Fast Training of Large Kernel Models with Delayed Projections

Amirhesam Abedsoltan
UC San Diego
abedsol1@ucsd.edu

Siyuan Ma[*]
Google
siyuan@siyuanm.com

Parthe Pandit
IIT Bombay
pandit@iitb.ac.in

Mikhail Belkin
UC San Diego
mbelkin@ucsd.edu

Abstract

Classical kernel machines have historically faced significant challenges in scaling to large datasets and model sizes—a key ingredient that has driven the success of neural networks. In this paper, we present a new methodology for building kernel machines that can scale efficiently with both data size and model size. Our algorithm introduces delayed projections to Preconditioned Stochastic Gradient Descent (PSGD) allowing the training of much larger models than was previously feasible. We validate our algorithm, EigenPro 4, across multiple datasets, demonstrating drastic training speedups without compromising the performance. Our implementation is publicly available at: https://github.com/EigenPro/EigenPro.

1 Introduction

Kernel methods have strong theoretical foundations and broad applicability. They have also served as the foundation for understanding many significant phenomena in modern machine learning [9, 4, 3, 28]. Despite these advantages, the scalability of kernel methods has remained a persistent challenge, particularly when applied to large datasets. Addressing this limitation is critical for expanding the utility of kernel-based techniques in modern machine learning applications.

A naive approach for training kernel machines is to directly solve the equivalent kernel matrix inversion problem. In general, the computational complexity of this approach is $O(n^3)$, where n is the number of training samples. Thus, computational cost grows rapidly with the size of the dataset, making it computationally intractable for datasets with more than $\sim 10^5$ data points.

To address this challenge, various methods employing iterative algorithms and approximations have been proposed. Among these, Gradient Descent (GD)-based algorithms like Pegasos [21] and EigenPro 1.0,2.0 [13, 14] have significantly reduced the computational complexity to $O(n^2)$. These methods, adaptable for stochastic settings, offer more efficient implementations. Nevertheless, the scalability of kernel machines remains constrained by the inherent linkage between the model size and the training set.

Furthermore, the Nyström methods have emerged as a favored approach for scaling kernel machines, with seminal works with [26] paving the way. Methods such as NYTRO [5], Falkon [20] and ASkotch [19] leverage the Nyström Approximation (NA) in combination with other strategies to enhance performance. NYTRO merges NA with gradient descent to improve condition number, ASkotch combines it with block coordinate descent, whereas Falkon combines it with the Conjugate Gradient method, facilitating the handling of large training sets. However, these strategies are limited by model size due to memory restrictions, exhibiting quadratic scaling in relation to the size of the

[*]Work completed during author's time at Google. Current affiliation: Lynko.

Algorithm	FLOPS setup	FLOPS per batch	Memory
EigenPro 4	$O(1)$	$O(p)$	$O(p)$
EigenPro 3	$O(1)$	$O(p^2)$	$O(p)$
Falkon	$O(p^3)$	$O(p)$	$O(p^2)$

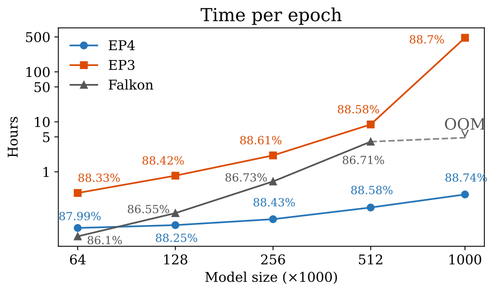

Figure 1: Comparison of per-epoch time complexity across solvers as a function of model size p. Performance in terms of classification test accuracy (indicated as percentages) is annotated next to each data point, showing that EP4 maintains superior or comparable performance across all model sizes. Details of the experiment and hardware can be found in Appendix C.

model. For instance, scaling Falkon [16] method to a model size of $512,000$ necessitates over 1TB of RAM, surpassing the capacity of most high-end servers available today.

Other lines of work in the Gaussian Processes literature, e.g., [22, 27, 7, 15], use so-called *inducing points* to control model complexity. However, these methods face similar scaling issues as they require quadratic memory in terms of the number of inducing points, preventing large models.

Recently, EigenPro 3.0 was introduced in [1]. Unlike previous versions, EigenPro 3.0 distangles the model from the training set, similar to Falkon, but with the added advantage that its memory requirements scales linearly with the model size. This advancement makes it feasible to tackle kernel models of sizes previously deemed unattainable. However, its per iteration time complexity remains quadratic relative to the model size, significantly slowing its practical application.

In this paper, we build upon EigenPro 3.0 and introduce EigenPro 4.0. This new algorithm retains the advantageous features of EigenPro 3.0, such as decoupling the model from the training set and linear scaling in memory complexity. Moreover, it significantly improves upon the time complexity, achieving amortized linear scaling per iteration with respect to model size. Empirically we observe that the proposed algorithm converges in fewer epochs, without compromising generalization performance.

1.1 Main contribution

Our method for kernel machine problems achieves three key advantages: (1) linear amortized time complexity per iteration, (2) linear memory scaling with model size, (3) comparable or superior performance compare to existing methods while demonstrating up to 600× speedup in our experiments. Figure 1 demonstrates these benefits on the CIFAR5M data set.

1.2 Organization of the Paper

The remainder of this paper is organized as follows. Section 2 provides the necessary background and preliminaries. In Section 3, we derive the complete algorithm, and introduce the key insights and techniques that enable its dramatic improvement in computational efficiency. Finally, Section 4 presents extensive experimental results across multiple datasets and model sizes.

2 Notation and Background

In what follows, functions are lowercase letters a, sets are uppercase letters A, vectors are lowercase bold letters $\boldsymbol{a}$, matrices are uppercase bold letters $\boldsymbol{A}$, operators are calligraphic letters $\mathcal{A}$, spaces and sub-spaces are boldface calligraphic letters $\boldsymbol{\mathcal{A}}$.

General kernel models. Following EigenPro 3.0 [1] notations, given training data $(X, \boldsymbol{y}) = \left\{\boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\right\}_{i=1}^n$, *General kernel models* are models of the form,

$$f(\boldsymbol{x}) = \sum_{i=1}^{p} \alpha_i K(\boldsymbol{x}, \mathbf{z}_i).$$

Here, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive semi-definite symmetric kernel function and $Z = \{\mathbf{z}_i \in \mathbb{R}^d\}_{i=1}^p$ are *centers*, which are not necessarily in the training set X. We will refer to p as the *model size*. We define $\mathcal{H}$ as the unique reproducing kernel Hilbert space (RKHS) corresponding to K.

Loss function. Our goal will be to find the solution to the following infinite-dimensional Mean Squared Error (MSE) problem for general kernel models,

$$\underset{f \in \mathcal{H}}{\text{minimize}}\ L(f) = \frac{1}{2}\sum_{i=1}^{n}(f(\boldsymbol{x}_i) - y_i)^2, \qquad \text{subject to} \qquad f \in \boldsymbol{\mathcal{Z}} := \text{span}\Big(\{K(\cdot, \mathbf{z}_j)\}_{j=1}^p\Big). \quad (1)$$

Evaluations and kernel matrices. The vector of evaluations of a function f over a set $X = \{\boldsymbol{x}_i\}_{i=1}^n$ is denoted $f(X) := (f(\boldsymbol{x}_i)) \in \mathbb{R}^n$. For sets X and Z, with $|X| = n$ and $|Z| = p$, we denote the kernel matrix $K(X, Z) \in \mathbb{R}^{n \times p}$, while $K(Z, X) = K(X, Z)^\top$. Similarly, $K(\cdot, X) \in \mathcal{H}^n$ is a vector of n functions, and we use $K(\cdot, X)\boldsymbol{\alpha} := \sum_{i=1}^n K(\cdot, \boldsymbol{x}_i)\alpha_i \in \mathcal{H}$, to denote their linear combination. Finally, for an operator $\mathcal{A}$, a function a, and a set $A = \{\boldsymbol{a}_i\}_{i=1}^k$, we denote the vector of evaluations,

$$\mathcal{A}\left\{a\right\}(A) := (b(\boldsymbol{a}_i)) \in \mathbb{R}^k \qquad \text{where} \quad b = \mathcal{A}\left(a\right). \quad (2)$$

Fréchet derivative. Given a function $J : \mathcal{H} \to \mathbb{R}$, the Fréchet derivative of J with respect to f is a linear functional, denoted $\nabla_f J$, such that for $h \in \mathcal{H}$

$$\lim_{\|h\|_{\mathcal{H}} \to 0} \frac{|J(f+h) - J(f) - \nabla_f J(h)|}{\|h\|_{\mathcal{H}}} = 0. \quad (3)$$

Since $\nabla_f J$ is a linear functional, it lies in the dual space $\mathcal{H}^*$. Since $\mathcal{H}$ is a Hilbert space, it is self-dual, whereby $\mathcal{H}^* = \mathcal{H}$. If f is a general kernel model, and L is the square loss for a given dataset $(X, \boldsymbol{y})$, i.e., $L(f) := \frac{1}{2}\sum_{i=1}^n (f(\boldsymbol{x}_i) - y_i)^2$ we can apply the chain rule, and using reproducing property of $\mathcal{H}$, and the fact that $\nabla_f \langle f, g\rangle_{\mathcal{H}} = g$, we get, that the Fréchet derivative of L, at $f = f_0$ is,

$$\nabla_f L(f_0) = \sum_{i=1}^n (f_0(\boldsymbol{x}_i) - y_i)\nabla_f f(\boldsymbol{x}_i) = K(\cdot, X)(f_0(X) - \boldsymbol{y}). \quad (4)$$

Hessian operator. The Hessian operator $\nabla_f^2 L : \mathcal{H} \to \mathcal{H}$ for the square loss is given by

$$\mathcal{K} := \sum_{i=1}^n K(\cdot, \boldsymbol{x}_i) \otimes K(\cdot, \boldsymbol{x}_i), \qquad \mathcal{K}\left\{f\right\}(\mathbf{z}) = \sum_{i=1}^n K(\mathbf{z}, \boldsymbol{x}_i) f(\boldsymbol{x}_i) = K(\mathbf{z}, X) f(X), \quad (5)$$

where $\otimes$ denotes the *outer product* between functions in the RKHS, defined as $(a \otimes b)(\cdot) = a(\cdot)\langle b, \cdot\rangle_{\mathcal{H}}$. The operator $\mathcal{K}$ has non-negative eigenvalues, which we order as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Hence, we can write its eigen-decomposition as $\mathcal{K} = \sum_{i=1}^n \lambda_i\, \psi_i \otimes \psi_i$. Combining Equations 4 and 5, we can rewrite the Fréchet derivative of the loss function as

$$\nabla_f L(f_0)(z) = \mathcal{K}\left\{f_0(X) - \boldsymbol{y}\right\}(z). \quad (6)$$

Exact minimum norm solution. The closed-form minimum $\|\cdot\|_{\mathcal{H}}$ norm solution to the problem defined in equation (1) is given by:

$$\hat{f} := K(\cdot, Z) K^\dagger(Z, X)\boldsymbol{y}, \quad (7)$$

where † is the pseudoinverse or Moore–Penrose inverse. In the case of $X = Z$ it simplifies to $\hat{f} := K(\cdot, X) K^{-1}(X, X)\boldsymbol{y}$.

Gradient Descent (GD). If we apply GD on the optimization problem in 1, with learning rate η, in the $\mathcal{H}$ functional space, the update is as following,

$$f_{t+1} = f_t - \eta \cdot \nabla_f L(f_t) = f_t - \eta K(\cdot, X)\left(f_t(X) - \boldsymbol{y}\right). \quad (8)$$

The first point to note is that the derivative lies in $\boldsymbol{\mathcal{X}} := \text{span}\left(\{K(\cdot, \boldsymbol{x}_j)\}_{j=1}^n\right)$ rather than in $\boldsymbol{\mathcal{Z}}$. Therefore, when $X \neq Z$, SGD cannot be applied in this form. We will revisit this issue later. The second point is that in the case of $X = Z$, traditional *kernel regression* problem, the convergence of SGD depends on the condition number of $\mathcal{K}$. Simply put, this is the ratio of the largest to the smallest non-zero singular value of the Hessian operator defined in 5. It is known that for general kernel models this condition number is ill conditioned and converges slow, see [2] for more details on this.

EigenPro . Prior work, EigenPro , by [13], addresses the slow convergence of SGD by introducing a preconditioned stochastic gradient descent mechanism in Hilbert spaces. The update rule is the same as Equation 9 but with an additional preconditioner $\mathcal{P}:\mathcal{H}\to\mathcal{H}$ applied to the gradient,

$$f_{t+1} = f_t - \eta\cdot\mathcal{P}\nabla_f L(f_t). \tag{9}$$

In short, the role of the preconditioner $\mathcal{P}$ is to suppress the top eigenvalues of the Hessian operator $\mathcal{K}$ to improve the condition number. We next explicitly define $\mathcal{P}$.

Definition 1 (Top-q Eigensystem)**.** Let $\lambda_1 > \lambda_2 > \ldots > \lambda_n$ be the eigenvalues of a Hermitian matrix $\boldsymbol{A}\in\mathbb{R}^{n\times n}$, where for standard unit vector $\boldsymbol{e}_i$, we have $\boldsymbol{A}\boldsymbol{e}_i = \lambda_i\boldsymbol{e}_i$. We define the tuple $(\Lambda_q, \boldsymbol{E}_q, \lambda_{q+1})$ as the top-q eigensystem, where:

$$\Lambda_q := \operatorname{diag}(\lambda_1,\lambda_2,\ldots,\lambda_q)\in\mathbb{R}^{q\times q}, \qquad \boldsymbol{E}_q := [\boldsymbol{e}_1,\boldsymbol{e}_2,\ldots,\boldsymbol{e}_q]\in\mathbb{R}^{n\times q}. \tag{10}$$

Preconditioner. Using Definition 1, let $(\Lambda_q, \boldsymbol{E}_q, \lambda_{q+1})$ be the top-q eigensystem of $K(X,X)$, the preconditioner $\mathcal{P}:\mathcal{H}\to\mathcal{H}$ can be explicitly written as, $\mathcal{P} := \mathcal{I} - \sum_{i=1}^{q}\left(1-\frac{\lambda_{q+1}}{\lambda_q}\right)\psi_i\otimes\psi_i$.

Nyström approximate preconditioner. EigenPro 2, introduced by [14], implements a stochastic approximation for $\mathcal{P}$ based on the Nyström extension, thereby reducing the time and memory complexity compared to EigenPro . The first step is to approximate the Hessian operator using the Nyström extension as follows,

$$\mathcal{K}^s := \sum_{k=1}^{s} K(\cdot,\boldsymbol{x}_{i_k})\otimes K(\cdot,\boldsymbol{x}_{i_k}) = \sum_{i=1}^{s}\lambda_i^s\cdot\psi_i^s\otimes\psi_i^s. \tag{11}$$

This is a Nyström approximation of $\mathcal{K}$ using s uniformly random samples from X, referred to as X_s, where $(\Lambda_q^s, \boldsymbol{E}_q^s, \lambda_{q+1}^s)$ represents the corresponding top-q eigensystem of $K(X_s,X_s)$. Using this approximation, we can define the approximated preconditioner as follows, $\mathcal{P}^s := \mathcal{I} - \sum_{i=1}^{q}\left(1-\frac{\lambda_{q+1}^s}{\lambda_q^s}\right)\psi_i^s\otimes\psi_i^s$. For more details on the performance of this preconditioner compared to the case of $s=n$, see [2], who showed that choosing $s\gtrsim\log^4 n$ is sufficient.

Of particular importance is the action of this preconditioner on any function of the form $K(\cdot,A)\boldsymbol{u}$.

$$\mathcal{P}^s K(\cdot,A)\boldsymbol{u} = K(\cdot,A)\boldsymbol{u} - \sum_{i=1}^{q}\left(1-\frac{\lambda_{q+1}^s}{\lambda_q^s}\right)\psi_i\psi_i(A)^\top\boldsymbol{u} \tag{12a}$$

$$= K(\cdot,A)\boldsymbol{u} - \sum_{i=1}^{q}\left(1-\frac{\lambda_{q+1}^s}{\lambda_q^s}\right)\frac{K(\cdot,X_s)\boldsymbol{e}_i}{\sqrt{\lambda_i}}\frac{\boldsymbol{e}_i^\top K(X_s,A)}{\sqrt{\lambda_i}}\boldsymbol{u} \tag{12b}$$

$$= K(\cdot,A)\boldsymbol{u} - K(\cdot,X_s)\boldsymbol{E}_q^s\boldsymbol{D}_q{\boldsymbol{E}_q^s}^\top K(X_s,A)\boldsymbol{u} \tag{12c}$$

where $\boldsymbol{D}_q := \Lambda_q^{-1} - \lambda_{q+1}\Lambda_q^{-2}$.

EigenPro 3. The primary limitation of EigenPro 2 was its inability to handle cases where $Z\neq X$, a necessary condition for disentangling the model and the training set. EigenPro 3 overcomes this limitation by recognizing that although the gradients in (4) may not lie within $\boldsymbol{\mathcal{Z}}$, it is possible to project them back to $\boldsymbol{\mathcal{Z}}$. Consequently, EigenPro 3 can be summarized as follows:

$$f_{t+1} = \text{proj}_{\boldsymbol{\mathcal{Z}}}\left(f_t - \eta\mathcal{P}^s\{\widetilde{\nabla}_f L(f_t)\}\right), \tag{13}$$

where $\text{proj}_{\boldsymbol{\mathcal{Z}}}(u) := \underset{f\in\boldsymbol{\mathcal{Z}}}{\operatorname{argmin}}\ \|u-f\|_{\mathcal{H}}^2$ for any $u\in\mathcal{H}$. As shown in [1, Section 4.2], the exact projection is, $\text{proj}_{\boldsymbol{\mathcal{Z}}}(u) = K(\cdot,Z)K^{-1}(Z,Z)u(Z)$.

This projection can be interpreted as solving a kernel in $\boldsymbol{\mathcal{Z}}$ and can be approximated using EigenPro 2, as done in [1], with a time complexity that scales quadratically with model size. However, since this projection must be performed after each stochastic step, it becomes the most computationally expensive part of the EigenPro 3 algorithm.

3 EigenPro 4: Algorithm Design, Derivation, and Optimization

In this section, we provide a high-level overview and illustrations to highlight the key components of EigenPro 4 and how it significantly reduces training time. We present the EigenPro 4 algorithm in three parts. First, we introduce the algorithm's main components: the pre-projection and projection steps. Then, we detail each of these steps in the following two subsections. Finally, we describe a computational optimization that reduces the runtime of EigenPro 4 by half.

Scaling Challenge: High Projection Overhead. The key development of EigenPro 3 over its contemporaries was that it could train general kernel models in $O(p)$ memory. This was a huge improvement over the prior methods which required $O(p^2)$ memory [20, 19]. However, EigenPro 3 has a high cost $O(mp + p^2)$ per batch of data processed, as summarized in the table in Figure 1. This is especially expensive when $m \ll p$, i.e., when the batch size m is small compared to the model size p.

Main Idea: Delayed Projection. To address computational complexity challenges, EP4 amortizes projection costs by delaying them for T iterations. An effective method for selecting T, along with an illustration of the delayed projection mechanism for $T = 4$ (Figure 4), is provided in Appendix B.

In fact, EigenPro 3 can be viewed as a special case of EigenPro 4 when the parameter T is set to 1. Figure 5 in Appendix B illustrates this relationship. Furthermore, as shown in equation (26) in the same appendix, the total training time is minimized when T is proportional to $\frac{p}{m}$, where m denotes the mini-batch size used in SGD. Under this setting, the per-batch training cost becomes $O(p)$.

3.1 Derivation of the EigenPro 4 Algorithm

As mentioned previously T is a crucial hyperparameter that determines the frequency of projection back to $\mathcal{Z}$ after every T steps. Before the projection step T, at every step when a new batch (X_m, y_m) is fetched, it is added to a set defined as the "temporary centers" set, denoted by Z_{tmp}. Starting with an empty set, $Z_{\mathsf{tmp}} = \emptyset$, we continuously add temporary centers to Z_{tmp} until the step count reaches T.

Formally, the prediction function prior to the projection is no longer fixed and is now expanding. The model can be described as follows:

$$f(x) = \overbrace{\sum_{\mathbf{z} \in Z} \alpha_{\mathbf{z}} K(\boldsymbol{x}, \mathbf{z})}^{\text{original model}} + \overbrace{\sum_{\mathbf{z} \in Z_{\mathsf{tmp}}} \beta_{\mathbf{z}} K(\boldsymbol{x}, \mathbf{z})}^{\text{temporary centers}} \tag{14}$$

where $\alpha_{\mathbf{z}}$ refers to the weights corresponding to the original model center $\mathbf{z}$ and $\beta_{\mathbf{z}}$ refers to the weights corresponding to the temporary model centers. The full EigenPro 4 algorithm has been illustrated in Figure 2 and mathematically can be summarized as following,

$$f_t = \begin{cases} \mathrm{proj}_{\mathcal{Z}} \left(f_{t-1} - \eta \mathcal{P}^s \{ \widetilde{\nabla}_f L(f_{t-1}) \} \right), & t \equiv 0 \mod T, \\ f_{t-1} - \eta \mathcal{P}^s \{ \widetilde{\nabla}_f L(f_t) \}, & \text{otherwise.} \end{cases} \tag{15}$$

where $\widetilde{\nabla} L$ is a stochastic gradient of the loss function computed over a mini-batch of data, and $\mathcal{P}^s$ is the preconditioner.

3.2 Steps before the Projection Step: Update for $t < T$

Based on Equation (15), suppose $(X_1, y_1), \ldots, (X_T, y_T)$ are the minibatches of size m, and the initial model is $f_0 = K(\cdot, Z)\boldsymbol{\alpha}$. After $t < T$ step, the following holds,

$$f_t = f_{t-1} - \eta \mathcal{P}^s K(\cdot, X_t)(f_{t-1}(X_t) - y_t) = f_0 - \eta \mathcal{P}^s \sum_{i=1}^{t} K(\cdot, X_i) \boldsymbol{g}_i$$

$$\boldsymbol{g}_i := f_{i-1}(X_i) - y_i$$

Replacing $\mathcal{P}^s$ using equation (11), setting $f_0 = K(\cdot, Z)\boldsymbol{\alpha}$, and following the notation introduced in section 2, let $(\Lambda_q, \boldsymbol{E}_q^s, \lambda_{q+1})$ denote the top-q eigensystem of $K(X_s, X_s)$, where $\boldsymbol{E}_q^s \in \mathbb{R}^{s \times q}$ and $\boldsymbol{D}_q := \Lambda_q^{-1} - \lambda_{q+1} \Lambda_q^{-2}$. Then, we can simplify the update above as follows:

$$f_t = K(\cdot, Z)\boldsymbol{\alpha}_0 - \eta \sum_{i=1}^{t} \left(K(\cdot, X_i) - K(\cdot, X_s) \boldsymbol{E}_q^s \boldsymbol{D}_q \boldsymbol{E}_q^{s^\top} K(X_s, X_i) \right) \boldsymbol{g}_i \tag{16}$$

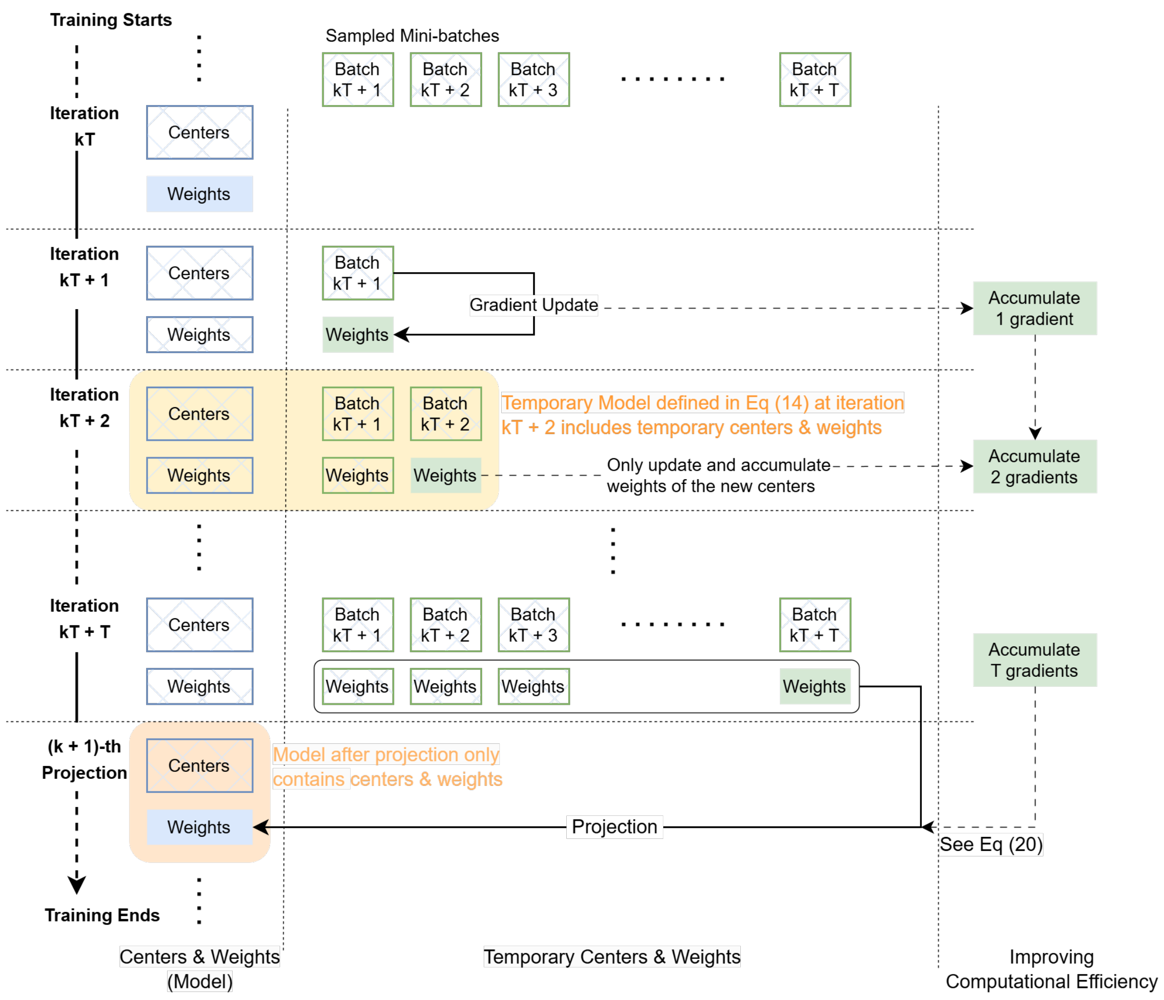

Figure 2: **Schematic of the iterations in EigenPro 4.** The figure illustrates how the model updates are performed over multiple iterations in EigenPro4. Weights are updated using batches, and gradients are accumulated iteratively until a projection step is executed. This approach reduces the computational cost by accumulating gradients before performing the projection, leading to more efficient batch processing.

This update rule implies that after t steps, the weights corresponding to the original centers Z remain unchanged, the weights for the temporary centers X_i are set once to $-\eta \boldsymbol{g}_i$ after they are added, and do not change thereafter, and finally, the weights associated with the Nyström samples X_s are $\eta \sum_{i=1}^{t} \boldsymbol{E}_q^s \boldsymbol{D}_q \boldsymbol{E}_q^{s\top} K(X_s, X_i)\boldsymbol{g}_i$ which can be updated after each batch via an additive update. This is how we update the weights before projection at step T.

3.3 Projection Step: Update for $t = T$

Once, we reach step T we need to project f_T into $\boldsymbol{\mathcal{Z}}$, or formally

$$\begin{aligned} f_T &= \operatorname{proj}_{\boldsymbol{\mathcal{Z}}}\left(f_{T-1} - \eta \mathcal{P}^s\{\widetilde{\nabla}_f L(f_{T-1})\}\right) \qquad (17)\\ &= \operatorname{proj}_{\boldsymbol{\mathcal{Z}}}\left(f_{T-1} - \eta\left(K(\cdot, X_T)\boldsymbol{g}_T - \eta K(\cdot, X_s)\boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} K(X_s, X_T)\boldsymbol{g}_T\right)\right), \end{aligned}$$

Applying Proposition 2 from [1], the solution to this projection problem is as follows,

$$f_T = K(\cdot, Z)K^{-1}(Z,Z)\left(f_{T-1}(Z) - \eta K(Z, X_T)\boldsymbol{g}_T + \eta K(Z, X_s)\boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} K(X_s, X_T)\boldsymbol{g}_T\right) \qquad (18)$$

Algorithm 1 EigenPro 4

```
Require: Data $(X, \boldsymbol{y})$, centers $Z$, batch size $m$, Nyström size $s$, preconditioner level $q$, projection
    period $T$
 1: Fetch subsample $X_s \subseteq X$ of size $s$
 2: $(\Lambda_q, \boldsymbol{E}_q^s, \lambda_{q+1}) \leftarrow$ top-$q$ eigensystem of $K(X_s, X_s)$   and define   $\boldsymbol{D}_q := (\Lambda_q^{-1} - \lambda_{q+1}\Lambda_q^{-2}) \in \mathbb{R}^{q\times q}$
 3: while Stopping criterion is not reached do
 4:     $Z_{\mathsf{tmp}} \leftarrow \emptyset$, $\boldsymbol{\alpha}_{\mathsf{tmp}} \leftarrow \emptyset$, $\boldsymbol{\alpha}_s \leftarrow \mathbf{0}_s$, $\boldsymbol{h} \leftarrow \mathbf{0}_p$
 5:     for $t = \{1, 2, \ldots, T\}$ do
 6:         Fetch minibatch $(X_m, \boldsymbol{y}_m)$ of $m$ samples
 7:         $\boldsymbol{g}_m \leftarrow K(X_m, Z)\boldsymbol{\alpha} + K(X_m, Z_{\mathsf{tmp}})\boldsymbol{\alpha}_{\mathsf{tmp}} + K(X_m, X_s)\boldsymbol{\alpha}_s - \boldsymbol{y}_m$
 8:         $Z_{\mathsf{tmp}}$.append$(X_m)$   and   $\boldsymbol{\alpha}_{\mathsf{tmp}}$.append$(-\eta \cdot \boldsymbol{g}_m)$
 9:         $\boldsymbol{\alpha}_s = \boldsymbol{\alpha}_s + \eta \cdot \boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} K(X_s, X_m)\boldsymbol{g}_m$
10:         $\boldsymbol{h} \leftarrow \boldsymbol{h} + K(Z, X_m)\boldsymbol{g}_m - K(Z, X_s)\boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} K(X_s, X_m)\boldsymbol{g}_m$
11:     end for
12:     $\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \eta \cdot \mathrm{proj}_{\boldsymbol{\mathcal{Z}}}(\boldsymbol{h})$        {Approximate projection implemented using EigenPro 2}
13: end while
```

3.4 Improving Computational Efficiency

Upon careful examination of the derivations in (16), we observe that $f_{T-1}(Z)$ have already been computed previously. This allows us to efficiently reuse $f_{T-1}(Z)$ as follows,

$$f_{T-1}(Z) = K(Z, Z)\boldsymbol{\alpha}_0 - \eta \left(\sum_{i=1}^{T-1} K(Z, X_i)\boldsymbol{g}_i - K(Z, X_s)\boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} K(X_s, X_i)\boldsymbol{g}_i \right) \tag{19}$$

plugging this in (18) we obtain,

$$f_T = K(\cdot, Z)\left(\boldsymbol{\alpha}_0 - \eta K^{-1}(Z, Z)\boldsymbol{h}\right), \tag{20a}$$

$$\boldsymbol{h} := \sum_{i=1}^{T} K(Z, X_i)\boldsymbol{g}_i - K(Z, X_s)\boldsymbol{E}_q^s \boldsymbol{D} \boldsymbol{E}_q^{s\top} \sum_{i=1}^{T} K(X_s, X_i)\boldsymbol{g}_i \tag{20b}$$

3.5 Final algorithm

The final EigenPro 4 can be found in Algorithm 1. Note that we follow the same *inexact projection* scheme used in [1] to approximate the exact projection in the last step of the algorithm.

The benefit of this approximation is that we don't need to solve the problem exactly in $\boldsymbol{\mathcal{X}}$, nor do we need to project back to $\boldsymbol{\mathcal{Z}}$ after each iteration. This approach offers the best of both worlds. In the next section, we demonstrate the effectiveness of this approach compared to prior state-of-the-art methods.

3.6 Convergence Analysis

In Appendix A, we provide a convergence analysis for the special case of Algorithm 1 where $T = \infty$ and exact projection is used in the projection step. However, a more rigorous theoretical analysis that accounts for the approximations introduced for scability—namely, finite T and inexact projections—is left for future work and is beyond the scope of this paper.

4 Numerical experiments

In this section, we demonstrate that our approach achieves orders-of-magnitude speedups over state-of-the-art kernel methods while maintaining comparable or superior generalization performance. We evaluate several kernel methods on the following datasets: (1) CIFAR5M, (2) CIFAR5M[2] [17], (3) ImageNet[1] [6], (4) WebVision[2] [12], and (5) LibriSpeech [18]. Dataset details are provided in Appendix C.

[2]Feature extraction using MobileNetV2

[2]Feature extraction using ResNet-18

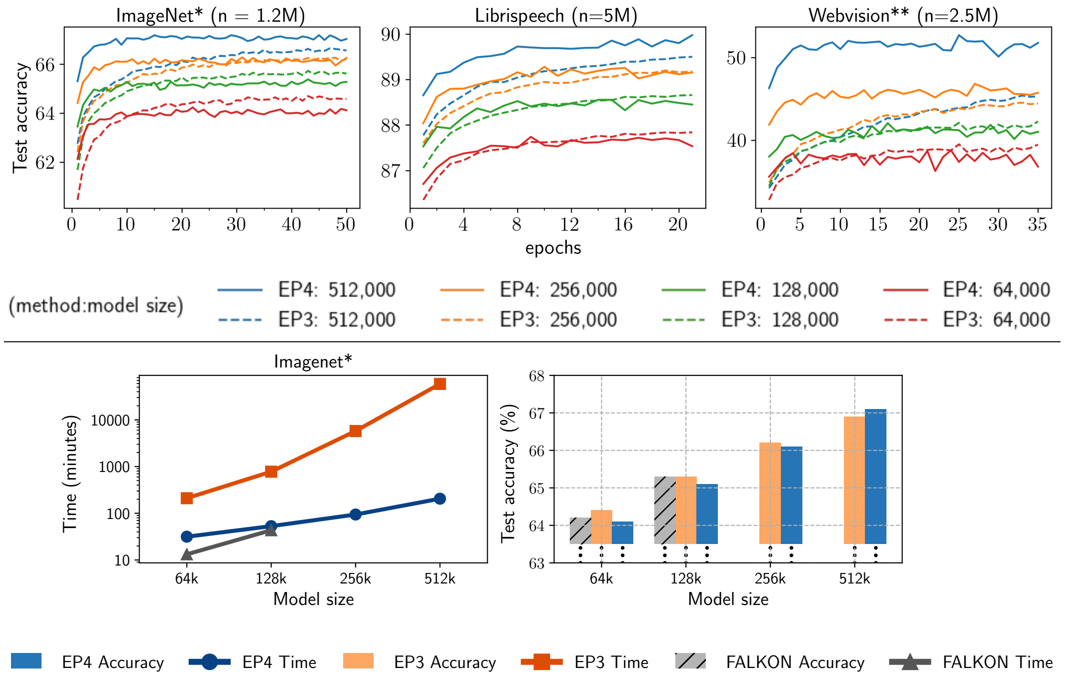

Figure 3: (top) Multi-epoch performance, convergence comparison for EigenPro 3 and EigenPro 4. (bottom) Time and performance comparison for Falkon, EigenPro 3 and EigenPro 4 for ImageNet. Details of the experiments and hardware can be found in Appendix C. (Here n refers to total number of training samples.)

While our method is compatible with any kernel function, we primarily use the Laplace kernel due to its simplicity and strong empirical performance. For completeness, we also report results using the Gaussian and NTK kernels in Appendix C.1.

For multi-class classification, we adopt a one-vs-all decomposition strategy, training a separate binary regressor for each class with targets in $0, 1$. The final prediction is obtained by selecting the class corresponding to the highest predicted value across all binary regressors.

Substantial Reduction in Per-Epoch Training Time. EigenPro 4 has substantially reduced the per-epoch training time, making it the most efficient kernel method on modern machine learning hardware. In contrast to performing projection every mini-batch iteration as in EigenPro 3, EigenPro 4 schedules one projection every few iterations such that its amortized cost is comparable to that of the standard iterations. This results in an ideal per-sample complexity $O(p)$, a remarkable improvement over the $O(p^2)$ complexity from EigenPro 3.

In Table 1, we evaluate the performance and computational timing for a single epoch of our proposed model against established kernel regression methods. As noted earlier, Falkon exhibits limitations due to its quadratic memory complexity. For the CIFAR5M* dataset, training with a model size of 512,000 required 1.3TB of RAM, while scaling to 1M model size necessitated over 5TB. Resource constraints limited our Falkon benchmarks to model sizes of 128,000 and 64,000 for the remaining datasets. While EigenPro 3 addresses these memory constraints, it demonstrates significant computational overhead, particularly evident in the Librispeech dataset where our method, EigenPro 4, achieves a $411\times$ speedup. Notably, EigenPro 4 maintains comparable or superior performance across all evaluated datasets relative to both baseline methods.

Linear Scaling with Model Size. The total training time and memory usage of EigenPro 4 scales linearly with the model size. In comparison, the time required for a single EigenPro 3 iteration grows quadratically with the model size, while the preprocessing time for Falkon grows cubically. Furthermore, the memory demand of Falkon increases quadratically with the model size. In practice, we are unable to run it with large model sizes, e.g., 128,000 centers for ImageNet data.

Model size	Method	CIFAR5M*($n = 5$M)	CIFAR5M ($n = 6$M)	Librispeech ($n = 10$M)	Webvision ($n = 5.2$M)
p = 64K	EigenPro 4	5m (4.6x, 88%)	3m (**15x**, **69%**)	16m (9.1x, **86.8%**)	2m (**45.5x**, **24.3%**)
	EigenPro 3	23m (1x, **88.3%**)	45m (1x, 68.8%)	145m (1x, 85.4%)	91m (1x, 24%)
	Falkon	3m (**7.67x**, 86.1%)	5m (9x, 57.7%)	9m (**16.11x**, 81.0%)	4m (22.75x, 21.7%)
p = 128K	EigenPro 4	5m (**10x**, 88.25%)	4m (**26.25x**, **70.9%**)	19m (**17.95x**, **87.8%**)	4m (**49.75x**, **24.9%**)
	EigenPro 3	50m (1x, **88.42%**)	105m (1x, 70.3%)	341m (1x, 84.75%)	199m (1x, 24.5%)
	Falkon	9m (5.56x, 86.55%)	11m (9.55x, 59.4%)	21m (16.24x, 82.30%)	13m (15.31x, 22.4%)
p = 256K	EigenPro 4	7m (**18.3x**, **88.61%**)	6m (**130.8x**, **71.8%**)	24m (**120x**, **88.33%**)	5m (**106.2x**, **26%**)
	EigenPro 3	128m (1x, **88.61%**)	785m (1x, 70.53%)	$\approx$ 2 days (1x)	531m (1x, 25.52%)
	Falkon	38m (3.37x, 86.73%)	OOM	OOM	OOM
p = 512K	EigenPro 4	12m (**44.25x**, **88.58%**)	10m ($>$ **288x**, 72.9%)	36m ($>$ **200x**, **88.89%**)	11m (**240x**, **27.3%**)
	EigenPro 3	531m (1x, 88.56%)	$>$ 2 days (1x)	$>$ 5 days (1x)	2 days (1x)
	Falkon	240m (2.21x, 86.71%)	OOM	OOM	OOM
p = 1M	EigenPro 4	21m ($>$ **274x**, **88.7%**)	17m ($>$ **508x**, **73.8%**)	70m ($>$ **411x**, **89.5%**)	21m ($>$ **686x**, **29.3%**)
	EigenPro 3	$>$ 4 days (1x)	$>$ 6 days (1x)	>20 days (1x)	$>$ 10 days (1x)
	Falkon	OOM	OOM	OOM	OOM

Table 1: Runtime (in minutes) comparison of EigenPro 4, EigenPro 3, and Falkon across different model sizes and datasets after 1 epoch. The values in parentheses represent the speedup over EigenPro 3 and the final accuracy. OOM indicates an Out-of-Memory error. Details of the experiments and hardware can be found in Appendix C.

We summarize all empirical results in Figure 3 and demonstrate that our method achieves both linear memory complexity and linear time complexity (empirically verified) with respect to model size, offering the best of both worlds. For the ImageNet dataset, we trained all methods until convergence. While EigenPro 3 does not have the quadratic memory scaling problem, Figure 3 shows that even for a relatively small dataset like ImageNet with 1M data points, training a model size of 512,000 centers requires approximately 43 days on a single GPU to reach convergence (about 100 epochs). In contrast, our proposed model achieves convergence in approximately 3 hours, requiring only 15 epochs, with each epoch being significantly more efficient than EigenPro 3 (see Table 1).

Faster Convergence with EigenPro 4. EigenPro 4 generally demonstrates the fastest convergence among all tested methods. In certain cases, such as ImageNet with 1.2 million model centers, EigenPro 4 converges in less than 10% of the epochs needed by other methods, while also delivering superior model performance. Figure 3 compares EigenPro 4 and EigenPro 3 across multiple training epochs, following the experimental setup established in [1]. Despite EigenPro 4's linear time complexity per iteration (compared to EigenPro 3's quadratic complexity), it demonstrates faster convergence with fewer epochs. This efficiency gain is particularly pronounced for larger model sizes, where EigenPro 4 maintains or exceeds EigenPro 3's accuracy while requiring significantly fewer epochs. These results empirically validate that EigenPro 4's algorithmic improvements translate to practical benefits: not only does each iteration run faster, but fewer iterations are needed to achieve optimal performance across diverse datasets. This empirically shows that our model has a linear time complexity with respect to the model size.

5 Conclusion

In this work, we introduced EigenPro 4, an advancement in training large kernel models that achieves linear time complexity per iteration and linear memory scaling with model size. By implementing a delayed projection strategy, we addressed the high computational overhead previously associated with frequent projections, achieving significant time and memory efficiency improvements over EigenPro 3 and Falkon. Our empirical results on diverse datasets highlight EigenPro 4 ability to match or exceed the performance of prior methods with vastly reduced computational resources. Specifically, the algorithm demonstrates both faster convergence and superior scalability, enabling training with model sizes and datasets that were previously infeasible due to memory and time constraints.

Furthermore, EigenPro 4 design opens up new possibilities for parallelization, as it is well-suited for multi-GPU and distributed architectures. Future work will explore these aspects, further expanding its potential in real-world applications requiring efficient, scalable kernel methods for massive data volumes.

Acknowledgements: We acknowledge support from the National Science Foundation (NSF) and the Simons Foundation for the Collaboration on the Theoretical Foundations of Deep Learning through awards DMS-2031883 and #814639, the TILOS institute (NSF CCF-2112665), and the Office of Naval Research (N8644-NV-ONR). This work used ACCESS (Advanced cyberinfrastructure coordination ecosystem: services & support) which is supported by NSF grants numbers #2138259, #2138286, #2138307, #2137603, and #2138296. Specifically, we used the resources from SDSC Expanse GPU compute nodes, and NCSA Delta system, via allocations TG-CIS220009. This work was done in part while AA was visiting the Simons Institute for the Theory of Computing. PP was supported by the DST INSPIRE Faculty Fellowship, Schmidt Sciences, and gifts from SBI and Amazon.

References

[1] Amirhesam Abedsoltan, Mikhail Belkin, and Parthe Pandit. Toward large kernel models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[2] Amirhesam Abedsoltan, Parthe Pandit, Luis Rademacher, and Mikhail Belkin. On the nyström approximation for preconditioning in kernel machines. In *International Conference on Artificial Intelligence and Statistics*, pages 3718–3726. PMLR, 2024.

[3] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[4] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In *International Conference on Machine Learning*, pages 541–549. PMLR, 2018.

[5] Raffaello Camoriano, Tomás Angles, Alessandro Rudi, and Lorenzo Rosasco. Nytro: When subsampling meets early stopping. In *Artificial Intelligence and Statistics*, pages 1403–1411. PMLR, 2016.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Jacob Gardner, Geoff Pleiss, Ruihan Wu, Kilian Weinberger, and Andrew Wilson. Product kernel interpolation for scalable gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1407–1416. PMLR, 2018.

[8] Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks. In *International Conference on Learning Representations*, 2021.

[9] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[10] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[12] Wen Li, Limin Wang, Wei Li, Eirikur Agustsson, and Luc Van Gool. Webvision database: Visual learning and understanding from web data. *arXiv preprint arXiv:1708.02862*, 2017.

[13] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. *Advances in neural information processing systems*, 30, 2017.

[14] Siyuan Ma and Mikhail Belkin. Kernel machines that adapt to gpus for effective large batch training. *Proceedings of Machine Learning and Systems*, 1:360–373, 2019.

[15] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, apr 2017.

[16] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. *Advances in Neural Information Processing Systems*, 33:14410–14422, 2020.

[17] Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi. The deep bootstrap framework: Good online learners are good offline generalizers. *International Conference on Learning Representations*, 2021.

[18] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[19] Pratik Rathore, Zachary Frangella, Jiaming Yang, Michał Dereziński, and Madeleine Udell. Have askotch: A neat solution for large-scale kernel ridge regression. *arXiv preprint arXiv:2407.10070*, 2025.

[20] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30, 2017.

[21] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814, 2007.

[22] Michalis Titsias. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR, 2009.

[23] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr. Xsede: Accelerating scientific discovery. *Computing in Science & Engineering*, 16(05):62–74, sep 2014.

[24] Shinji Watanabe, Takaaki Hori, Shigeki Karita, Tomoki Hayashi, Jiro Nishitoba, Yuya Unno, Nelson Enrique Yalta Soplin, Jahn Heymann, Matthew Wiesner, Nanxin Chen, Adithya Renduchintala, and Tsubasa Ochiai. ESPnet: End-to-end speech processing toolkit. In *Proceedings of Interspeech*, pages 2207–2211, 2018.

[25] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[26] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13, 2000.

[27] Andrew Wilson and Hannes Nickisch. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International conference on machine learning*, pages 1775–1784. PMLR, 2015.

[28] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We demonstrate the efficiency of our method across multiple datasets and scales: Figures 1 and 2 illustrate performance trends, and Table 1 provides an extensive quantitative evaluation, showing speedups of up to $\times 600$.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide a theoretical convergence analysis for the exact version of our method in Appendix A. The full algorithm is based on an efficient approximation of this exact version. Although a formal theoretical analysis of the approximation is not included, we validate its effectiveness extensively through experiments on diverse datasets and scales. Establishing a complete theoretical understanding of the approximation involves nontrivial technical challenges, which we consider an important direction for future work beyond the scope of this paper. We addressed the limitation in 3.6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical result is presented in Appendix A, where we include all necessary assumptions and provide a complete proof.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all necessary details for reproducing our experiments in Appendix C. We will release our GitHub repository upon acceptance and have also included a `.zip` file version of the code in supplementary for the review process.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case

of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
    (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
    (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
    (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
    (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We use publicly available datasets and provide all necessary details for reproducing our experiments in Appendix C. The code for our algorithm is included in the supplemental material for review process.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide all thethe necessary details for reproducing our experiments in Appendix C.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Our experimental setup ensures stability and reproducibility by using the same training/test splits, center selections, and random seeds across all methods. As is typical in kernel methods, there is no randomness from model or weight initialization, and the only minor source of stochasticity arises from optimization, which is consistently controlled. Because all methods are evaluated under identical conditions, traditional error bars would not provide meaningful additional information. We validate the effectiveness and robustness of our method through extensive experiments on a wide range of datasets, consistently demonstrating significant speedups with comparable or better performance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide all necessary details in Appendix C.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: We followed the NeuIPS Code of Ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: We believe there is no immediate societal impact, either positive or negative, as this work focuses on foundational algorithmic improvements to kernel methods. The contributions are methodological in nature and not tied to any specific application domain or deployment scenario. As such, we mark this question as not applicable.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out

that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This work does not involve the release of pretrained models, generative systems, or scraped datasets that pose a high risk of misuse. Therefore, no specific safeguards are necessary, and we mark this question as not applicable.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: To the best of our knowledge, all datasets and codebases used in this work have been properly cited and used in accordance with their licenses and terms of use.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We introduce new code implementing our algorithm, which we include as a `.zip` file in the supplemental material for the review process. The code is documented with instructions for reproduction and GitHub repository will be released publicly upon acceptance.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This work does not involve any crowdsourcing or research with human subjects, so this question is not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This work does not involve any research with human subjects, and therefore no IRB or equivalent approval was necessary.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: This work does not involve the use of large language models as an important, original, or non-standard component of the core methodology. Any use of language models was limited to minor writing assistance and does not impact the scientific contribution of the paper.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Convergence analysis

In this section, we derive EigenPro4.0-Exact (Algorithm 2), a precursor to EigenPro4.0. However, this version does not scale efficiently. In Section 3, we enhance its scalability by introducing stochastic approximations, resulting in EigenPro4.0 (Algorithm 1).

Recall that the derivatives of the loss function, as defined in (13), lie in the span of the training data, denoted as $\boldsymbol{\mathcal{X}}$. However, these derivatives cannot directly update the model, which resides in the span of the model centers, $\boldsymbol{\mathcal{Z}}$. To address this, we first fit the labels within the $\boldsymbol{\mathcal{X}}$ and then project the solution into the $\boldsymbol{\mathcal{Z}}$. This process is repeated iteratively on the residual labels until convergence, as outlined in Algorithm algorithm 2.

Algorithm 2 EigenPro 4-Exact

```
Require: Data $(X, \boldsymbol{y})$, centers $Z$
1: $\tilde{\boldsymbol{y}}_0 = \boldsymbol{y}$
2: for $t = 1, 2, \ldots$ do
3:     $\boldsymbol{\alpha}_t = K^{-1}(X, X)\tilde{\boldsymbol{y}}_t$
4:     $K(\cdot, Z)\boldsymbol{\beta}_t = \text{proj}_{\boldsymbol{\mathcal{Z}}}\left(K(\cdot, X)\boldsymbol{\alpha}_t\right)$
5:     $\tilde{\boldsymbol{y}}_{t+1} = \boldsymbol{y} - K(X, Z)\boldsymbol{\beta}_t$
6: end for
```

The following proposition provides the fixed point analysis for this algorithm.

Proposition 1. *Consider any dataset $X, \boldsymbol{y}$ and a choice of model centers Z, with a kernel function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. Assume that $K(X, X)$ and $K(Z, X)$ are full Rank. Then, Algorithm 2 converges to the following solution:*

$$\hat{f} = K(\cdot, Z)\left(K(Z, X)K^{-1}(X, X)K(X, Z)\right)^{-1} K(Z, X)K^{-1}(X, X)\boldsymbol{y}. \tag{21}$$

Furthermore, if $\boldsymbol{y} = K(X, Z)\boldsymbol{\beta}^* + \boldsymbol{\xi}$, where $\boldsymbol{\xi}$ is a vector of independent centered random noise with $\mathbb{E}[\xi_i^2] = \sigma^2$, then

$$\lim_{t\to\infty} \mathbb{E}[\boldsymbol{\beta}_t] = \boldsymbol{\beta}^*, \quad \lim_{t\to\infty} \frac{\mathbb{E}[\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|^2]}{\sigma^2} = \\ \operatorname{tr}\left(\left(K(Z, X)K^{-1}(X, X)K(X, Z)\right)^{-2} K(Z, X)K^{-2}(X, X)K(X, Z)\right).$$

Proof. We begin by expressing Algorithm 2 recursively and substituting $\text{proj}_{\boldsymbol{\mathcal{Z}}}$ with the expression in (18). Recall that $f_t = K(\cdot, Z)\boldsymbol{\beta}_t$ with base case $\boldsymbol{\beta}_0 = 0$. The update rule for $\boldsymbol{\beta}_t$ is given by:

$$\boldsymbol{\beta}_t = K^{-1}(Z, Z)K(Z, X)K^{-1}(X, X)(\boldsymbol{y} - K(X, Z)\boldsymbol{\beta}_{t-1}) + \boldsymbol{\beta}_{t-1}. \tag{22}$$

Let us define the matrices:

$$B := K^{-1}(Z, Z)K(Z, X)K^{-1}(X, X), \quad C := BK(X, Z) - I,$$

which allows us to rewrite the recursion more succinctly:

$$\begin{aligned} \boldsymbol{\beta}_t &= B(\boldsymbol{y} - K(X, Z)\boldsymbol{\beta}_{t-1}) + \boldsymbol{\beta}_{t-1} \\ &= B\boldsymbol{y} - C\boldsymbol{\beta}_{t-1} = B\boldsymbol{y} - CB\boldsymbol{y} + C^2\boldsymbol{\beta}_{t-2} \\ &\quad\vdots \\ &= \left(\sum_{i=0}^{t-1}(-1)^i C^i\right) B\boldsymbol{y}. \end{aligned} \tag{23}$$

As the number of iterations tends to infinity, we can define the infinite series sum:

$$S := \sum_{i=0}^{\infty}(-1)^i C^i.$$

Observe that:

$$S + CS = I.$$

Substituting the definition $C = BK(X,Z) - I$ and $B = K^{-1}(Z,Z)K(Z,X)K^{-1}(X,X)$, we have:

$$K^{-1}(Z,Z)K(Z,X)K^{-1}(X,X)K(X,Z)S = I.$$

Thus, this simplifies to:

$$S = \left(K(Z,X)K^{-1}(X,X)K(X,Z)\right)^{-1} K(Z,Z).$$

Therefore, the final solution converges to:

$$\hat{f} = K(\cdot, Z) \left(K(Z,X)K^{-1}(X,X)K(X,Z)\right)^{-1} K(Z,X)K^{-1}(X,X)\boldsymbol{y}. \tag{24}$$

Substituting $\boldsymbol{y} = K(X,Z)\boldsymbol{\beta}^* + \boldsymbol{\xi}$ readily completes the second claim.

$\square$

line in Algorithm 1	computation	flops
7	$K(X_m, Z)\boldsymbol{\alpha} - \boldsymbol{y}_t$	mp
7	$K(X_m, Z_{\mathsf{tmp}})\boldsymbol{\alpha}_{\mathsf{tmp}}$	$m^2(t-kT-1)$
7	$K(X_m, X_s)\boldsymbol{\alpha}_s$	ms
9,10	$\boldsymbol{h}_1 := \boldsymbol{F}^\top K(X_s, X_m)\boldsymbol{g}_m \in \mathbb{R}^q$	$ms + sq$
9	$\boldsymbol{F}\boldsymbol{h}_1$	sq
10	$K(Z, X_m)\boldsymbol{g}_m$	mp
10	$\boldsymbol{M}\boldsymbol{h}_1$	pq

Table 2: Computational cost analysis of Algorithm 1 for processing batch t for $kT < t \le (k+1)T$ for some $k \in \mathbb{N}$.
The cost of processing batch t without the post-processing adds up to $2mp + 2ms + 2sq + pq + m^2(t - kT - 1)$ flops.

B Computational complexity comparison

We assume that EigenPro4 is processing T batches of data at once before running the post-processing step of projection. Here we show we calculated the optimal value of T.

Cost for processing t^{th} batch of data. For a some $k \in \mathbb{N}$, let $kT < t \le (k+1)T$. See Table 2.

Cost of processing T batches of data before post-processing The total cost for processing T batches $t = kT + 1$ to $t = (k+1)T$ before the projection is the sum of the above

$$T(2mp + 2ms + 2sq + pq) + m^2 \sum_{t=kT+1}^{(k+1)T} (t - kT - 1) = T(2mp + 2ms + 2sq + pq) + m^2 \frac{T(T-1)}{2} \tag{25}$$

Average of processing T batches of data with post-processing Assuming the post processing involves T_{ep2} epochs of EigenPro 2, the average cost of processing T batches is

$$\frac{T(2mp + 2ms + 2sq + pq) + m^2 \frac{T(T-1)}{2} + p^2 T_{\mathsf{ep2}}}{T} \tag{26}$$

A simple calculation shows that

$$T^\star = \frac{p}{m}\sqrt{2T_{\mathsf{ep2}}} \tag{27}$$

minimizes the average time above. The average cost of processing a batch is thus

$$2mp(1 + \sqrt{2T_{\mathsf{ep2}}}) + 2ms + 2sq + pq \tag{28}$$

Illustration for delayed projection for $T = 4$.

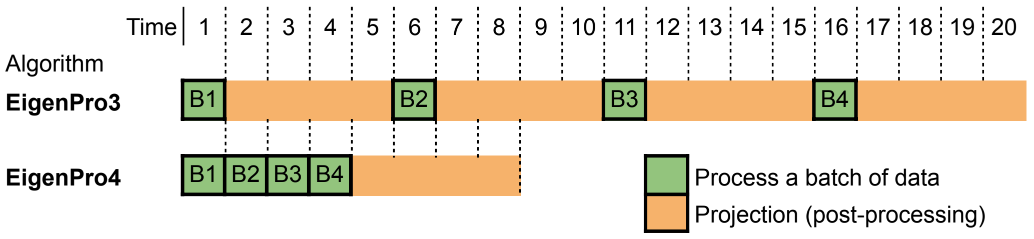

Figure 4: **Design of EigenPro4.** An illustration of how batches of data are processed by the two algorithms. EigenPro3 involves an expensive *projection* step when processing every batch of data. EigenPro4 waits for multiple batches to be processed before running the projection step for all of them together. This reduces the amortized cost for processing each batch.

Comparison between EigenPro 4 and EigenPro 3. Figure 5 shows how EigenPro 4 and EigenPro 3 perform over training iterations. EigenPro 4 accuracy improves between projections and drops after each projection step. While EigenPro 3 projects at every step, EigenPro 4 maintains comparable accuracy with fewer projections. The left panel of Figure 5 confirms that both methods reach similar final accuracy, while the right panel shows EigenPro 4 significant speed advantage. With continued training, EigenPro 4 accuracy drops from projections become progressively smaller.

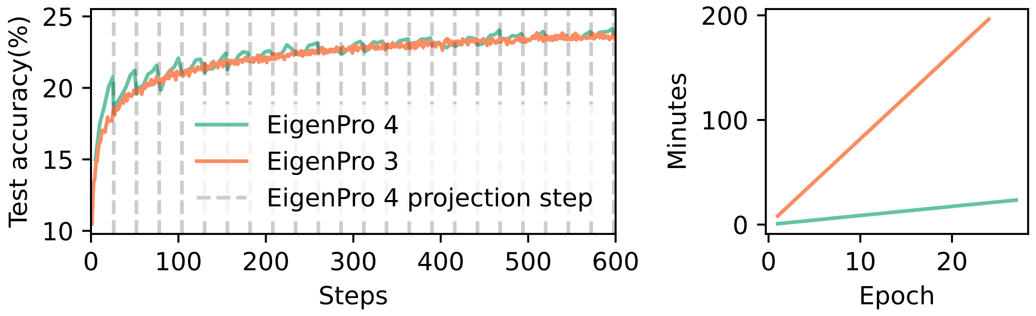

Figure 5: Performance and computational time comparison between EigenPro 4.0 ($T = 11$) and EigenPro 3.0 (equivalent to $T = 1$), highlighting the impact of the projection step on the performance of EigenPro 4.0. The detail of the experiment can be found in Appendix C.

Algorithm	setup	FLOPS per batch*	Memory
EigenPro 4.0	$O(s^2q)$	$2mp(1+\sqrt{2T_{\text{ep2}}})+2ms+2sq+pq$	$s^2+p(1+\sqrt{2T_{\text{ep2}}})$
EigenPro 3.0	$O(s^2q)$	$2mp+p^2T_{\text{ep2}}+2ms+2sq+pq$	s^2+p
Falkon	$O(p^3)$	$2mp$	p^2

Table 3: **Comparing complexity of algorithms.** Number of training samples n, number of model centers p, batch size m, Nyström sub-sample size s, preconditioner level q. Here we assumed only a constant number of epochs of EigenPro 2.0 is needed for large scale experiments. Cost of kernel evaluations and number of classes are assumed to be $O(1)$, also it is reasonable to assume $p \gg s \gg q$.
* FLOPS per iteration reported are amortized over multiple batches processed.

C Experiments Results

C.1 Other kernels

For consistency with prior work, we primarily used the Laplace kernel, which offers efficient per-batch computation and scales linearly with total training time—making it feasible to train methods such as EigenPro 3.0 at larger scales. For completeness, we also report results using alternative kernels: Gaussian (bandwidth 20) and NTK (ReLU MLP, depth 2), evaluated on the same CIFAR5M embeddings as in Figure 1.

Table 4: Comparison of kernel types and number of centers.

Kernel Type	64k Centers	128k Centers	256k Centers
Gaussian	87.77% (178s)	87.80% (194s)	88.14% (233s)
NTK	87.01% (206s)	87.34% (309s)	87.54% (564s)

C.2 Computational resources used

This work used the Extreme Science and Engineering Discovery Environment (XSEDE) [23]. We used machines with NVIDIA-V100, NVIDIA-A100 and NVIDIA-A40 GPUs, with a V-RAM up to 1.3 T, and 8x cores of Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with a RAM of 100 GB. NOte that we had 1.3T of RAM for just one experiment CIFAR5M*, for the rest of expermients we where constraint with 400G of RAM.

C.3 Datasets

We perform experiments on these datasets: (1) CIFAR10, [11], (2) CIFAR5M, [17], (3) ImageNet, (4) Webvision.[12], and (5) librispeech.

CIFAR5M. In our experiments, we utilized both raw and embedded features from the CIFAR5M data-set. The embedded features were extracted using a MobileNetv2 model pre-trained on the ImageNet data-set, obtained from *timm* library [25]. We indicate in our results when pre-trained features were used by adding an asterisk (*) to the corresponding entries.

ImageNet. In our experiments, we utilized embedded features from the ImageNet data-set. The embedded features were extracted using a MobileNetv2 model pre-trained on the ImageNet dataset, obtained from *timm* library [25]. We indicate in our results when pre-trained features were used by adding an asterisk (*) to the corresponding entries.

Webvision. In our experiments, we utilized embedded features from the Webvision data-set. The embedded features were extracted using a ResNet-18 model pre-trained on the ImageNet dataset, obtained from *timm* library [25]. Webvision data set contains 16M images in 5K classes. However, we used only a subset of it for different experiments.

Important note about Webvision dataset. In the experiments shown in Figure 3, to ensure computational feasibility for running EigenPro 3.0, we used only data with labels smaller than 500, corresponding to roughly 2.5 million data points. In contrast, the results in Table 1 were obtained using labels up to 1000, encompassing about 5.2 million data points. Since this larger label range represents a more challenging classification problem, the test accuracy in Figure 3 is higher than in Table 1.

Librispeech. Librispeech [18] is a large-scale (1000 hours in total) corpus of 16 kHz English speech derived from audio books. We choose the subset train-clean-100 and train-clean-300 (5M samples) as our training data, test-clean as our test set. The features are got by passing through a well-trained acoustic model (a VGG+BLSTM architecture in [8]) to align the length of audio and text. It is doing a 301-wise classification task where different class represents different uni-gram [10]. The implementation of extracting features is based on the ESPnet toolkit [24].

C.4 Experiments details

Figure 1 This experiment used CIFAR5M* data set, where embedding has been generated using a pre-trained mobile-net network mentioned earlier. this is the only experiment that we had access to 1.3T of VRAM. We set the bandwidth to 5.0 and use $1k$ Nystrom samples with preconditioning level of size 100. We used float16 for this experiment.

Figure 5 This experiment has been run over Webvision data set with extracted embedding through Resnet18. the model size here is set to $100k$ number of centers. The bandwidth used is 5.0, $1k$ Nystrom samples with preconditioning level of size 100. We used float16 for this experiment.

Figure 3 We follow the setting in [1]. The bandwith used here is 20 for Librispeach and Webvision and 16 for imagnet. Here again we used extracted feature of these datasets mentioned earlier. The precision used here is float32. with $10k$ Nystrom samples with preconditioning level of size 1000.

Table 1 For all datasets here we used bandwidth of 5.0 with $1k$ Nystrom samples with preconditioning level of size 100. We used float16 for all dataset except for Librispeach where we used float32. Further, we note that Falkonlatest library ran out of GPU memory for model sizes larger than 256000 number of centers that is the reason we could not run it for 256000. And as mentioned for model sizes 521000 and above the algorithm has inherent quadratic scaling with respect to model size and we ran out of VRAM. In the plot we refer to both of these memry issues as OOM.