DROP: A Workload-Aware Optimizer for Dimensionality Reduction

Sahaana Suri, Peter Bailis
Stanford University

ABSTRACT
Dimensionality reduction (DR) is critical in scaling machine learning pipelines: by reducing input dimensionality in exchange for a preprocessing overhead, DR enables faster end-to-end runtime. Principal component analysis (PCA) is a DR standard, but can be computationally expensive: classically \(O(dn^2 + n^3)\) for an \(n\)-dimensional dataset of \(d\) points. Theoretical work has optimized PCA via iterative, sample-based stochastic methods. However, these methods execute for a fixed number of iterations or to convergence, sampling too many or too few datapoints for end-to-end runtime improvements. We show how accounting for downstream analytics operations during DR via PCA allows stochastic methods to efficiently terminate after processing small (e.g., 1%) samples of data. Leveraging this, we propose DROP, a DR optimizer that enables speedups of up to \(5\times\) over Singular-Value-Decomposition (SVD)-based PCA, and \(16\times\) over conventional DR methods in end-to-end nearest neighbor workloads.

ACM Reference Format:
Sahaana Suri, Peter Bailis. 2019. DROP: A Workload-Aware Optimizer for Dimensionality Reduction. In International Workshop on Data Management for End-to-End Machine Learning (DEEM’30), June 30, 2019, Amsterdam, Netherlands. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3329486.3329490

1 INTRODUCTION
Rapid growth in high-dimensional data from automated sources poses a scalability challenge for machine learning pipelines [10, 34]. Practitioners turn to dimensionality reduction (DR) techniques to alleviate this challenge [13, 21, 35, 37]. DR methods transform an \(n\)-dimensional dataset to a lower \(k\)-dimensional representation while preserving salient dataset features. This allows downstream analytics routines to run in time that scales with \(k\) while preserving downstream task accuracy. Thus, in exchange for a preprocessing overhead, DR techniques can decrease end-to-end workload runtime.

To attain a target downstream task accuracy with as low a transformation dimension \(k\) as possible, Principal Component Analysis (PCA) is often practitioners’ DR method of choice [33]. However, classic task-independent PCA implementations (i.e., via Singular Value Decomposition or SVD) scale poorly, with overheads that outweigh DR’s downstream runtime benefit. In response, practitioners may use faster DR techniques that return higher \(k\) for a given accuracy, but provide a lower end-to-end runtime. For instance, PCA is traditionally considered too computationally expensive for time series similarity search [24], with widely-cited work excluding it for more efficient, less precise methods [20].

In this work, we accelerate PCA by leveraging the fact that practitioners are willing to trade dimensionality for whole workload runtime. We build off the key insight from theoretical work has optimized PCA via iterative, sample-based stochastic methods [17, 48]: instead of processing the entire dataset at once, data samples can be used to iteratively refine an estimate of the transformation until the transformation converges. However, for end-to-end runtime optimization, sampling to convergence can be unnecessary as DR for downstream tasks such as similarity search are effective even with suboptimal, higher-dimensional transformations [20]. Taking into account this workload-dependent tolerance to error during sample-based PCA would enable us to terminate prior to convergence, and thus more quickly return an accuracy-preserving transformation. This leaves the open question of: given a dataset and workload, what sampling rate (i.e., termination condition) is required to optimize for both accuracy and end-to-end runtime?

To resolve this data- and workload-dependent trade-off, we develop DROP, a system that dynamically identifies the amount of sampling required for stochastic PCA by using downstream task information. DROP takes as input a high-dimensional dataset, a workload-dependent constraint on approximation accuracy (e.g., pairwise Euclidean distance to 5% for similarity search, see § 3.2), and an optional runtime model expressing downstream computational cost 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 preserving the input constraint.

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 increases the number of samples until termination. To identify the termination point that minimizes 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 [24, 29], 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. When running PCA on a series of larger samples, later samples will increase DR runtime, but may return lower $k$ (lower downstream runtime)—DROP must estimate how these values will change in future iterations to navigate this trade-off between end-to-end runtime and dimensionality. DROP uses the results obtained from previous iterations to fit predictive models for dimensionality and runtime of the next iteration.

Finally, given the predicted marginal benefit, DROP must optimize end-to-end runtime. While an application-agnostic approach would iterate until successive iterations yield little or no benefit, a user-provided runtime model may reveal that trading a higher $k$ for a lower DR runtime may decrease overall runtime. DROP evaluates the runtime model at each iteration to minimize the expected workload runtime.

DROP is a system that combines recent theoretical advances in DR and classic techniques from approximate query processing for end-to-end workflow optimization. In this work, we make the following contributions:

- We show the data sample required to perform accuracy-achieving PCA is often small (as little as 1%), and sampling can enable up to $91 \times$ speedup over baseline PCA.
- 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 $5 \times$ faster end-to-end execution over PCA via SVD.

2 RELATED WORK

Dimensionality Reduction DR is a well-studied operation [16, 23, 41] in the database [7, 13, 37], data mining [36, 42], statistics and machine learning [18, 31, 48] communities. In this paper, our focus is on DR via PCA. While classic PCA via SVD is inefficient, stochastic [17, 48] and randomized [27] methods provide scalable alternatives. DROP draws from both the former to tackle the challenge of how much data to sample, and the latter for its default PCA operator (though DROP’s modular architecture makes it simple to use any method in its place). Further, to the best of our knowledge, these advanced methods for PCA have not been empirically compared head-to-head with conventional DR approaches such as Piecewise Approximate Averaging [36].

Approximate Query Processing (AQP) Inspired by AQP engines [45] as in online aggregation [30], DROP performs progressive sampling. While DROP performs simple uniform sampling, the literature contains a wealth of techniques for various biased sampling techniques [9, 14]. DROP performs online progress estimation to minimize the end-to-end analytics cost function. This is analogous to query progress estimation [43] and performance prediction [44] 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 an extension of recent results on integrating complex analytics function including model training [22, 32] and data exploration [52, 53, 55] operators into analytics engines.

3 BACKGROUND AND PROBLEM

In this section, we provide background on dimensionality reduction (DR) and our problem of workload-aware DR.

3.1 Dimensionality Reduction

The goal of DR is to find a low-dimensional representation of a dataset that preserves metrics of interest, such as data point similarity [16, 23]. Formally, consider a data matrix $X \in \mathbb{R}^{d \times n}$, where each row $i$ corresponds to data point $x_i \in \mathbb{R}^d$, with $d > n$. DR computes a transformation function $T : \mathbb{R}^n \rightarrow \mathbb{R}^k$ that maps each $x_i$ to a more compact representation, 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 [50].

3.2 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. Indexes built over this data can improve the efficiency of this repeated query workload in exchange for a preprocessing overhead. DR with a multi-dimensional index structure in the reduced space is a classic way of achieving this [13, 29, 35, 39, 46].

**DR in Similarity Search.** Similarity search is a repeated-query workload performed over data types including images, documents and time series [20, 25]. When similarity is measured by Euclidean distance, our goal is to find a low-dimensional representation that approximately preserves pairwise $\ell_2$-distances between data points. We quantify this distance-preservation property using the Tightness of Lower Bounds (TLB) metric [20], which estimates the performance difference in a downstream similarity search routine (i.e., k-Nearest Neighbors) after DR without running the routine:

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

(1)

Given the large amount of research in the space, we use time series similarity search as a running case study throughout this paper. We briefly revisit a comparison of DR techniques for time series similarity search from VLDB 2008 [20] to verify that PCA can outperform conventionally used techniques (low $k$), but with a higher DR runtime cost. The authors omit PCA due to it being "untenable for large data sets."

We compare PCA via SVD to baseline techniques based on returned dimensionality and runtime with respect to TLB over the largest datasets from [20]. We use their two fastest methods as our baselines as they show the remainder exhibited "very little difference": Fast Fourier Transform (FFT) and Piecewise Aggregate Approximation (PAA). On average, PCA admits an output dimension $k$ that is $2.3\times$ (up to $3.9\times$) and $3.7\times$ (up to $26\times$) smaller than PAA and FFT for TLB $= 0.75$, and $2.9\times$ (up to $8.3\times$) and $1.8\times$ (up to $5.1\times$) smaller for TLB $= 0.99$. However, 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.

While the margin between PCA and alternatives is dataset-dependent, PCA almost always preserves TLB with a lower dimensional representation at a higher runtime cost. This runtime-dimensionality trade-off motivates our study of workload-aware DR methods.

### 3.3 Problem: Workload-Aware DR

In workload-aware DR, we perform DR to minimize workload runtime subject to downstream metric constraints. DR is a fixed cost (i.e., index construction for similarity search), while workload queries incur a marginal cost dependent on DR dimensionality (i.e., nearest neighbor query).

As input, consider a dataset $X$, target metric preservation or constraint on approximation accuracy $B$ (e.g., $TLB \geq 99$), and optional downstream runtime model as a function of dimensionality $C_d(n)$ for an $d \times n$ matrix. The metric provides insight into how DR affects downstream accuracy by characterizing workload accuracy without requiring the workload to be run (i.e., TLB for similarity search). Denoting DR runtime as $R$, we define the problem:

**Problem 3.1.** 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 $TLB(T_kX) \geq B$ with confidence $c$.

We assume $C_d(n)$ is monotonically increasing in $n$. The more DR time spent, the smaller the transformation (as in the case study), thus the lower the workload runtime. To minimize $R + C_d(k)$, we determine how much time to spend on DR (thus, what $k$ to return) to minimize overall runtime.

### 4 DROP: WORKLOAD OPTIMIZATION

In this section, we introduce DROP, a system that performs workload-aware DR via progressive sampling and online progress estimation. DROP takes as input a target dataset, metric to preserve (default, target TLB), and an optional downstream runtime model. DROP then uses sample-based PCA to identify and return a low-dimensional representation of the input that preserves the specified property while minimizing estimated workload runtime (Figure 2, Alg. 1).
4.1 DROP Algorithm

DROP operates over a series of data samples, and determines when to terminate via a four-step procedure at each iteration:

Step 1: Progressive Sampling (§4.2)
DROP draws a data sample, performs PCA over it, and uses a novel reuse mechanism across iterations (§4.7).

Step 2: Transform Evaluation (§4.3)
DROP evaluates the above by identifying the size of the smallest metric-preserving transformation that can be extracted.

Step 3: Progress Estimation (§4.4)
Given the size of the smallest metric-preserving transform and the time required to obtain this transform, DROP estimates the size and computation time of continued iteration.

Step 4: Cost-Based Optimization (§4.5)
DROP optimizes over DR and downstream task runtime to determine if it should terminate.

4.2 Progressive Sampling

Inspired by stochastic PCA methods (§2), DROP uses sampling to tackle workload-aware DR. Many real-world datasets are intrinsically low-dimensional; a small data sample is sufficient to characterize dataset behavior. To verify, we extend our case study (§3.2) by computing how many uniformly selected data samples are required to obtain a TLB-preserving transform with k equal to input dimension n. On average, a sample of under 0.64% (up to 5.5%) of the input is sufficient for TLB = 0.75, and under 4.2% (up to 38.6%) is sufficient for TLB = 0.99. If this sample rate is known a priori, we obtain up to 91× speedup over PCA via SVD.

However, this benefit is dataset-dependent, and unknown a priori. We thus turn to progressive sampling (gradually increasing the sample size) to identify how large a sample suffices. Figure 2 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 (which increases PCA runtime) provides lower k for the same TLB. However, this decrease in dimension plateaus as the number of samples increases. Thus, while progressive sampling allows DROP to tune the amount of time spent on DR, DROP must determine when the downstream value of decreased dimension is overpowered by the cost of DR—that is, whether to sample to convergence or terminate early (e.g., at 0.3 proportion of data sampled for SmallKitchenAppliances).

Concretely, DROP first repeatedly chooses a subset of data and computes a n-dimensional transformation via PCA on the subsample, and then proceeds to determine if continued sampling is beneficial to end-to-end runtime. We consider a simple uniform sampling strategy: each iteration, DROP samples a fixed percentage of the data.

4.3 Transform Evaluation

DROP must accurately and efficiently evaluate this iteration’s performance with respect to the metric of interest over the entire dataset. We define this iteration’s performance as the size of the lowest dimensional TLB-preserving transform (k_i) that it can return. There are two challenges in performance evaluation. First, the lowest TLB-achieving k_i is unknown a priori. Second, brute-force TLB computation would dominate the runtime of computing PCA over a sample. We now describe how to solve these challenges.

4.3.1 Computing the Lowest Dimensional Transformation.

Given the n-dimensional transformation from step 1, to reduce dimensionality, DROP must determine if a smaller dimensional TLB-preserving transformation can be obtained and return the smallest such transform. Ideally, the smallest k_i would be known a priori, but in practice, this is not true—thus, DROP uses the TLB constraint and two properties of PCA to automatically identify it.

First, PCA via SVD produces an orthogonal linear transformation where the principal components are returned in order of decreasing dataset variance explained. As a result, once

Algorithm 1 DROP Algorithm

```
function DROP(X, B, C_d):
    Initialize: i = 0; k_0 = \infty  // iteration and current basis size
    do
        i++;
        clock.restart;
        X_i = sample(X, sample-schedule(i))  // § 4.2
        T_k_i = compute-transform(X, X_i, B) // § 4.3
        r_i = clock.elapsed  // R = \sum r_i
        k_{i+1}, r_{i+1} = estimate(k_i, r_i)  // § 4.4
        while optimize(C_d, k_i, r_i, k_{i+1}, r_{i+1})  // § 4.5
    return \hat{T}_{k_i}
```

Input: X: data; B: target metric preservation level; C_d: cost of downstream operations
Output: T_{k_i}: k-dimensional transformation matrix

Figure 2: Reduction in dimensionality for TLB = 0.80 with progressive sampling. Dimensionality decreases until reaching a state equivalent to running PCA over the full dataset ("convergence").
DROP has computed the transformation matrix for dimension $n$, DROP obtains the transformations for all dimensions $k$ less than $n$ by truncating the matrix to $n \times k$.

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, an $n$-dimensional PCA perfectly preserves $\ell_2$-distance between data points. As $\ell_2$-distance is a sum of squared (positive) terms, the more principal components retained, the better the representation preserves $\ell_2$-distance.

Using the first property, DROP obtains all low-dimensional transformations for the sample from the $n$-dimensional basis. Using the second property, DROP runs binary search over these transformations to return the lowest-dimensional basis that attains $B$ (Alg. 2, 11). If $B$ cannot be realized with this sample, DROP omits further optimization steps and continues the next iteration by drawing a larger sample.

Additionally, computing the full $n$-dimensional basis at every iteration may be wasteful. Thus, if DROP has 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 allows for more efficient PCA computation for future iterations, as advanced PCA routines can exploit the $n'$-th eigengap to converge faster (§2).

### 4.3.2 Efficient TLB Computation

Given a transformation, DROP must determine if it preserves the desired TLB. Computing pairwise TLB for all data points requires $O(d^2 n)$ 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 use sampling and confidence intervals to compute the TLB to arbitrary confidences.

Given a transformation, DROP iteratively refines an estimate of its TLB (Alg. 2, 111) by incrementally sampling an increasing number of pairs from the input data (Alg. 2, 115), transforming each pair into the new basis, then measuring the distortion of $\ell_2$-distance between the pairs, providing a TLB estimate to confidence level $c$ (Alg. 2, 119). If the confidence interval’s lower bound is greater than the target TLB, the basis is a sufficiently good fit; if its 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 cannot determine if the target TLB is achieved. Thus, DROP automatically samples additional pairs to refine its estimate.

To estimate the TLB to confidence $c$, DROP uses the Central Limit Theorem: computing the standard deviation of a set of sampled pairs’ TLB measures and applying a confidence interval to the sample according to the $c$.

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.

### Algorithm 2 Basis Evaluation and Search

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

1: **function** compute-transform($X$, $X_kB$):
2:     $pca.fit(X)$  // 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 ! = high) **do**
5:         $T_k_i$, $B_i$ = evaluate-tlb($X$, $B_i$)
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($X$, $B_i$, $k$):
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:         $B_i$, $B_{hi}$ = TLB($X$, $p$, $k$)
16:         **if** ($B_{hi} > B$ 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

Recall that the goal of workload-aware DR is to minimize $R + G_d(k)$ such that $TLB(XT_k) \geq B$, with $R$ denoting total DR (i.e., DROP’s) runtime, $T_k$ the $k$-dimensional TLB-preserving transformation of data $X$ returned by DROP, and $G_d(k)$ 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 a lower dimensional transform, we show in §4.5 that DROP must estimate $obj_{i+1}$. To do so, 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 $i + 1$, $k_{i+1}$. DROP cannot directly measure $r_{i+1}$ or $k_{i+1}$ without performing iteration $i + 1$, thus performs online progress
estimation. Specifically, DROP performs online parametric fitting to compute future values based on prior values for \( r_i \) and \( k_i \) (Alg. 1, 18). 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:

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

### 4.5 Cost-Based Optimization

DROP must determine if continued PCA on additional samples will improve overall runtime. Given predictions of the next iteration’s runtime \( (\hat{r}_{i+1}) \) and dimensionality \( (\hat{k}_{i+1}) \), DROP uses a greedy heuristic to estimate the optimal stopping point. If the estimated objective value is greater than its current value \( (obj_j < \hat{obj}_{j+1}) \), DROP will terminate. If DROP’s runtime is convex in the number of iterations, we can prove that this condition is the optimal stopping criterion via convexity of composition of convex functions. This stopping criterion leads to the following check at each iteration (Alg.1, 19):

\[
obj_j < \hat{obj}_{j+1} \\
C_d(k_j) + \sum_{j=0}^{i} r_j < C_d(\hat{k}_{i+1}) + \sum_{j=0}^{i} \hat{r}_{j+1} \\
C_d(k_j) - C_d(\hat{k}_{i+1}) < \hat{r}_{i+1}
\]

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

### 4.6 Choice of PCA Subroutine

The most straightforward means of implementing PCA via SVD in DROP is computationally inefficient compared to DR alternatives (§3). DROP computes PCA via a randomized SVD algorithm from [27] (SVD-Halko). Alternative efficient methods for PCA exist (i.e., PPCA, which we also provide), but we found that SVD-Halko is asymptotically of the same running time as techniques used in practice, is straightforward to implement, is 2.5 – 28x faster than our baseline implementations of SVD-based PCA, PPCA, and Oja’s method, and does not require hyperparameter tuning for batch size, learning rate, or convergence criteria.

### 4.7 Work Reuse

A natural question arises due to DROP’s iterative architecture: can we combine information across each sample’s transformations without computing PCA over the union of the data samples? Stochastic PCA methods enable work reuse across samples as they iteratively refine a single transformation matrix, but other methods do not. DROP uses two insights to enable work reuse over any PCA routine.

First, given PCA transformation matrices \( T_1 \) and \( T_2 \), their horizontal concatenation \( H = [T_1|T_2] \) is a transformation into the union of their range spaces. Second, principal components returned from running PCA on repeated data samples generally concentrate to the true top principal components for datasets with rapid spectrum drop off. Work reuse thus proceeds as follows: DROP maintains a transformation history consisting of the horizontal concatenation of all transformations to this point, computes the SVD of this matrix, and returns the first \( k \) columns as the transformation matrix.

Although this requires an SVD computation, computational overhead is dependent on the size of the history matrix, not the dataset size. 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 §5.

## 5 EXPERIMENTAL EVALUATION

We evaluate DROP’s runtime, accuracy, and extensibility. We demonstrate that (1) DROP outperforms PAA and FFT in end-to-end workloads, (2) DROP’s optimizations each contribute to performance, and (3) DROP extends beyond time series.

### 5.1 Experimental Setup

**Implementation** We implement DROP\(^1\) in Java using the multi-threaded Matrix-Toolkits-Java (MTJ) library [28], 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 and k-means.

**Datasets** We first consider the UCR Time Series Classification Archive [15], excluding datasets with fewer than 1 million entries, and fewer datapoints than dimensionality, leaving 14 datasets. As these are all relatively small time series, we consider four additional datasets to showcase DROP’s scalability and generalizability: the MNIST digits dataset [40], the FMA featurized music dataset [19], a sentiment analysis IMDb dataset [8], and the fashion MNIST dataset [54].

**DROP Configuration** We use a runtime model for k-NN and k-means computed via polynomial interpolation on data of varying dimension. While the model is an optional input parameter, any function estimation routine can estimate it given black-box access to the downstream workload. To evaluate sensitivity to runtime model, we report on the effect of operating without it (i.e., sample until convergence). We set TLB constraints such that k-NN accuracy remains unchanged, corresponding to \( B = 0.99 \) for the UCR data. We use

\(^1\)https://github.com/stanford-futuredata/DROP
Figure 3: 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 representations than conventional alternatives (FFT, PAA), and is on average faster than PAA and FFT.

5.2 DROP Performance

We first evaluate DROP’s performance compared to PAA and FFT using the time series case study extended from [20].

k-NN Performance We summarize DROP’s results on a 1-Nearest Neighbor classification in Figure 3. 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 that DROP quickly identifies. We do not display k-NN accuracy as all techniques meet the \( T_{LB} \) 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, translating to significantly smaller k-NN query time. End-to-end runtime with DROP is on average \( 2.2 \times \) and \( 1.4 \times \) (up to 10x and 3.9x) faster than PAA and FFT, respectively, when using brute force linear search. \( 2.3 \times \) and \( 1.2 \times \) (up to 16x and 3.6x) faster when using K-D trees, and \( 1.9 \times \) and \( 1.2 \times \) (up to 5.8x and 2.6x) faster when using cover trees. When evaluating Figure 3, 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 §5.3). Determining if DROP is a good fit for a dataset is an exciting area for future work (§6).

Varying Index-Query Ratio DROP is optimized for a low index-query ratio, as in many streaming and/or high-volume data use cases. If there are many more data points queried than used for constructing an index, but not enough such that expensive, naïve PCA is justified, DROP will outperform alternatives. A natural question that arises is: at what scale is it beneficial to use DROP? While domain experts are typically aware of the scale of their workloads, we provide a heuristic to answer this question given rough runtime and cardinality...
estimates of the downstream task and the alternative DR technique in consideration.

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
\[
\sum_{i=0}^{n_i} r_d + x_d + \sum_{q=0}^{n_q} r_a < \sum_{i=0}^{n_i} r_a + x_a + \sum_{q=0}^{n_q} r_d
\]

To verify, we obtained estimates of the above and empirically validate when running k-NN using cover trees (Figure 4). 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 and FFT in the 1:5 index-query ratio case, where it is on average 1.51× faster than PAA and 1.03× faster than FFT. As the ratio decreases to 1:50, DROP is up to 1.9× faster than alternatives.

### Time Series Similarity Search Extensions

Given the breadth of research in time series indexing, we evaluate how DROP, a general operator for PCA, compares to time series indexes. As a preliminary evaluation, we consider iSAX2+ [12], 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 case, where it is is on average 1.51× faster than PAA and 1.03× faster than FFT. As the ratio decreases to 1:50, DROP is up to 1.9× faster than alternatives.

5.3 Ablation Study

We perform an ablation study of the runtime 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.

Figure 5 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 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.

Work reuse here typically slightly affects end-to-end runtime as it is useful primarily when a large number of DROP iterations are required. We also observe this behavior on certain small datasets with moderate intrinsic dimensionality, such as the yoga dataset in Figure 5b. Work reuse provides a 15% improvement over 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 5c). In this setting, DROP incrementally examines a large
We consider generalizability beyond our initial case study workloads across image classification, music analysis, and a k-means task over the MNIST digits dataset. We fit a downsampling representation of an IMDb sentiment analysis dataset across 25,000 movies with 5000 features [8]; Fashion MNIST’s 70,000 images of dimension 784 [54]. We present our results in Figure 6. As these datasets are larger than those in [15], 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 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.

**5.4 Beyond Time Series**

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

**Data Domain.** We examine classification/similarity search workloads across image classification, music analysis, and natural language processing. We repeat the k-NN retrieval experiments with a 1:1 index-query ratio. We use the MNIST hand-written digit image dataset of 70,000 images of dimension 784 (obtained by flattening each 28 × 28-dimensional image into a single vector [40], combining both the 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 [54]. We present our results in Figure 6. As these datasets are larger than those in [15], 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 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 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 downstream workflow runtime model as we did with k-NN, and operate under a 1:1 index-query ratio. DROP terminates in 1488ms, which is 16.5× and 6.5× faster than PAA and FFT.

**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, Aurooj Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded
