

DROP: Optimizing Stochastic Dimensionality Reduction via Workload-Aware Progressive Sampling

Sahaana Suri, Peter Bailis
Stanford University

ABSTRACT

Dimensionality reduction is a critical step in machine learning pipelines. Principal Component Analysis (PCA) is frequently the method of choice, yet is often prohibitively expensive. Theoretical means of accelerating PCA via sampling have been proposed, but typically treat PCA as a reusable statistical operator, independent of downstream analytics workflows. We show how accounting for downstream analytics operations during dimensionality reduction via PCA allows stochastic methods to efficiently operate over very small (e.g., 1%) subsamples of input data, thus reducing computational overhead and end-to-end runtime. This enables end-to-end optimization over both dimensionality reduction and analytics tasks. By combining techniques spanning progressive sampling, approximate query processing, and cost-based optimization, we propose a dimensionality reduction optimizer that enables speedups of up to **5×** over **Singular-Value-Decomposition-based** PCA techniques, and achieves parity with or exceeds conventional approaches like FFT and PAA by up to **16×** in end-to-end workloads.

1 INTRODUCTION

Continued, rapid growth in high-dimensional data from automated data sources [11, 39] poses a scalability challenge for machine learning (ML) pipelines. Dimensionality reduction (DR) techniques can alleviate this scalability challenge [15, 26, 40, 42]. In exchange for a runtime cost (R), DR methods transform an n -dimensional dataset to a lower k -dimensional representation while preserving salient dataset features, allowing downstream analytics routines to terminate in runtime proportional to $k > n$, while preserving accuracy on downstream tasks (see Figure 1).

Principal Component Analysis (PCA) is often the DR method of choice for practitioners with respect to transformation

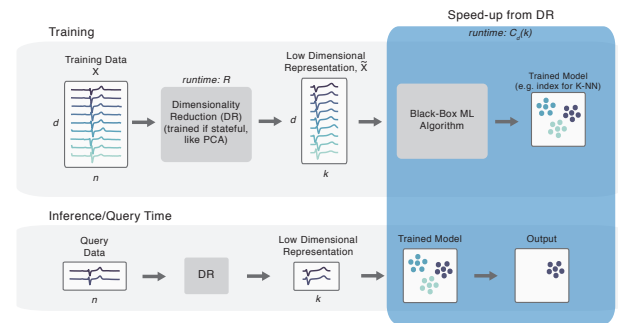


Figure 1: Sample machine learning pipeline with dimensionality reduction. Spending time on DR provides downstream runtime speed-ups.

quality (k) for a target accuracy [38]. However, naïve implementations of PCA are data and workload-independent, and scale poorly with dimensionality, resulting in runtimes (R) that exceed the downstream runtime benefit of DR via PCA. Accordingly, practitioners often sacrifice quality for end-to-end runtime, and use alternative DR methods [22].

Sample-based, stochastic PCA algorithms [19, 63] are a scalable alternative to classical PCA. However, the amount of sampling required is highly data-dependent. If we sample too many data points, then the runtime overhead of PCA in an end-to-end analytics workload could outweigh the statistical benefits. If we fail to sample enough data points, then PCA could fail to deliver a sufficiently high-quality reduction and compromise the runtime and/or accuracy of downstream analytics. This raises a critical question: can we develop a data and workload-dependent means of efficiently and accurately determining the sampling rate for stochastic PCA, such that we can obtain PCA’s transformation quality while minimizing end-to-end workload runtime?

To this end, we develop DROP, a system that performs whole-workload runtime optimization by dynamically identifying the amount of sampling required for stochastic PCA. DROP takes a high-dimensional dataset,¹ property to preserve (e.g., pairwise Euclidean distance to 5%), and optional

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

DEEM’19, June 2019, Amsterdam, The Netherlands

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

¹Our primary focus for performance evaluation is a case study on time series similarity search, given the amount of study in the database community [22] and the resurgence of interest in time series analytics systems [10, 11, 59]. We provide a preliminary generalizability analysis in Section 5.

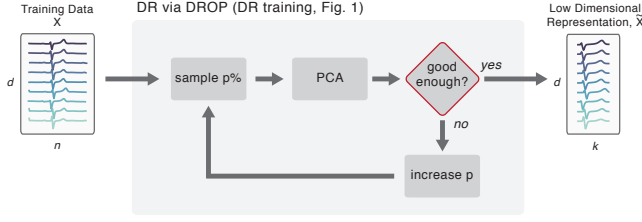


Figure 2: DROP is a workload-aware DR operator compatible with standard ML pipelines. The challenge DROP solves is when to stop training (“good enough?”)

runtime model expressing downstream workload performance as a function of dimensionality (e.g., for k-Nearest Neighbors [k-NN], runtime is linear in dimensionality). DROP returns a low-dimensional transformation for the input using as few samples as needed to minimize the projected overall workload runtime while satisfying quality constraints.

To achieve this functionality, DROP addresses the question of how much to sample the input dataset via data-dependent progressive sampling and online progress estimation at runtime. DROP performs PCA on a small sample to obtain a candidate transformation, then progressively increases the number of samples until termination (see Figure 2). To identify the termination point that minimizes the overall runtime, DROP must overcome three challenges:

First, given the results of PCA on a data sample, DROP must *evaluate the quality* of the current candidate transformation. Popular analytics and data mining tasks often require approximate preservation of metrics such as average pairwise distances between data points [25, 34], which are costly to compute. Thus, DROP adapts confidence intervals for fast estimation of the input metric to preserve.

Second, DROP must *estimate the marginal benefit of sampling additional datapoints* for another iteration. When running PCA on a series of progressively larger samples, later samples will incur higher computational cost but may return lower-dimensional transformations. To navigate this trade-off between end-to-end runtime and transformation quality, DROP uses the results obtained from previous iterations to build a predictive performance model for future iterations.

Finally, given the current quality and expected marginal benefit of the next iteration, DROP must *optimize end-to-end runtime*. While an application-agnostic approach would iterate until successive iterations yield no benefit to quality, many analytics operators such as k-Nearest Neighbors are tolerant of error [26], so it is frequently advantageous to trade a slightly higher-dimensional basis for faster pre-processing (DR). To address this challenge, the system performs workload-specific optimization to minimize the expected runtime of the complete end-to-end analytics pipeline.

We view DROP as a pragmatic combination of recent theoretical advances in dimensionality reduction and classic techniques from approximate query processing, and a useful system for performing end-to-end workflow optimization. We make the following contributions in this work:

- We show the data sample required to perform accuracy-achieving PCA is often small (as little as 1%), and data-dependent sampling can enable **91×** speedup compared to PCA via singular value decomposition (SVD).
- We propose DROP, an online optimizer for DR that uses information about downstream analytics tasks to perform efficient stochastic PCA.
- We present techniques based on progressive sampling, approximate query processing, online progress estimation, and cost based optimization to enable up to **5×** faster end-to-end execution over PCA via SVD.

2 PROBLEM STATEMENT AND BACKGROUND

In this section, we provide background on dimensionality reduction (DR), and then define our problem of workload-aware dimensionality reduction (DR). Finally, we revisit a widely cited empirical comparison of DR techniques from VLDB 2008 [22] **that we use as a case study to validate** that Principal Component Analysis (PCA) can outperform classic techniques, but at a high computational cost.

2.1 Dimensionality Reduction

DR refers to finding a low-dimensional representation of a dataset that preserves properties of interest, such as data point similarity [18, 28]. Formally, consider a data matrix $X \in \mathbb{R}^{d \times n}$, where each row i corresponds to data point $x_i \in \mathbb{R}^n$, with $d > n$. DR computes a transformation function ($T : \mathbb{R}^n \rightarrow \mathbb{R}^k$) that maps each x_i to a new basis as $\tilde{x}_i \in \mathbb{R}^k$ where $k \leq n$, resulting in a new data matrix $T(X) = \tilde{X} \in \mathbb{R}^{d \times k}$.

Principal Component Analysis (PCA). PCA is a linear DR technique that identifies a new orthogonal basis for a dataset that captures its directions of highest variance. Of all linear transformations, this basis minimizes reconstruction error in a mean square sense. Classically implemented PCA uses a Singular Value Decomposition (SVD) routine [65].

2.2 DR for Repeated-Query Workloads

In workloads such as similarity search, clustering, **or classification**, ML models are periodically trained over historical data, and are *repeatedly queried* as incoming data arrives or new query needs arise (see Fig 1). Indexes built over this data can improve the efficiency of this repeated query workload in exchange for a preprocessing overhead. DR with a multidimensional index structure is a classic way of achieving this,

and is the basis for popular similarity search procedures and extensions in the data mining and machine learning communities [7, 15, 34, 40, 42, 49, 60, 73]; a metric-preserving transformation reduces input dimensionality, and an index is built in this new space for subsequent queries.

DR in Similarity Search. Similarity search is a common repeated-query workload performed over a variety of data types including images, documents and time series [22, 29], which we use as a running case study given its popularity and the large amount of research in the space. The Tightness of Lower Bounds (*TLB*) is typically the metric preserved by DR in this setting [22], as it measures how well a *contractive* DR transformation (i.e. distances in the transformed space are less than or equal to those in the original) preserves pairwise Euclidean distances. This can be used to identify the quality of a low dimensional transformation without performing the downstream similarity search task:

$$TLB = \frac{2}{d(d-1)} \sum_{i < j} \frac{\|\tilde{x}_i - \tilde{x}_j\|_2}{\|x_i - x_j\|_2}. \quad (1)$$

2.3 Problem: Workload-Aware DR

In end-to-end analytics tasks, we wish to minimize the combined runtime of DR and downstream applications. DR is a fixed cost (i.e., in index construction for similarity search), while each query (i.e., nearest neighbor query) over the dataset incurs a marginal cost that is dependent on DR quality: lower-dimensional data points result in faster queries.

In workload-aware dimensionality reduction, we perform DR to minimize overall workload runtime. As input, we consider a set of data points, a desired level of metric preservation (B ; default TLB , e.g., $TLB \geq .99$) and, optional downstream runtime as a function of dimensionality ($C_d(n)$ for an $d \times n$ data matrix). We use this information to efficiently return a DR function that satisfies the metric constraint with a configurable degree of confidence. More formally, denoting DR runtime as R , we define the optimization problem:

Problem 2.1. Given $X \in \mathbb{R}^{d \times n}$, TLB constraint $B \in (0, 1]$, confidence c , and workload runtime function $C_d : \mathbb{Z}_+ \rightarrow \mathbb{R}_+$, find k and transformation matrix $T_k \in \mathbb{R}^{n \times k}$ that minimizes $R + C_d(k)$ such that $TLB(XT_k) \geq B$ with confidence c .

We assume the downstream runtime model $C_d(n)$ is monotonically increasing in n as the premise of DR for efficient analytics relies on downstream tasks running faster on lower dimensional data. Absent this, we default to execution until convergence (i.e. until k plateaus) as described in Section 3, and demonstrate the cost of doing so in Section 5.

The more time spent on DR (R), the smaller the transformation (k), thus the lower the workload runtime. To minimize $R + C_d(k)$, we must determine how much time to spend on DR to minimize end-to-end runtime.

DEEM'19, June 2019, Amsterdam, The Netherlands

2.4 Case Study: PCA Speed vs. Quality

While improved quality provides faster repeated query execution (as seen in Section 5), the cost of DR via PCA dominates this speedup, encouraging the use of faster, lower-quality alternatives [22]. This motivates our study of downstream-workload-aware, stochastic, sampling-based PCA methods.

To briefly quantify this trade-off, we augment a widely-cited time series similarity search DR study from VLDB 2008 [22] by evaluating PCA—which the authors did not benchmark due to it being “untenable for large data sets” despite providing “optimal linear dimensionality reduction.” We compare PCA via SVD to baseline techniques based on both runtime and DR performance with respect to TLB over the largest datasets from [22]. We use two of their fastest methods as our baselines since they show the remainder exhibited “very little difference”: Fast Fourier Transform (FFT) and Piecewise Aggregate Approximation (PAA). We verify that PCA offers more effective dimensionality reduction than alternative techniques for time series similarity search, but with a large computational overhead.

TLB Performance Comparison We compute the minimum dimensionality (k) achieved by each technique subject to a TLB constraint. On average across the datasets, PCA provides bases that are $2.3\times$ and $3.7\times$ smaller than PAA and FFT for $TLB = 0.75$, and $2.9\times$ and $1.8\times$ smaller for $TLB = 0.99$. While the margin between PCA and alternatives is dataset-dependent, PCA almost always preserves TLB with a lower dimensional representation.

Runtime Performance Comparison PCA implemented via out-of-the-box SVD is on average over $26\times$ (up to $56\times$) slower than PAA and over $4.6\times$ (up to $9.7\times$) times slower than FFT when computing the smallest TLB -preserving basis. This substantiates the observation that classic PCA is incredibly slow compared to alternatives [22].

3 SAMPLE-BASED COMPUTATION

We **augment our case study** to show that running PCA on data samples does not sacrifice DR quality, but that the number of samples required varies per dataset. We then show how progressively increasing the sampling rate can help dynamically identify how much to sample a given dataset. AND TO SOLVE OUR PROBLEM.

3.1 Feasibility of Sampling

Many real-world **datasets** are intrinsically low-dimensional, as evidenced by their rapid falloff in their eigenvalue spectrum. A data sample thus captures much of the dataset’s “interesting” behavior, so fitting a model over such a sample will generalize well. We verify this phenomenon by varying the target TLB and examining the minimum number of

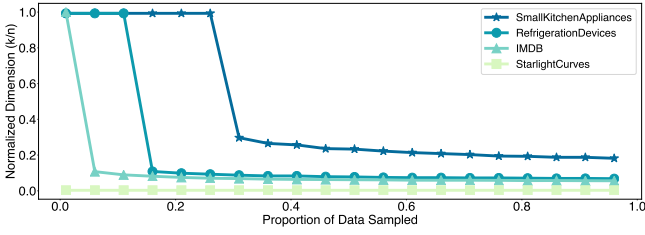


Figure 3: Improvement in representation size for $TLB = 0.80$ across three datasets. Higher sampling rates improve quality until reaching a state equivalent to running PCA over the full dataset ("convergence")

samples required to obtain a TLB -preserving transform with output dimension k equal to input dimension n .

On average, across the considered UCR time series datasets, a sample of under 0.64% (up to 5.5%) of the input is sufficient for a TLB of 0.75, and under 4.15% (up to 38.6%) is sufficient for a TLB of 0.99. When this proportion is known a priori, we obtain up to **91 \times speedup** compared to a naïve implementation of PCA via SVD—with no algorithmic improvement.

3.2 Incremental, Progressive Sampling

Sampling benefit is dataset-dependent; we must identify how large a sample suffices to compute high-quality transforms. Figure 3 shows how the dimensionality required to attain a given TLB changes when we vary dataset and proportion of data sampled. Increasing the number of samples provides lower dimensional transformations for the same quality. However, this decrease in dimension plateaus as the number of samples increases. Thus, we must determine when the downstream value of decreased dimension is overpowered by the cost of additional DR—that is, whether to sample to convergence (evaluated in Section 5.3) or terminate early (e.g., at 0.3 proportion of data sampled for SmallKitchenAppliances).

4 DROP: WORKLOAD OPTIMIZATION

We now introduce DROP, a system that performs progressive sampling and online progress estimation to control the amount of sampling to minimize overall runtime. DROP answers a crucial question that stochastic PCA techniques have traditionally ignored: how long should these methods run?

4.1 DROP Architecture

DROP operates over a series of progressively larger data samples, and determines when to terminate via a four-step procedure that is repeated for each iteration: progressive sampling, transformation evaluation, progress estimation, and cost-based optimization:

Step 1: Progressive Sampling (§4.2, Alg 1 L5, Fig 2A)

Algorithm 1 DROP Algorithm

Input:

X : data matrix

B : target metric preservation level

C_d : cost of downstream operations; default tuned to k -NN

Output:

T_k : k -dimensional transformation matrix

```

1: function DROP( $X, B, C_d$ ):
2:   Initialize:  $i = 0; k_0 = \infty$   $\triangleright$  iteration and current basis size
3:   do
4:      $i++$ , CLOCK.RESTART
5:      $X_i = \text{SAMPLE}(X, \text{SAMPLE-SCHEDULE}(i))$   $\triangleright$  § 4.2
6:      $T_{k_i} = \text{COMPUTE-TRANSFORM}(X, X_i, B)$   $\triangleright$  § 4.3
7:      $r_i = \text{CLOCK.ELAPSED}$   $\triangleright R = \sum_i r_i$ 
8:      $\hat{k}_{i+1}, \hat{r}_{i+1} = \text{ESTIMATE}(k_i, r_i)$   $\triangleright$  § 4.4
9:     while OPTIMIZE( $C_d, k_i, r_i, \hat{k}_{i+1}, \hat{r}_{i+1}$ )  $\triangleright$  § 4.5
10:  return  $T_{k_i}$ 

```

DROP draws a data sample and performs PCA over this sample. Additionally, DROP makes use of a novel means of reusing work across iterations (§4.7).

Step 2: Transform Evaluation (§4.3, Alg 1 L6, Fig 2B)

Given the PCA result from Step 1, DROP evaluates its quality by identifying the size of the smallest metric-preserving transformation that can be extracted.

Step 3: Progress Estimation (§4.4, Alg 1 L8, Fig 2C)

Given the size of the metric-preserving transform and the computation time required to obtain this transform, DROP estimates the size and computation time of running an additional DROP iteration.

Step 4: Cost-Based Optimization (§4.5, Alg 1 L9, Fig 2D)

Given this and the estimated future iteration's transformation sizes and computation times, DROP optimizes over the end-to-end DR and downstream task runtime to determine if it should terminate.

4.2 Progressive Sampling

DROP repeatedly chooses a subset of data and computes PCA on the subsample (via one of several methods described in Section 4.6). By default, we consider a simple uniform sampling strategy where at each iteration, DROP samples a fixed percentage of the data. Exploring data-dependent and weighted sampling schemes that are dependent on the current basis is an exciting area for future work.

4.3 Evaluating Transformations

Given a transformation obtained by running PCA over a sample, DROP must accurately and efficiently evaluate the

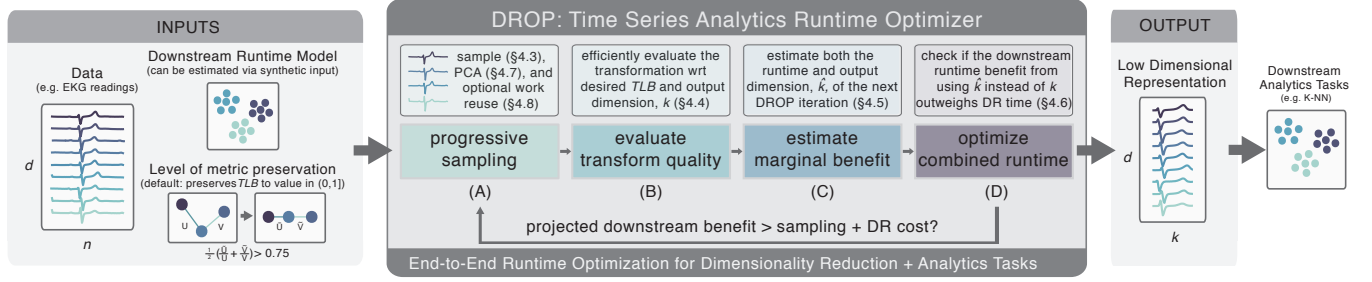


Figure 4: High-level DROP architecture depicting DROP's inputs, outputs, and core components.

performance of this transformation with respect to a metric of interest **over the entire dataset, not just the data sample**.

We define the performance of a transformation computed over a sample as the size of the lowest dimensional TLB -preserving transform that can be extracted. There are two challenges in evaluating this performance, which DROP overcomes. First, the size of the lowest dimensional transformation that achieves TLB constraints is rarely known a priori. Second, brute-force TLB computation would dominate the runtime of computing PCA over a sample.

4.3.1 Computing the Lowest Dimensional Transformation. DROP first computes a full, n -dimensional basis (i.e., of dimension equal to the input dimension) via PCA over the data sample. To reduce dimensionality, DROP must determine if a smaller dimensional TLB -preserving transformation can be computed over this sample, and return the smallest such transform. Ideally, the dimension of the best (smallest) transformation would be known, but in practice, this information is rarely known a priori. Therefore, DROP uses the TLB constraint to automatically identify the size of the returned transformation. A naïve strategy would evaluate the TLB for every combination of the n basis vectors for every transformation size, requiring $O(2^n)$ evaluations. Instead, DROP exploits two key properties of PCA to avoid this.

First, PCA via SVD produces an orthogonal linear transformation where the first principal component explains the most variance in the dataset, the second explains the second most—subject to being orthogonal to the first—and so on. Therefore, once DROP has computed the transformation matrix for dimension n , DROP obtains the transformations for all dimensions k less than n truncating the matrix to dimension $n \times k$.

Second, with respect to TLB preservation, the more principal components that are retained, the better the lower-dimensional representation in terms of TLB . This is because orthogonal transformations such as PCA preserve inner products. Therefore, a full PCA (where no dimensions are omitted) perfectly preserves \mathcal{L}_2 -distance between data points. As the \mathcal{L}_2 -distance is a sum of squared (positive) terms, the

more principal components that are retained, the better the representation preserves \mathcal{L}_2 -distance.

Using the first property (i.e., PCA's ordering), DROP obtains all low-dimensional transformations for the sample from the n -dimensional basis. Using the second property (i.e., of monotonicity of principal components), DROP then runs binary search over these transformations to find and return the lowest-dimensional basis that attains B (i.e., COMPUTE-TRANSFORM, line 1 of Algorithm 2). If a target B cannot be realized with this sample, DROP omits all further optimization steps in this iteration and continues the next iteration by drawing a larger sample.

Computing the full n -dimensional basis at every step may be wasteful. To avoid this need, DROP's exploits information from previous iterations: if DROP has previously found a candidate TLB -preserving basis of size $n' < n$ in prior iterations, then DROP only computes n' components at the start of the next iteration. **This is because similar to a hold-out or validation set, TLB evaluation is representative of the entire dataset, not just the current sample (see Alg. 2 L5). Thus, sampling additional training datapoints enables DROP to better learn global data structure and perform at least as well as over a smaller sample.** This reduces the space of lower dimensions to consider, and allows for more efficient PCA computation for future iterations, as advanced PCA routines can exploit the n' -th eigengap to converge faster (§7).

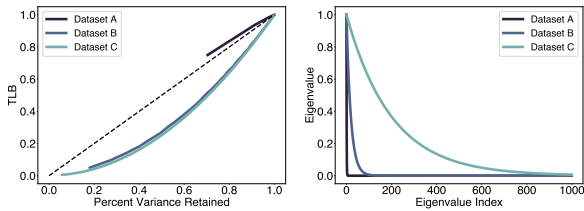
4.3.2 TLB Computation. Given a transformation, DROP must efficiently determine if the basis preserves the desired TLB . Computing pairwise TLB for all data points requires $O(d^2n)$ time, which dominates the runtime of computing PCA on a sample. However, as the TLB is an average of random variables bounded from 0 to 1, DROP can adapt techniques from online aggregation and approximate query processing [35, 58], using statistical sampling and confidence intervals to compute the TLB to arbitrary confidences.

Given a transformation, DROP iteratively refines an estimate of its TLB (function EVALUATE- TLB in Algorithm 2, line 11) by **incrementally sampling an increasing number of** pairs from the input data (Algorithm 2, line 15), transforming

each pair into the new basis, then measuring the distortion of \mathcal{L}_2 distance between the pairs, providing a *TLB* estimate to confidence level c (Algorithm 2, line 19). If the confidence interval's lower bound is greater than the target *TLB*, the basis is a sufficiently good fit; if its the upper bound is less than the target *TLB*, the basis is not a sufficiently good fit. If the confidence interval contains the target *TLB*, **DROP is unable to conclude whether or not the target *TLB* is achieved. Thus, DROP automatically samples additional pairs to refine its estimate; in practice, and especially for our initial target time series datasets, DROP rarely uses more than 500 pairs on average in its *TLB* estimates (often using far fewer).**

To estimate the *TLB* to confidence c , DROP uses the Central Limit Theorem (similar to online aggregation [35]): computing the standard deviation of a set of sampled pairs' *TLB* measures and applying a confidence interval to the sample according to the c . For data with low variance, DROP evaluates a candidate basis with few samples from the dataset **as the confidence intervals shrink rapidly.**

The techniques in this section are presented in the context of *TLB*, but can be applied to any downstream task and metric for which we can compute confidence intervals and are monotonic in number of principal components retained. **For instance, DROP can operate while using all of its optimizations when using any L^p -norm. Euclidean similarity search** is simply one such domain that is a good fit for PCA: when performing DR via PCA, as we increase the number of principal components, a clear positive correlation exists between the percent of variance explained and the *TLB* regardless of data spectrum. We demonstrate this correlation in the experiment below, where we generate three synthetic datasets with predefined spectrum (right), representing varying levels of structure present in real-world datasets. The positive correlation is evident (left) despite the fact that the two do not directly correspond ($x = y$ provided as reference). This holds true for all of the evaluated real world datasets.



For alternative preservation metrics, we can utilize closed-form confidence intervals [35, 56, 71], or bootstrap-based methods [23, 45], which incur higher overhead but can be more generally applied.

Algorithm 2 Basis Evaluation and Search

Input: X : sampled data matrix B : target metric preservation level; default *TLB* = 0.98

```

1: function COMPUTE-TRANSFORM( $X, X_i B$ ):
2:   PCA.FIT( $X_i$ )                                ▶ fit PCA on the sample
3:   Initialize: high =  $k_{i-1}$ ; low = 0;  $k_i = \frac{1}{2}(\text{low} + \text{high})$ ;  $B_i = 0$ 
4:   while (low  $\neq$  high) do
5:      $T_{k_i}, B_i = \text{EVALUATE-TLB}(X, B, k_i)$ 
6:     if  $B_i \leq B$  then low =  $k_i + 1$ 
7:     else high =  $k_i$ 
8:      $k_i = \frac{1}{2}(\text{low} + \text{high})$ 
9:    $T_{k_i}$  = cached  $k_i$ -dimensional PCA transform
10: return  $T_{k_i}$ 

11: function EVALUATE-TLB( $X, B, k$ ):
12:   numPairs =  $\frac{1}{2}d(d-1)$ 
13:    $p = 100$                                 ▶ number of pairs to check metric preservation
14:   while ( $p < \text{numPairs}$ ) do
15:      $B_i, B_{lo}, B_{hi} = \text{TLB}(X, p, k)$ 
16:     if ( $B_{lo} > B$  or  $B_{hi} < B$ ) then break
17:     else pairs  $\times= 2$ 
18: return  $B_i$ 

19: function TLB( $X, p, k$ ):
20:   return mean and 95%-CI of the TLB after transforming
       $p$   $d$ -dimensional pairs of points from  $X$  to dimension  $k$ . The
      highest transformation computed thus far is cached to avoid
      recomputation of the transformation matrix.
```

4.4 Progress Estimation

Given a low dimensional *TLB*-achieving transformation from the evaluation step, DROP must identify the quality (dimensionality) and cost (runtime) of the transformation that would be obtained from an additional DROP iteration.

Recall that DROP seeks to minimize objective function $R + C_d(k)$ such that $\text{TLB}(XT_k) \geq B$, with R denoting DROP's total runtime, T_k the k -dimensional *TLB*-preserving transformation of data X returned by DROP, and $C_d(k)$ the workload cost function. Therefore, given a k_i -dimensional transformation T_{k_i} returned by the evaluation step of DROP's i^{th} iteration, DROP can compute the value of this objective function by substituting its elapsed runtime for R and T_{k_i} for T_k . We denote the value of the objective at the end of iteration i as obj_i . To decide whether to continue iterating to find an improved transformation, DROP must be able to estimate the objective function value of future iterations.

In Section 4.5 we show that DROP requires obj_{i+1} to minimize this objective function. To estimate obj_{i+1} , DROP must estimate the runtime required for iteration $i + 1$ (which we denote as r_{i+1} , where $R = \sum_i r_i$ after i iterations) and the

dimensionality of the *TLB*-preserving transformation produced by iteration $i + 1$, k_{i+1} . Because DROP cannot directly measure r_{i+1} or k_{i+1} without performing iteration $i + 1$, DROP performs online progress estimation to estimate these quantities. Specifically, DROP performs online parametric fitting to compute future values based on prior values for r_i and k_i in line 8 of Algorithm 1. By default, given a sample of size m_i in iteration i , DROP performs linear extrapolation to estimate k_{i+1} and r_{i+1} . The estimate of r_{i+1} , for instance, is:

$$\hat{r}_{i+1} = r_i + \frac{r_i - r_{i-1}}{m_i - m_{i-1}}(m_{i+1} - m_i)$$

DROP's use of a basic first-order approximation is motivated by the fact that when adding a small number of data samples each iteration, both runtime and resulting lower dimension do not change drastically (i.e., see Fig. 3 after a feasible point is achieved). While linear extrapolation acts as a proof-of-concept for progress estimation, the architecture can incorporate more sophisticated functions as needed (§7).

4.5 Cost-Based Optimization

Given the results of the progress estimation step, DROP must determine if continued PCA on additional samples will be beneficial to overall runtime, or if it is better to terminate.

Given predictions of the next iteration's runtime (\hat{r}_{i+1}) and dimensionality (\hat{k}_{i+1}), DROP uses a greedy heuristic in estimating the optimal objective-minimizing stopping point. Concretely, if the objective function estimate for the next iteration is greater than its current objective function value ($obj_i < \widehat{obj}_{i+1}$), then DROP will terminate. If DROP's runtime is convex in the number of iterations, it is straightforward to prove that this condition is in fact the optimal stopping criterion (i.e., via convexity of composition of convex functions). This stopping criterion leads to a simple check at each DROP iteration that is used by OPTIMIZE in Algorithm 1 line 9:

$$\begin{aligned} obj_i &< \widehat{obj}_{i+1} \\ C_d(k_i) + \sum_{j=0}^i r_j &< C_d(\hat{k}_{i+1}) + \sum_{j=0}^i r_j + \hat{r}_{i+1} \\ C_d(k_i) - C_d(\hat{k}_{i+1}) &< \hat{r}_{i+1} \end{aligned} \quad (2)$$

DROP terminates when the projected time of the next iteration exceeds the estimated downstream runtime benefit.

While we empirically observed the runtime to be convex, this does not hold true in the general case as the rate of decrease in dimension (k_i) is data dependent. Should k_i plateau before continued decrease, DROP will terminate prematurely. We encounter this scenario during DROP's first iterations if sufficient data to meet the *TLB* threshold at a dimension lower than n had not been sampled (SmallKitchenAppliances

in Fig. 3). To combat this challenge, optimization is only enabled once a feasible point is attained as we prioritize accuracy over runtime (i.e., 0.3 for SmallKitchenAppliances in Fig. 3). We show the implications of this decision in DROP in Section 5.3.

4.6 Choice of PCA Subroutine

At each iteration, DROP uses PCA as its means of DR. The most straightforward means of implementing this PCA step is to compute a full SVD over the data (§2.1). There are many suitable libraries for this task—many of which are highly optimized—and therefore this strategy is pragmatic and easy to implement. However, this approach is computationally inefficient compared to other DR techniques (§2).

In our DROP implementation, we compute PCA via a randomized SVD algorithm by Halko, Martinsson, and Tropp (SVD-Halko) that calculates an approximate rank- k factorization (truncated SVD) of a data matrix [32]. While additional advanced methods for efficient PCA exist (§7), we found that not only is SVD-Halko asymptotically of the same running time as techniques used in practice (such as probabilistic PCA used in a recent SIGMOD 2015 paper on scalable PCA [24]), it is straightforward to implement, can take advantage of optimized linear algebra libraries, and does not require tuning for hyperparameters such as batch size, learning rate, or convergence criteria. SVD-Halko is not as efficient as other techniques with respect to communication complexity, as probabilistic PCA used in [24], or convergence rate, as recent work in accelerated, momentum-based PCA [19]. However, these techniques can be easily substituted for SVD-Halko in DROP's architecture.

4.7 Work Reuse

A natural question arises due to DROP's iterative architecture: can we combine the information from each sample's transformation without computing PCA over the union of the sampled data points? While stochastic methods for PCA enable such work reuse across samples as they iteratively refine a single transformation matrix, other methods do not. We propose an algorithm that allows reuse of previous work when utilizing arbitrary PCA routines with DROP.

DROP uses two key insights in order to enable this work reuse. First, given two transformation matrices produced via PCA, T_1 and T_2 , the horizontal concatenation of these matrices $H = [T_1|T_2]$ is a transformation into the union of their range spaces. Second, for datasets that have rapid drop off in spectrum, the principal components returned from running PCA on repeated data samples will generally concentrate to the true top principal components. Thus, work reuse proceeds via two step concatenate-distill approach: DROP first maintains a transformation history consisting of

the horizontal concatenation of all PCA transformations to this point, and then computes the SVD of this history matrix and returns the first k columns as the transformation matrix.

Although this routine requires an SVD computation, computational overhead is not dependent on the raw dataset size, but on the size of the history matrix, H . This size is proportional to the original dimensionality n and size of lower dimensional transformations, which are in turn proportional to the data's intrinsic dimensionality and the TLB constraint. As preserving *all history* can be expensive in practice, DROP periodically shrinks the history matrix using DR via PCA. We validate the benefit of using work reuse—up to 15% on real-world data—in Section 5.

5 EXPERIMENTAL EVALUATION

We now evaluate DROP's DR efficiency along three dimensions: runtime, accuracy, and extensibility. We demonstrate:

- (1) DROP outperforms PAA and FFT in end-to-end, repetitive-query workloads (§5.2).
- (2) DROP's optimizations for sampling, downstream task and work reuse contribute to performance (§5.3).
- (3) DROP's DR runtime scales with intrinsic dimensionality, independently of data size (§5.4).
- (4) DROP extends beyond our time series case study (§5.5).

5.1 Experimental Setup

Implementation We implement DROP as an in-memory, batch-oriented feature transformation dataflow operator² in Java using the multi-threaded *Matrix-Toolkits-Java (MTJ)* library [33] and *netlib-java* [4] linked against Intel MKL [5] for compute-intensive linear algebra operations. We use multi-threaded *JTransforms* [2] for FFT, and implement multi-threaded PAA from scratch. We use the Statistical Machine Intelligence and Learning Engine (SMILE) library [1] for k-NN with different index structures, and k-means.

Environment We run experiments on a server with two Intel Xeon E5-2690v4 @ 2.60Ghz CPUs, each with 14 physical and 28 virtual cores (with hyper-threading). The server contains 512GB of RAM. We report indexing/DR and downstream workload runtimes in isolation, excluding data loading and parsing time.

Datasets To showcase DROP's performance in an end-to-end setting and contributions from each optimization, we use several real world datasets. We first consider data from the UCR Time Series Classification Archive [17], the gold standard from the time series data mining community, for our indexing experiments and lesion studies. We exclude

datasets that have fewer than 1 million entries, and fewer datapoints than dimensionality, leaving 14 UCR datasets.

Further, due to the relatively small size of these time series datasets, we consider three additional datasets to showcase tangible wall-clock runtime improvements with DROP. We use the standard MNIST hand-written digits dataset [50], the FMA featurized music dataset [21], and a labeled sentiment analysis IMBD dataset [8], which also demonstrate extensibility beyond time series data.

DROP Configuration We use a runtime cost function for k-NN obtained via linear interpolation on data of varying dimension (implemented via cover trees [12], K-D trees [62], or brute force search in SMILE). To evaluate the sensitivity to cost model, we also report on the effect of operating without a cost model (i.e., sample until convergence) in Section 5.3. We set TLB constraints such that the accuracy of K-NN tasks remain unchanged before and after indexing via DR, corresponding to $B = 0.99$ for the UCR datasets. Unless otherwise specified, we use a default sampling schedule that begins with and increases by 1% of the input. It is possible to optimize (and possibly overfit) this schedule for our target time series, but we provide a conservative, more general schedule.

Baselines We report runtime, accuracy, and reduced dimension compared to FFT, PAA, PCA via SVD-Halko, and PCA via SVD. Each computes a transformation over the entire data, then performs binary search to identify the smallest dimensional basis that satisfies the target TLB .

Similarity Search/k-NN Setup While many methods for similarity search exist, as in [22], we consider k-NN in our evaluation as it is classically used and interoperates with new use cases including one-shot learning and deep metric learning [54, 61, 68]. Further, adopting k-NN (and, k-means in Section 5.5), which is not a classically supported relational operator, as our target task demonstrates that simple runtime estimation routines can be extended to time-series-specific operators. To evaluate DR performance when used with downstream indexes, we vary k-NN's multidimensional index structure: cover trees, K-D trees, or no index.

As previously noted, end-to-end performance of similarity search depends on the number of queries in the workload. DROP is optimized for the repeated-query use case. Due to the small size of the UCR datasets, we choose a 1:50 ratio of data indexed to number of query points, and vary this index-query ratio in later microbenchmarks and experiments. We also provide a simple cost model for assessing the break-even point that balances the cost of a given DR technique against its indexing benefits to each query point.

²<https://github.com/stanford-futuredata/DROP>

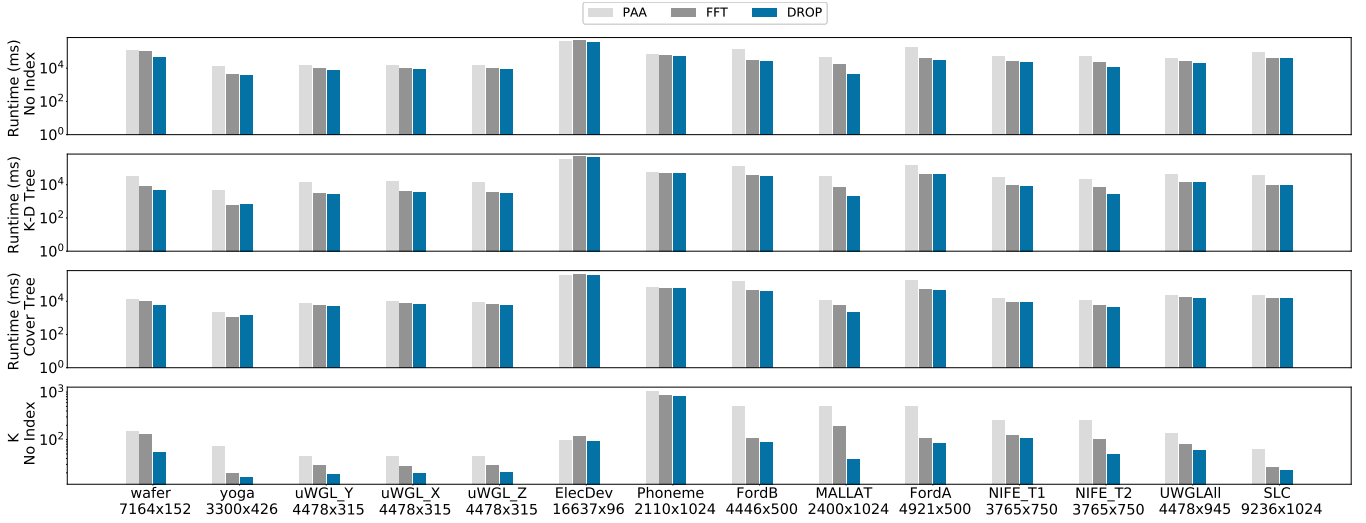


Figure 5: End-to-End DR and k-NN runtime (top three) and returned lower dimension (bottom) over the largest UCR datasets for three different indexing routines. DROP consistently returns lower dimensional bases than conventional alternatives (FFT, PAA), and is on average faster than PAA and FFT.

5.2 DROP Performance

We evaluate DROP’s performance compared to PAA and FFT using the time series case study.

k-NN Performance We summarize DROP’s results on an end-to-end 1-Nearest Neighbor classification in Figure 5. We display the end-to-end runtime of DROP, PAA, and FFT for each of the considered index structures: no index, K-D trees, cover trees. We display the size of the returned dimension for the no indexing scenario, as the other two scenarios return near **identical values. This occurs as many of the datasets used in this experiment are small and possess low intrinsic dimensionality; DROP’s cost model thus determines to quickly identify this dimensionality prior to termination.** We do not display k-NN accuracy as all techniques meet the TLB constraint, and achieve the same accuracy within 1%.

On average, DROP returns transformations that are $2.3\times$ and $1.4\times$ smaller than PAA and FFT, respectively, translating to significantly smaller k-NN query time. As a result, end-to-end runtime with DROP is on average **$2.2\times$ and $1.4\times$ (up to $10\times$ and $3.9\times$) faster than PAA and FFT, respectively**, when using brute force linear search, **$2.3\times$ and $1.2\times$ (up to $16\times$ and $3.6\times$) faster when using K-D trees**, and **$1.9\times$ and $1.2\times$ (up to $5.8\times$ and $2.6\times$) faster when using cover trees**. We demonstrate in our lesion study in Section 5.3 that DROP also outperforms our baseline PCA via SVD implementation, as well as our SVD-Halko implementation.

When evaluating Figure 5, it becomes clear that DROP’s runtime improvement is data dependent for both smaller datasets, and for datasets that do not possess a low intrinsic

dimension (such as Phoneme, elaborated on in Section 5.3). Thus, in the end of the evaluation section, we provide guidelines on how to determine if DROP is a good fit for a dataset.

Varying Index-Query Ratio

DROP is optimized for scenarios with highly structured data and a low index-query ratio, as in many streaming and/or high-volume data use cases. That is, if there are many more data points queried than used for training/constructing an index, DROP will outperform alternatives. A natural question that arises is in which concrete scenarios is it beneficial to use DROP, and in which would a lower quality but faster reduction suffice. Domain experts are typically aware of the scale of their query workloads. However, lacking this knowledge, we provide a heuristic to answer this question given rough runtime and cardinality estimates of the downstream task at hand and the choice of alternative DR technique.

Let x_d and x_a be the per-query runtime of running a downstream task with the output of DROP and a given alternative method, respectively. Let r_d and r_a denote the amortized per-datapoint runtime of DROP and the alternative method, respectively. Let n_i and n_q the number of indexed and queried points. DROP is faster when $n_q x_d + n_i r_d < n_q x_a + n_i r_a$.

To verify, we obtained estimates of the above and compared DROP against FFT and PAA in lower-query-volume scenarios when running k-NN using cover trees, and display the results in Figure 6. We first found that in the 1:1 index-query ratio setting, DROP should be slower than PAA and FFT, as observed. However, as we decrease the ratio, DROP becomes faster, with a break-even point of slightly lower than 1:3. We show that DROP does indeed outperform PAA

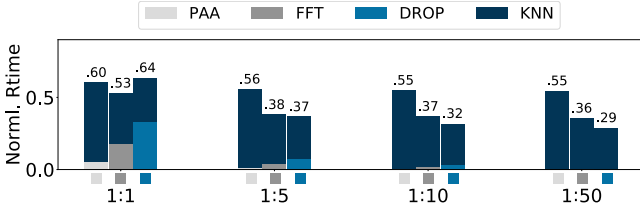


Figure 6: Effect of decreasing the index-query ratio. As an index is queried more frequently, DROP’s relative runtime benefit increases.

and FFT in the 1:5 index-query ratio case, where it is on average $1.51\times$ faster than PAA and $1.03\times$ faster than FFT. As the ratio decreases to 1:50, DROP is $1.24\times$ faster than FFT and $1.9\times$ faster than PAA.

Time Series Similarity Search Extensions Given the breadth of research in time series indexing, a natural question to ask is how DROP, a general operator for PCA, compares to state-of-the-art time series indexes. As a preliminary evaluation, we consider iSAX2+ [14], a state-of-the-art indexing tool in a 1:1 index-query ratio setting, using a publicly available Java implementation [3]. While these indexing techniques also optimize for the low index-query ratio setting, we find index construction to be a large bottleneck in these workloads. For iSax2+, index construction is on average $143\times$ (up to $389\times$) slower than DR via DROP, but is on average only $11.3\times$ faster than k-NN on the reduced space. However, given high enough query workload, these specialized techniques will surpass DROP.

We also verify that DROP is able to perform well when using downstream similarity search tasks relying on alternative distance metrics, namely, Dynamic Time Warping (DTW)—a commonly used distance measure in the literature [64]. As proof-of-concept, we implement a 1-NN task using DTW with a 1:1 index-query ratio, and find that even with this high ratio, DROP provides on average $1.2\times$ and $1.3\times$ runtime improvement over PAA and FFT, respectively. As DTW is known to be incredibly slow [43], it is unsurprising that DROP provides large runtime benefits for tasks using DTW without additional pruning—in terms of absolute runtime, DROP saves 2.8 minutes on the FordA dataset compared to PAA, and 2.2 minutes on the wafer dataset compared to FFT.

Finally, as the considered time series are fairly short, we perform the same experiment over a standard gaussian random walk synthetic dataset [46, 53] consisting of 50,000 time series of dimension 10,000. Each time series is generated by, for each time step, generating a random value distributed via standard normal distribution, and adding it to the running sum of all previous time steps. We find that DROP takes

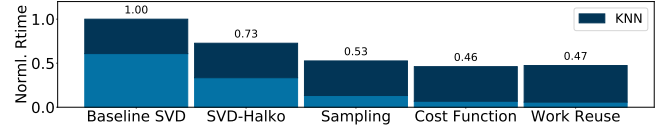


Figure 7: Average result of lesion study over the UCR datasets.

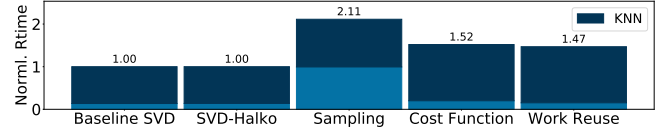


Figure 8: Lesion study of the UCR phoneme, a dataset with high intrinsic dimensionality, meaning sampling to convergence is orders of magnitude slower than a batch SVD. DROP’s cost function enables it to terminate in advance, returning a higher dimensional basis to minimize reduce overall compute.

4150ms to complete, whereas PAA and FFT take 5523ms ($1.3\times$ faster) and 15329ms ($3.7\times$ faster), respectively.

5.3 Lesion Study

We perform a factor analysis of the incremental runtime and dimensionality contributions of each of DROP’s components compared to baseline SVD methods. We only display the results of k-NN with cover trees; the results hold for the other indexes. We use a 1:1 index-query ratio **with data inflated by $5\times$** to better highlight the effects of each contribution to the DR routine, and display average results over the UCR datasets in Figure 7, excluding Phoneme.

Figure 7 first demonstrates the boost from using SVD-Halko over a naïve implementation of PCA via SVD, which comes from not computing the full transformation a priori, incrementally binary searching as needed. It then shows the runtime boost obtained from running on samples until convergence, where DROP samples and terminates after the returned lower dimension from each iteration plateaus. This represents the naïve sampling-until-convergence approach described in Section 3 that DROP defaults to sans user-specified cost model. We finally introduce cost based optimization and work reuse. Each of these optimizations improves runtime, with the exception of work reuse, which has a negligible impact on average but disproportionately impacts certain datasets.

On average, DROP is $2.1\times$ faster (up to $41\times$) than PCA via SVD, and $1.6\times$ faster than SVD-Halko (up to $3.3\times$). DROP with cost-based optimization is faster than sampling to convergence by $1.2\times$ on average, but this default strategy is still $1.4\times$ faster than SVD-Halko on average.

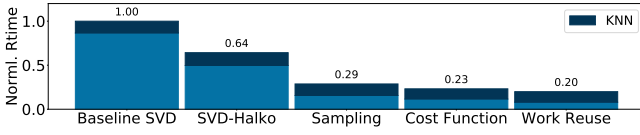


Figure 9: Lesion study over the UCR yoga dataset. Work reuse provides a 15% runtime improvement.

Work reuse typically only slightly affects end-to-end runtime as it is useful primarily when a large number of DROP iterations are required (i.e., when dataset spectrum is not well-behaved). We also observe this behavior on certain small datasets with moderate intrinsic dimensionality, such as the yoga dataset in Figure 9. Work reuse provides a 15% improvement in addition to cost based optimization.

DROP’s sampling operates on the premise that the dataset has data-point-level redundancy. However, datasets without this structure are more difficult to reduce the dimensionality of. Phoneme is an example of one such dataset (Figure 8). In this setting, DROP **incrementally examines a large proportion of data before enabling cost-based optimization**, resulting in a performance penalty. We discuss extensions to DROP to mitigate this in the extended version of this manuscript.

5.4 Scalability

Data generated by automated processes such as time series often grows much faster in size than intrinsic dimensionality. DROP can exploit this intrinsic dimensionality to compute PCA faster than traditional methods as it only processes an *entire* dataset if a low intrinsic dimensionality does not exist.

To demonstrate this, we fix intrinsic dimensionality of a synthetic dataset generated via random projections to 8 as we grow the number of datapoints from 5K to 135K. Hence, the sample size an algorithm requires to uncover this dataset’s intrinsic dimensionality is constant regardless of the full dataset size. In this experiment, we enable DROP’s fixed-size sampling schedule set to increase by 500 datapoints at each iteration. As Figure 10 shows, DROP is able to find a 8-dimensional basis that preserves *TLB* to 0.99 within **145ms** for dataset sizes up to 135K data points, and is **12×** faster than binary search with SVD-Halko. Runtime is near constant as dataset size increases, with small overhead due to sampling from larger datasets. This near-constant runtime contrasts with PCA via SVD and SVD-Halko as they do not exploit the intrinsic dimensionality of the dataset and process all provided points, further illustrating the scalability and utility of sample-based DR.

5.5 Beyond the Time Series Case Study

We consider generalizability beyond our initial case study along two axes: data domain and downstream workload.

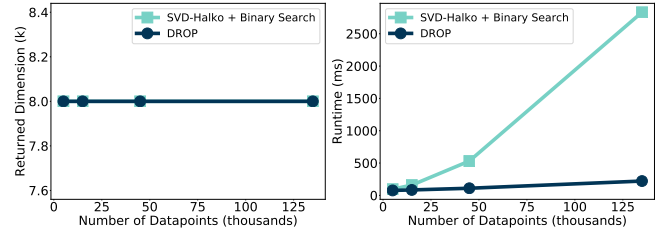


Figure 10: Effect of dataset size on time and output dimension (k), with constant intrinsic data dimensionality of 8. DROP runtime with a fixed schedule remains near constant.

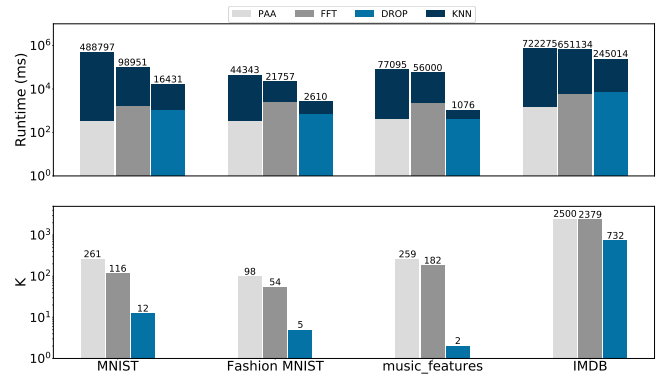


Figure 11: End-to-End k-NN runtime (top) and returned dimension k (bottom) over the entire MNIST dataset and the FMA featurized music dataset.

These preliminary results show promise in extension to additional domains and target tasks.

Data Domain. We examine classification/similarity search workloads across image classification, music analysis, and natural language processing. To better show the trade-off in DR and downstream workload, we repeat the k-NN retrieval experiments with a 1:1 index-query ratio. We use the MNIST hand-written digit image dataset containing 70,000 images of dimension 784 (obtained by flattening each 28×28 -dimensional image into a single vector [50], combining both the provided training and testing datasets); FMA’s featurized music dataset, providing 518 features across 106,574 music tracks; a bag-of-words representation of an IMDB sentiment analysis dataset across 25,000 movies with 5000 features [8]; **Fashion MNIST’s 70,000 images of dimension 784** [70]. We present our results in Figure 11. As the given datasets are larger than the ones presented in [17], DROP’s ability to find a *TLB*-preserving low dimensional basis is more valuable as this more directly translates to significant reduction in end-to-end runtime—up to a **7.6 minute wall-clock improvement** in MNIST, **42 second improvement** in Fashion MNIST,

1.2 minute improvement in music features, and 8 minute improvement in IMDB compared to PAA. These wall-clock runtime effects will only be amplified as the index-query ratio decreases, to be more typical of the repeated-query setting. For instance, when we decrease the ratio to 1:5 on the music features dataset, DROP provides a 6.1 and 4.5 minute improvement compared to PAA and FFT, respectively.

Downstream Workload. To demonstrate the generalizability of both DROP's pipeline as well as black-box runtime cost-model estimation routines, we extend our pipeline to perform a k-means task over the MNIST digits dataset. We fit a new downstream workload runtime model as we did with k-NN, and operate under a 1:1 index-query ratio. In this workload, DROP terminates in 1488ms, which is $16.5\times$ faster than PAA and $6.5\times$ faster than FFT.

6 FUTURE WORK

6.1 Data-Aware Sampling

DROP's efficiency is determined by the dataset's spectrum; MALLAT, with the sharpest drop-off, performs extremely well, and Phoneme, with a near uniform distribution, does not. Datasets such as Phoneme perform poorly under the default configuration as we enable cost-based optimization after reaching a feasible point. Thus, DROP spends a disproportionate time sampling (Fig. 8). Extending DROP to more efficiently determine if a dataset is amenable to aggressive sampling is an exciting area of future work. For instance, recent theoretical results that use sampling to estimate spectrum, even when the number of samples are small in comparison to the input dimensionality [47], can be run alongside DROP with minimal alteration.

6.2 Streaming Execution

Given a stationary input distribution, users can extract fixed-length sliding windows from the source and apply DROP's transformation over these segments as they arrive. Should the data distribution not be stationary over time, DROP can be periodically retrained in one of two ways. First, DROP can use of the wide body of work in changepoint or feature drift detection [31, 44] to determine when to retrain. Alternatively, DROP can maintain a reservoir sample of incoming data [66], tuned to the specific application, and retrain if the metric of interest no longer satisfies user-specified constraints. Due to DROP's default termination condition, cost-based optimization must be disabled until the metric constraint is achieved to prevent early termination.

7 RELATED WORK

Dimensionality Reduction DR is a classic operation [18, 28, 51, 65] that is well studied in the database [7, 15, 42, 60],

data mining [41, 52], statistics and machine learning [20, 63], and theoretical CS [30, 36] communities.

Recent breakthroughs in the theoretical statistics community provided new algorithms for PCA that promise substantial scalability improvements without compromising result quality [19, 20, 32]. Foremost among these techniques are advanced stochastic methods [19, 63], and techniques for randomized SVD [32]. While we default to the latter for use by DROP's PCA operator, DROP's modular architecture makes it simple to use any method in its place, including recent systems advances in scalable PCA [24]. To the best of our knowledge, advanced methods for PCA have not been empirically compared head-to-head with conventional dimensionality reduction approaches such as Piecewise Approximate Averaging [41], especially on real datasets. In addition, DROP combines these methods with row-level sampling to provide benefits similar to using stochastic methods for PCA, regardless of the chosen PCA subroutine.

Approximate Query Processing Inspired by approximate query processing engines [58] as in online aggregation [35], DROP performs progressive sampling. In contrast with more general data dimensionality estimation methods [13], DROP optimizes for *TLB*. As we illustrated in Section 5, this strategy confers substantial runtime improvements. While DROP performs simple uniform sampling, the literature contains a wealth of techniques for various biased sampling techniques [9, 16]. Finally, DROP performs online progress estimation to minimize the end-to-end analytics cost function. This is analogous to query progress estimation [55] and performance prediction [57] in database and data warehouse settings and has been exploited in approximate query processing engines such as BlinkDB [6].

Scalable Workload-Aware, Complex Analytics DROP is an operator for analytics dataflow pipelines. Thus, DROP is as an extension of recent results on integrating complex analytics function including model training [27, 37, 48] and data exploration [67, 69, 72] operators into analytics engines.

8 CONCLUSION

Advanced data analytics techniques must scale to rising data volumes. DR techniques offer a powerful toolkit when processing these datasets, with PCA frequently outperforming popular techniques in exchange for high computational cost. In response, we propose DROP, a new dimensionality reduction optimizer. DROP combines progressive sampling, progress estimation, and online aggregation to identify high quality low dimensional bases via PCA without processing the entire dataset by balancing the runtime of downstream tasks and achieved dimensionality. Thus, DROP provides a first step in bridging the gap between quality and efficiency in end-to-end DR for downstream analytics.

REFERENCES

- [1] 2008. SMILE. (2008). <http://haifengl.github.io/smile/>.
- [2] 2015. JTransforms. (2015). <https://sites.google.com/site/piotrwendykier/software/jtransforms>.
- [3] 2017. DPiSAX. (2017). <http://djameledine-yagoubi.info/projects/DPiSAX/>.
- [4] 2017. netlib-java. (2017). <https://github.com/fommil/netlib-java>.
- [5] 2018. Intel MKL. (2018). <https://software.intel.com/en-us/mkl>.
- [6] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 29–42.
- [7] Charu C Aggarwal. 2001. On the effects of dimensionality reduction on high dimensional similarity search. In *PODS*.
- [8] Peter T. Pham Dan Huang Andrew Y. Ng Andrew L. Maas, Raymond E. Daly and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL 2011*.
- [9] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 539–550.
- [10] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*. ACM.
- [11] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. 2017. Prioritizing Attention in Fast Data: Principles and Promise.
- [12] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 97–104.
- [13] Francesco Camastra. 2003. Data dimensionality estimation methods: a survey. *Pattern recognition* 36, 12 (2003), 2945–2954.
- [14] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.
- [15] Kaushik Chakrabarti and Sharad Mehrotra. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*.
- [16] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)* 32, 2 (2007), 9.
- [17] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. (July 2015). www.cs.ucr.edu/~eamonn/time_series_data/.
- [18] John P Cunningham and Zoubin Ghahramani. 2015. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research* 16, 1 (2015), 2859–2900.
- [19] Christopher De Sa, Bryan He, Ioannis Mitliagkas, Chris Re, and Peng Xu. 2018. Accelerated Stochastic Power Iteration. In *Artificial Intelligence and Statistics*.
- [20] Christopher De Sa, Kunle Olukotun, and Christopher Ré. 2015. Global Convergence of Stochastic Gradient Descent for Some Non-convex Matrix Problems. In *ICML*.
- [21] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. 2017. FMA: A Dataset For Music Analysis. In *18th International Society for Music Information Retrieval Conference*.
- [22] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. In *VLDB*.
- [23] Bradley Efron and Robert Tibshirani. 1997. Improvements on cross-validation: the 632+ bootstrap method. *J. Amer. Statist. Assoc.* 92, 438 (1997), 548–560.
- [24] Tarek Elgamal, Maysam Yabandeh, Ashraf Aboulmaga, Waleed Mustafa, and Mohamed Hefeeda. 2015. sPCA: Scalable Principal Component Analysis for Big Data on Distributed Platforms. In *SIGMOD*.
- [25] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 12.
- [26] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. *Fast subsequence matching in time-series databases*.
- [27] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*.
- [28] Imola K Fodor. 2002. *A survey of dimension reduction techniques*. Technical Report. Lawrence Livermore National Lab., CA (US).
- [29] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.
- [30] Navin Goyal, Santosh Vempala, and Ying Xiao. 2014. Fourier PCA and Robust Tensor Decomposition. In *STOC*.
- [31] Valery Guralnik and Jaideep Srivastava. 1999. Event detection from time series data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 33–42.
- [32] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.
- [33] Sam Halliday and Björn-Ove Heimsund. 2008. matrix-toolkits-java. (2008). <https://github.com/fommil/matrix-toolkits-java>.
- [34] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.
- [35] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *SIGMOD*.
- [36] Prateek Jain, Chi Jin, Sham M Kakade, Praneeth Netrapalli, and Aaron Sidford. 2016. Streaming PCA: Matching Matrix Bernstein and Near-Optimal Finite Sample Guarantees for Oja’s Algorithm. In *COLT*.
- [37] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. 2008. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*.
- [38] Ian T Jolliffe. 1986. Principal component analysis and factor analysis. In *Principal component analysis*. Springer, 115–128.
- [39] Yannis Katsis, Yoav Freund, and Yannis Papakonstantinou. 2015. Combining Databases and Signal Processing in Plato. In *CIDR*.
- [40] Eamonn Keogh. 2006. A decade of progress in indexing and mining large time series databases. In *VLDB*.
- [41] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information systems* 3, 3 (2001), 263–286.
- [42] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Sigmod Record* 30, 2 (2001), 151–162.
- [43] Eamonn J Keogh and Michael J Pazzani. 1999. Scaling up dynamic time warping to massive datasets. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 1–11.
- [44] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107, 500 (2012), 1590–1598.
- [45] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. 2012. The big data bootstrap. *arXiv preprint arXiv:1206.6415* (2012).
- [46] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A scalable bottom-up approach for building data series indexes. *Proceedings of the VLDB Endowment* 11, 6 (2018), 677–690.

- [47] W. Kong and G. Valiant. 2017. Spectrum Estimation from Samples. *Annals of Statistics* 45, 5 (2017), 2218–2247.
- [48] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. 2013. MLbase: A Distributed Machine-learning System.. In *CIDR*.
- [49] Hans-Peter Kriegel, Peer Kröger, and Matthias Renz. 2010. Techniques for efficiently searching in spatial, temporal, spatio-temporal, and multimedia databases. In *ICDE*. IEEE.
- [50] Yann LeCun. 1998. The MNIST database of handwritten digits. <http://yann.lecun.com/exdb/mnist/> (1998).
- [51] John A Lee and Michel Verleysen. 2007. *Nonlinear dimensionality reduction*. Springer Science & Business Media.
- [52] Jessica Lin, Eamonn Keogh, Wei Li, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery* 15, 2 (2007), 107.
- [53] Chen Luo and Anshumali Shrivastava. 2017. SSH (Sketch, Shingle, & Hash) for Indexing Massive-Scale Time Series. In *NIPS 2016 Time Series Workshop*. 38–58.
- [54] Renqiang Min, David A Stanley, Zineng Yuan, Anthony Bonner, and Zhaolei Zhang. 2009. A deep non-linear feature mapping for large-margin knn classification. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 357–366.
- [55] Chaitanya Mishra and Nick Koudas. 2007. A lightweight online framework for query progress indicators. In *ICDE*.
- [56] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge university press.
- [57] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. 2010. Estimating the progress of MapReduce pipelines. In *ICDE*.
- [58] Barzan Mozafari. 2017. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*.
- [59] Milos Nikolic, Badrish Chandramouli, and Jonathan Goldstein. 2017. Enabling Signal Processing over Data Streams. In *SIGMOD*. ACM.
- [60] KV Ravi Kanth, Divyakant Agrawal, and Ambuj Singh. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*.
- [61] Ruslan Salakhutdinov and Geoff Hinton. 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*. 412–419.
- [62] Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [63] Ohad Shamir. 2015. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *ICML*. 144–152.
- [64] Jin Shieh and Eamonn Keogh. 2008. i SAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 623–631.
- [65] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.
- [66] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.
- [67] Abdul Wasay, Xinding Wei, Niv Dayan, and Stratos Idreos. 2017. Data Canopy: Accelerating Exploratory Statistical Analysis. In *SIGMOD*.
- [68] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*. 1473–1480.
- [69] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*.
- [70] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).
- [71] Taro Yamane. 1973. *Statistics: An introductory analysis*. (1973).
- [72] E. Zraggen, A. Galakatos, A. Crotty, J. D. Fekete, and T. Kraska. 2016. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Transactions on Visualization and Computer Graphics* (2016).
- [73] Yunyue Zhu and Dennis Shasha. 2003. Warping indexes with envelope transforms for query by humming. In *SIGMOD*.