{YX}[\![P_X\hat{C}P_X]\!]_r^\dagger = \tilde{C}_{YX}[\![\tilde{C}_\lambda]\!]_r = \hat{A}^{\text{KRR}}_{m,\lambda}P_{\tilde{C}_\lambda},$$

for $\lambda = 0$ and with $P_{\tilde{C}_\lambda}$ being the orthogonal projector onto leading r eigenspace of $\tilde{C}_\lambda$. So, to prove Lemma 4.7 for PCR estimator, denote $\hat{A}^{\text{PCR}}_{m,\lambda} := \hat{A}^{\text{KRR}}_{m,\lambda}P_{\tilde{C}_\lambda}$ for $\lambda \ge 0$, and let us define the population version $A^{\text{PCR}}_\lambda = A_\lambda P_{C_\lambda}$, where P_{C_λ} being the orthogonal projector onto leading r eigenspace of C_λ. As in the previous section we start with decomposition

$$\begin{aligned}\mathcal{E}(\hat{A}^{\text{PCR}}_m)^{1/2} =& \|\Phi_{Y|X} - A_\lambda\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} + \|(A_\lambda - A^{\text{PCR}}_\lambda)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} +\\ &\|(A^{\text{PCR}}_\lambda - \hat{A}^{\text{PCR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} + \|(\hat{A}^{\text{PCR}}_{m,\lambda} - \hat{A}^{\text{PCR}}_m)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})}.\end{aligned}$$

The first and the second term are easily bounded by $\|\Phi_{Y|X} - A_\lambda\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} \le a\,\lambda^{1/2}$, and

$$\begin{aligned}\|(A_\lambda - A^{\text{PCR}}_\lambda)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} &= \|A_\lambda(I - P_{C_\lambda})\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})}\\ &\le a\,\|(I - P_{C_\lambda})C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \le a\,\sigma_{r+1}(\Phi_X).\end{aligned}$$

For the third term, start by observing that

$$\begin{aligned}\|(A^{\text{PCR}}_\lambda - \hat{A}^{\text{PCR}}_{m,\lambda})\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} &= \|(A_\lambda P_{C_\lambda} - \hat{A}^{\text{KRR}}_{m,\lambda}P_{\tilde{C}_\lambda})C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\\ &\le \|A_\lambda(P_{C_\lambda} - P_{\tilde{C}_\lambda})C^{1/2}\|_{\mathcal{B}(\mathcal{H})} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda}C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\\ &\le a\,K\,\|P_{C_\lambda} - P_{\tilde{C}_\lambda}\|_{\mathcal{B}(\mathcal{H})} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda}C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\\ &\le a\,K\,\frac{\|\tilde{C}_\lambda - C_\lambda\|_{\mathcal{B}(\mathcal{H})}}{\sigma_r^2(\Phi_X) - \sigma_{r+1}^2(\Phi_X)} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda}C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\\ &\le K\,\frac{\varepsilon_0 + 2\,a\,K\,\varepsilon_1}{\sigma_r^2(\Phi_X) - \sigma_{r+1}^2(\Phi_X)} + \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda}C^{1/2}\|_{\mathcal{B}(\mathcal{H})}\end{aligned}$$

where the second last inequality is due to [27, Proposition 4] and the last one uses Eq. (43). Moreover, we have that

$$\begin{aligned}\|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda} C^{1/2}\|_{\mathcal{B}(\mathcal{H})} &\leq \|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\, \|C_\lambda^{-1/2}\tilde{C}_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\, \|\tilde{C}_\lambda^{-1/2} P_{\tilde{C}_\lambda} C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &\leq \varepsilon_2 \sqrt{1+\theta_2^2}\, \|P_{\tilde{C}_\lambda}\tilde{C}_\lambda^{-1/2} C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &\leq \varepsilon_2 \sqrt{1+\theta_2^2}\, \|C_\lambda^{1/2}\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})},\end{aligned}$$

where we have used Eq. (44) and the fact that $P_{\tilde{C}_\lambda}$ is the spectral projector of $\tilde{C}_\lambda^{-1/2}$. Therefore, due to

$$\begin{aligned}\|C_\lambda^{1/2}\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} = \|C_\lambda^{1/2}[P_X^\perp + P_X]\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} &\leq \|C_\lambda^{1/2}P_X\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \|C_\lambda^{1/2}P_X^\perp\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &\leq \theta_2\|\hat{C}_\lambda^{1/2}P_X\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \|C_\lambda^{1/2}P_X^\perp\|_{\mathcal{B}(\mathcal{H})}\|\tilde{C}_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &\leq \theta_2\|\hat{C}_\lambda^{1/2}P_X(P_X\hat{C}_\lambda P_X)^\dagger P_X\hat{C}_\lambda^{1/2}\|^{1/2}_{\mathcal{B}(\mathcal{H})} + \|C_\lambda^{1/2}P_X^\perp\|_{\mathcal{B}(\mathcal{H})}\lambda^{-1/2} \\ &\leq \theta_2 + \varepsilon_1\lambda^{-1/2}\end{aligned}$$

we obtain

$$\|(A_\lambda - \hat{A}^{\text{KRR}}_{m,\lambda})P_{\tilde{C}_\lambda} C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \leq \varepsilon_2\left(1.68 + 1.5\lambda^{-1/2}\,\varepsilon_1\right),$$

provided that $n\lambda^\tau > 32c_\tau\beta$.

Finally for the last term, observe that $[\![\tilde{C}_\lambda]\!]_r^\dagger$ and $[\![\tilde{C}_0]\!]_r^\dagger$ share the same eigenvectors, and hence $[\![\tilde{C}_0]\!]_r^\dagger - [\![\tilde{C}_\lambda]\!]_r^\dagger = \lambda[\![\tilde{C}_\lambda\tilde{C}_0]\!]_r^\dagger$. Hence, it holds that

$$\begin{aligned}\|(\hat{A}^{\text{PCR}}_{m,\lambda} - \hat{A}^{\text{PCR}}_m)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} &= \|\tilde{C}_{YX}([\![\tilde{C}_\lambda]\!]_r^\dagger - [\![\tilde{C}_0]\!]_r^\dagger)C^{1/2}\|_{\mathcal{B}(\mathcal{H})} = \lambda\|\tilde{C}_{YX}\tilde{C}_\lambda^{-1}[\![\tilde{C}_0]\!]_r^\dagger C^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &\leq \lambda\|\tilde{C}_{YX}\tilde{C}_\lambda^{-1}[\![\tilde{C}_0]\!]_r^\dagger\tilde{C}_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}\|\tilde{C}_\lambda^{-1/2}C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &= \lambda\|\tilde{C}_{YX}\tilde{C}_\lambda^{-1/2}[\![\tilde{C}_0]\!]_r^\dagger\|_{\mathcal{B}(\mathcal{H})}\|\tilde{C}_\lambda^{-1/2}C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})} \\ &= \lambda\,\|[\![\tilde{C}_0]\!]_r^\dagger\|_{\mathcal{B}(\mathcal{H})}\,\|\tilde{B}\|_{\mathcal{B}(\mathcal{H})}\,\|\tilde{C}_\lambda^{-1/2}C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}.\end{aligned}$$

Now, recalling Eq. (45), we can bound

$$\begin{aligned}\|\tilde{B}\|^2_{\mathcal{B}(\mathcal{H})} &\leq \|C_\lambda^{-1/2}C_{YX}^* C_{YX} C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} + \|BB^* - \tilde{B}\tilde{B}^*\|_{\mathcal{B}(\mathcal{H})} \\ &\leq a^2K^2 + 3\varepsilon_0 + 6.93\,K\,a^2\lambda^{1/2} + 2.25\,\varepsilon_2^2,\end{aligned}$$

and,

$$\lambda^{1/2}\|[\![\tilde{C}_0]\!]_r^\dagger\|_{\mathcal{B}(\mathcal{H})} = \frac{\lambda^{1/2}}{\lambda_r(P_X\hat{C}P_X)} \leq \frac{\lambda^{1/2}}{\lambda_r(C) - \|C_\lambda - \tilde{C}_\lambda\|_{\mathcal{B}(\mathcal{H})}} \leq \frac{\lambda^{1/2}}{\sigma_r^2(\Phi_X) - \|\hat{C} - C\|_{\mathcal{B}(\mathcal{H})} - 2\,K\,\varepsilon_1}.$$

Thus, consequently, we obtain

$$\begin{aligned}\|(\hat{A}^{\text{PCR}}_{m,\lambda} - \hat{A}^{\text{PCR}}_m)\Phi_X\|_{\mathcal{B}(L^2_\pi,\mathcal{H})} \leq &\lambda^{1/2}\left(\theta_2 + \varepsilon_1\lambda^{-1/2}\right)\left(a^2K^2 + 3\varepsilon_0 + 6.93\,K\,a^2\lambda^{1/2} + 2.25\,\varepsilon_2^2\right)\cdot \\ &\frac{\lambda^{1/2}}{\sigma_r^2(\Phi_X) - \|\hat{C} - C\|_{\mathcal{B}(\mathcal{H})} - 2\,K\,\varepsilon_1}.\end{aligned}$$

To conclude, observe that $r > n^{\frac{1}{\beta(1+\beta)}}$ due to Assumption 4.4 implies that $\sigma_{r+1}(\Phi_X) \lesssim n^{-\frac{1}{2(1+\beta)}}$.

Therefore, collecting all the terms, under the assumptions of Lemma 4.7 we obtain

$$\mathcal{E}(\hat{A}^{\text{PCR}}_m)^{1/2} \lesssim c_{\text{PCR}}\, n^{-\frac{1}{2(1+\beta)}},$$

where $c_{\text{PCR}} = (\sigma_r^2(\Phi_X) - \sigma_{r+1}^2(\Phi_X))^{-1}$ is the problem dependant constant.

H Auxiliary results

Lemma H.1 ([27, Proposition 2] with $\alpha = 1$): *Under Assumption 4.2 it holds*

$$\|\Phi_X \mathcal{K}_\pi^* - C_{YX} C_\lambda^{-1} \Phi_X\| \leq a\lambda^{1/2}.$$

Lemma H.2: *Let A be a bounded operator. Under Assumptions 4.1 and 4.5, it holds*

$$\|C_{YX} A\| \leq a\|CA\| \tag{46}$$

Proof of Lemma H.2: Note that under Assumption 4.5, as $C_{XY}C_{YX} \preccurlyeq a^2 C^2$ it also holds $A^* C_{XY} C_{YX} A \preccurlyeq a^2 A C^2 A$ and thus:

$$\begin{aligned}\|C_{YX} A\| &= \|A^* C_{YX} C_{YX} A\|^{1/2} \\ &\leq a\|A^* C^2 A\|^{1/2} \\ &= a\|CA\|.\end{aligned}$$

□

The next lemma is a consequence of Assumption 4.3 and will be used in our concentration inequalities.

Lemma H.3: *Under Assumption 4.3, it holds π-almost surely for any ν:*

$$\left\| C_\lambda^{-(1-\nu)/2} \phi(x) \right\|^2 \leq c_\tau \lambda^{-[\tau-\nu]_+} K^{2[\nu-\tau]_+}.$$

The two following corollaries can be obtained picking $\nu = 0$ and $\nu = -1$:

$$\left\| C_\lambda^{-1/2} \phi(x) \right\|^2 \leq \frac{c_\tau}{\lambda^\tau} \quad \textit{and} \quad \left\| C_\lambda^{-1} \phi(x) \right\|^2 \leq \frac{c_\tau}{\lambda^{\tau+1}}.$$

Proof of Lemma H.3: By [16, Theorem 9], it holds $c_\tau := \|k_\pi^\tau\|_\infty^2 = \operatorname{ess\,sup}_{x\sim\pi} \sum_{i\in I} \mu_i^\tau |e_i(x)|^2$ (where (e_i) is defined in Appendix A.3, and we recall that $(\sqrt{\mu_i} e_i)_{i\in\mathbb{N}}$ is an orthonormal basis of $\mathcal{H}$. Denoting $\mu_i := \lambda_i(C)$, it holds

$$\begin{aligned}\left\| C_\lambda^{-(1-\nu)/2} \phi(x) \right\|^2 &= \left\| \left(\sum_{i\in I} (\mu_i + \lambda)^{-(1-\nu)/2} (\sqrt{\mu_i} e_i) \otimes (\sqrt{\mu_i} e_i) \right) \phi(x) \right\|^2 \\ &= \left(\sum_{i\in I} \mu_i e_i(x)^2 (\mu_i + \lambda)^{-1+\nu} \right) \\ &= \sum_{i\in I} \mu_i^{1-\tau} (\mu_i + \lambda)^{-1+\nu} \mu_i^\tau e_i(x)^2 \\ &= \sum_{i\in I} \left(\frac{\mu_i}{\mu_i + \lambda} \right)^{1-\tau} (\mu_i + \lambda)^{\nu-\tau} \mu_i^\tau e_i(x)^2 \\ &\leq \sum_{i\in I} (\mu_i + \lambda)^{-(\tau-\nu)} \mu_i^\tau e_i(x)^2 \\ &\leq c_\tau \lambda^{-[\tau-\nu]_+} K^{2[\nu-\tau]_+}.\end{aligned}$$

where we used $\sup |\mu_i| \leq K^2$. □

I Deterministic sketching results

Lemma I.1: *Denoting* $R := I - \hat{C}_\lambda^{1/2} g_{\text{KRR}}(\hat{C})\hat{C}_\lambda^{1/2}$*, it holds*

$$R\hat{C}_\lambda^{1/2} = R\hat{C}_\lambda^{1/2} P_X^\perp.$$

Proof of Lemma I.1: This is a direct consequence of the fact that $g_{\text{KRR}}(\hat{C})\hat{C}_\lambda P_X = P_X$:

$$R\hat{C}_\lambda^{1/2} P_X = \hat{C}_\lambda^{1/2} P_X - \hat{C}_\lambda^{1/2} g_{\text{KRR}}(\hat{C})\hat{C}_\lambda P_X = 0.$$

□

J Concentration results

J.1 Generic concentration lemmas

All our concentration results derive from two versions of the Bernstein inequality. We first state an inequality for sums of random variables in a Hilbert space based on [66, Theorem 3.3.4], which itself derives from a result of [44].

Lemma J.1: *Let* $(A_i)_{1\leq i\leq n}$ *be i.i.d. copies of a random variable* A *in a separable Hilbert space* $(H, \|\cdot\|)$*. Assume* $\mathbf{E}A = \mu$ *and* $\exists \sigma > 0, \exists L > 0, \forall p \geq 2, \mathbf{E}\|A-\mu\|^p \leq \frac{1}{2}p!\sigma^2 L^{p-2}$*. Then for any* $\delta \in]0,1[$ *it holds:*

$$P\left[\left\|\frac{1}{n}\sum_{i=1}^n A_i - \mu\right\| \leq \frac{2L\log(2/\delta)}{n} + \sqrt{\frac{2\sigma^2\log(2/\delta)}{n}}\right] \geq 1-\delta \tag{47}$$

The assumption on the moments holds in particular when $\operatorname{ess\,sup}\|A\| \leq L/2$ *and* $\mathbf{E}[\|A\|^2] \leq \sigma^2$*.*

Proof of Lemma J.1: See proof of [9, Lemma E.3] for a precise derivation based on [66, Theorem 3.3.4]. □

We now state a version of the Bernstein concentration inequality for self-ajoint operators in operator norm, which is a restatement of [31, Lemma 24]. In the following, we denote $\mathrm{r}_{\text{eff}}(A) := \operatorname{tr}(A)/\|A\|$ the effective rank of a nonnegative definite operator A.

Lemma J.2 (Bernstein for self-ajoint operators acting on a Hilbert): *Let* H *be a separable Hilbert space and* A_i *be i.i.d. copies of a random variable* A *taking values in the space of self-adjoint Hilbert-Schmidt operators on* H*. Assume* $\mathbf{E}A = 0$*,* $\operatorname{ess\,sup}\|A\|_{\text{op}} \leq c$ *for some* $c > 0$ *(where* $\|\cdot\|_{\text{op}}$ *denotes the operator norm) and that there exists a positive semi-definite trace class operator* V *such that* $\mathbf{E}[A^2] \preccurlyeq V$*. Then for any* $\delta \in]0,1[$ *and* $n \geq 1$ *it holds*

$$P\left[\left\|\frac{1}{n}\sum_{i=1}^n A_i\right\|_{\text{op}} \geq \frac{2c\beta}{3n} + \sqrt{\frac{2\|V\|\beta}{n}}\right] \leq \delta \quad \textit{where} \quad \beta = \log\Big(\tfrac{4\,\mathrm{r}_{\textit{eff}}(V)}{\delta}\Big) \tag{48}$$

Proof of Lemma J.2: See [31, Appendix B.7, Lemma 24]. □

J.2 Applied concentration lemmas

Lemma J.3: *Let Assumption 4.1 hold. Let $\delta \in]0,1[$. Then for i.i.d. samples $(x_i, y_i)_{1\leq i\leq n}$ and any $\lambda \in]0, \|C\|_{\mathcal{B}(\mathcal{H})}]$ it holds*

$$P\left[\left\|C_\lambda^{-1/2}(\hat{C}-C)C_\lambda^{-1/2}\right\|_{\mathcal{B}(\mathcal{H})} \geq \frac{4c_\tau\beta}{3n\lambda^\tau} + \sqrt{\frac{2c_\tau\beta}{n\lambda^\tau}}\right] \leq \delta \quad \textit{where} \quad \beta = \log\left(\tfrac{8K^2}{\delta\lambda}\right) \tag{49}$$

Proof of Lemma J.3: We apply Lemma J.2 on the random variables $A_i = \xi(X_i)\otimes\xi(X_i) - C_\lambda^{-1/2}CC_\lambda^{-1/2}$ where $\xi(X_i) := C_\lambda^{-1/2}\phi(X_i)$. It holds

$$\begin{aligned}
\operatorname{ess\,sup}\|A_i\|_{\mathcal{B}(\mathcal{H})} &\leq 2\operatorname{ess\,sup}\|\xi(X_i)\|^2 \\
&\leq \frac{2c_\tau}{\lambda^\tau}. \quad \text{(by Lemma H.3)} \\
\mathbf{E}[A_i^2] &= \mathbf{E}[\|\xi(X_i)\|^2\xi(X_i)\xi(X_i)^*] - (C_\lambda^{-1/2}CC_\lambda^{-1/2})^2 \\
&\preccurlyeq \mathbf{E}[\|\xi(X_i)\|^2\xi(X_i)\xi(X_i)^*] \\
&\preccurlyeq \frac{c_\tau}{\lambda^\tau}\mathbf{E}[\xi(X_i)\xi(X_i)^*] \\
&= \frac{c_\tau}{\lambda^\tau}CC_\lambda^{-1}
\end{aligned}$$

Thus applying Lemma J.2 with $c = \frac{2c_\tau}{\lambda^\tau}$ and $V = \frac{c_\tau}{\lambda^\tau}CC_\lambda^{-1}$, we get

$$P\left[\left\|\frac{1}{n}\sum_{i=1}^n A_i\right\| \geq \frac{4c_\tau\beta}{3n\lambda^\tau} + \sqrt{\frac{2c_\tau\beta}{\lambda^\tau n}}\right] \leq \delta \quad \text{where} \quad \beta = \log\left(\tfrac{8K^2}{\delta\lambda}\right) \tag{50}$$

where we used the fact that $\|CC_\lambda^{-1}\|_{\mathcal{B}(\mathcal{H})} \leq 1$ and controlled the effective rank using $\operatorname{tr}(CC_\lambda^{-1}) \leq K^2/\lambda$ and $\|CC_\lambda^{-1}\|_{\mathcal{B}(\mathcal{H})} = \|C\|_{\mathcal{B}(\mathcal{H})}/(\|C\|_{\mathcal{B}(\mathcal{H})}+1) \geq 1/2$ because $\lambda \leq \|C\|_{\mathcal{B}(\mathcal{H})}$ by assumption. □

Lemma J.4: *Let Assumptions 4.1 and 4.3 hold. Let $\delta \in]0,1[$. Then for i.i.d. samples $(x_i, y_i)_{1\leq i\leq n}$ we get*

$$P\Big[\|(C-\hat{C})C_\lambda^{-1/2}\|_{\text{op}} \leq \epsilon(\lambda,\delta)\Big] \geq 1-\delta \tag{51}$$

$$\textit{and} \quad P\Big[\|(C_{YX}-\hat{C}_{YX})C_\lambda^{-1/2}\|_{\text{op}} \leq \epsilon(\lambda,\delta)\Big] \geq 1-\delta \tag{52}$$

$$\textit{where} \quad \epsilon(\lambda,\delta) := \frac{2K\sqrt{c_\tau}\log(2/\delta)}{\lambda^{\tau/2}n} + \sqrt{\frac{2K^2 d_{\text{eff}}(\lambda)\log(2/\delta)}{n}} \tag{53}$$

Proof of Lemma J.4: We first write the proof for the eq. (51). For this result, we use the fact that $\|(C-\hat{C})C_\lambda^{-1/2}\|_{\mathcal{B}(\mathcal{H})} \leq \|(C-\hat{C})C_\lambda^{-1/2}\|_{\text{HS}}$ and bound the Hilbert-Schmidt norm. As $(\text{HS}(\mathcal{H}), \|\cdot\|_{\text{HS}(\mathcal{H})})$ is a Hilbert space, we apply Lemma J.1 on the random variables $A_i = \phi(x_i)\otimes\xi(x_i)$ where $\xi(x) = C_\lambda^{-1/2}\phi(x)$.

$$\begin{aligned}
\operatorname{ess\,sup}\|A\|_{\text{HS}} &= \operatorname{ess\,sup}\|\phi(x)\|\|\xi(x)\| \\
&\leq \frac{K\sqrt{c_\tau}}{\lambda^{\tau/2}} \quad \text{(by assumption 4.1 and lemma H.3)} \\
\mathbf{E}[\|A\|_{\text{HS}}^2] &= \mathbf{E}[\|\phi(x)\|^2\|\xi(x)\|^2] \\
&\leq K^2 d_{\text{eff}}(\lambda)
\end{aligned}$$

Thus applying Lemma J.1 with $L = \frac{K\sqrt{c_\tau}}{\lambda^{\tau/2}}$ and $\sigma^2 = K^2 d_{\text{eff}}(\lambda)$ gives

$$P\left[\left\|\frac{1}{n}\sum_{i=1}^{n} A_i - \mu\right\|_{\text{HS}} \leq \frac{2K\sqrt{c_\tau}\log(2/\delta)}{\lambda^{\tau/2}n} + \sqrt{\frac{2K^2 d_{\text{eff}}(\lambda)\log(2/\delta)}{n}}\right] \geq 1-\delta.$$

This yields the desired result via the inequality between operator and Hilbert-Schmidt norms.
For the bound eq. (52) on the cross-covariance, we take $A_i = \phi(y_i) \otimes \xi(x_i)$ but the rest of the proof is inchanged. □

Lemma J.5: *Let Assumptions 4.1 and 4.3 hold. Let $\delta \in]0,1[$. Then for i.i.d. samples $(x_i, y_i)_{1\leq i\leq n}$ we get*

$$P\left[\|(C-\hat{C})C_\lambda^{-1}\|_{\text{op}} \leq \frac{2K\sqrt{c_\tau}\log(2/\delta)}{\lambda^{(\tau+1)/2}n} + \sqrt{\frac{2K^2\operatorname{tr}(C_\lambda^{-2}C)\log(2/\delta)}{n}}\right] \geq 1-\delta. \tag{54}$$

Proof of Lemma J.5: For this result, we use the fact that $\|(C-\hat{C})C_\lambda^{-1}\|_{\mathcal{B}(\mathcal{H})} \leq \|(C-\hat{C})C_\lambda^{-1}\|_{\text{HS}}$ and bound the Hilbert-Schmidt norm. As $(\text{HS}(\mathcal{H}), \|\cdot\|_{\text{HS}(\mathcal{H})})$ is a Hilbert space, we apply Lemma J.1 on the random variables $A_i = \phi(x_i) \otimes \omega(x_i)$ where $\omega(x) = C_\lambda^{-1}\phi(x)$.

$$\begin{aligned}
\operatorname{ess\,sup}\|A\|_{\text{HS}} &= \operatorname{ess\,sup}\|\phi(x)\|\|\omega(x)\| \\
&\leq \frac{K\sqrt{c_\tau}}{\lambda^{(\tau+1)/2}} \quad \text{(by assumption 4.1 and lemma H.3)} \\
\mathbf{E}[\|A\|_{\text{HS}}^2] &= \mathbf{E}[\|\phi(x)\|^2\|\omega(x)\|^2] \\
&\leq K^2\mathbf{E}[\operatorname{tr}(C_\lambda^{-2}\phi(x)\phi(x)^*)] \\
&= K^2\operatorname{tr}(C_\lambda^{-2}C)
\end{aligned}$$

Thus applying Lemma J.1 with $L = \frac{K\sqrt{c_\tau}}{\lambda^{(\tau+1)/2}}$ and $\sigma^2 = K^2\operatorname{tr}(C_\lambda^{-2}C)$ gives

$$P\left[\left\|\frac{1}{n}\sum_{i=1}^{n} A_i - \mu\right\|_{\text{HS}} \leq \frac{2K\sqrt{c_\tau}\log(2/\delta)}{\lambda^{(\tau+1)/2}n} + \sqrt{\frac{2K^2\operatorname{tr}(C_\lambda^{-2}C)\log(2/\delta)}{n}}\right] \geq 1-\delta.$$

This yields the desired result via the inequality between operator and Hilbert-Schmidt norms. □

J.3 Probabilistic inequalities

Lemma J.6 (Uniform Nyström approximation): *Let Assumption 4.1 hold. Let $P : \mathcal{H} \to \mathcal{H}$ denote the orthogonal projection on $\operatorname{span}\left\{\, \phi(\tilde{\boldsymbol{x}}_j) \;\middle|\; 1 \leq j \leq m \,\right\}$, where the landmarks $(\tilde{\boldsymbol{x}}_j)_{1\leq j\leq m}$ are drawn i.i.d. from the empirical data. Then for any $\lambda \in]0, \|C_\lambda\|_{\mathcal{B}(\mathcal{H})}]$ we have*

$$\|P^\perp C_\lambda^{1/2}\|_{\mathcal{B}(\mathcal{H})}^2 \leq 3\lambda$$

with probability at least $1-\delta$ provided

$$m \geq \max(67, 5\operatorname{ess\,sup}\|C_\lambda^{-1/2}\phi(x)\|^2)\log\frac{4K^2}{\lambda\delta}.$$

J.4 Concentration lemmas for the sketched operators

Lemma J.7: *It holds almost surely*

$$\|(C_{YX} - P_Y \hat{C}_{YX}) C_\lambda^{-1/2}\| \leq \|P_Y^\perp C_\lambda^{1/2}\| + \|(C_{YX} - \hat{C}_{YX}) C_\lambda^{-1/2}\|$$

Proof of Lemma J.7: It holds

$$\begin{aligned} C_{YX} - P_Y \hat{C}_{YX} &= C_{YX} - P_Y C_{YX} + P_Y C_{YX} - P_Y \hat{C}_{YX} \\ &= P_Y^\perp C_{YX} + P_Y (C_{YX} - \hat{C}_{YX}) \end{aligned}$$

Thus

$$\begin{aligned} \|(C_{YX} - P_Y \hat{C}_{YX}) C_\lambda^{-1/2}\| &= \|(P_Y^\perp C_{YX} + P_Y (C_{YX} - \hat{C}_{YX})) C_\lambda^{-1/2}\| \\ &\leq \|P_Y^\perp C_{YX} C_\lambda^{-1/2}\| + \|P_Y (C_{YX} - \hat{C}_{YX}) C_\lambda^{-1/2}\| \\ &\leq \|P_Y^\perp C_\lambda^{1/2}\| \|C_\lambda^{-1/2} C_{YX} C_\lambda^{-1/2}\| + \|(C_{YX} - \hat{C}_{YX}) C_\lambda^{-1/2}\| \end{aligned}$$

Eventually it holds $\|C_\lambda^{-1/2} C_{YX} C_\lambda^{-1/2}\| \leq 1$. Indeed, as π is invariant, it holds that

$$\|\mathcal{K}_\pi\| = \sup_{f \in L^2_\pi : \|f\|_{L^2_\pi} \leq 1} \int_x \left| \int f(y) p(x, \mathrm{d}y) \right|^2 \mathrm{d}\pi(x) \leq 1.$$

and denoting $\Phi_X = C^{1/2} U$ the polar decomposition of Φ_X for some partial isometry $U : L^2_\pi \to \mathcal{H}$, and using $\Phi^*_{Y|X} = \mathcal{K}_\pi \Phi^*_X$, we get

$$\begin{aligned} \|C_\lambda^{-1/2} C_{YX} C_\lambda^{-1/2}\| &= \|C_\lambda^{-1/2} \Phi_{Y|X} \Phi_X^* C_\lambda^{-1/2}\| \\ &\leq \|C_\lambda^{-1/2} C^{1/2}\| \|U \mathcal{K}_\pi^* U^*\| \|C^{1/2} C_\lambda^{-1/2}\| \\ &\leq 1. \end{aligned}$$

□

J.5 Concentration for mixing processes

Lemma J.8 (Kostic et al. [26, Lemma 1]): *Let X be strictly stationary with values in a normed space $(\mathcal{X}, \|\cdot\|)$ and assume $n = 2pk$ with $p, k \in \mathbb{N}$. Let $Z_1, \ldots, Z_p$ be p independent copies of $Z_1 = \sum_{i=1}^k X_i$. Then for $s > 0$:*

$$P\Big[\Big\|\sum_{i=1}^n X_i\Big\| > s\Big] \leq 2P\Big[\Big\|\sum_{j=1}^p Z_j\Big\| > s/2\Big] + 2(p-1)\beta_X(k).$$