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

ABSTRACT

Index construction via dimensionality reduction is an increasingly critical step in analyzing high-volume, high-dimensional datasets. 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 these techniques 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, our optimizer enables speedups of up to 41× over Singular-Value-Decomposition-based PCA techniques, and achieves parity with or exceeds conventional approaches like FFT and PAA by up to 2×.

1 INTRODUCTION

There has been continued, rapid growth in high-dimensional data volumes from automated data sources [8, 44, 49]. This scale poses a challenge for advanced repeated-query processing operations and analytics tasks where existing datapoints are repeatedly retrieved to process incoming queries against new data, such as in similarity search, clustering, regression, and classification [7, 28]. In such scenarios, indexing using dimensionality reduction (DR) techniques can improve performance by accelerating queries while preserving accuracy [12, 29, 50, 52].

The standard approach of performing Principal Component Analysis (PCA) for DR is frequently the method of choice for practitioners [48]. However, naïve implementations of PCA—for example, those that compute the Singular Value Decomposition (SVD) of the full covariance matrix—scale poorly with dimensionality. The database literature thus advocates trading quality for speed, encouraging the use of alternatives, such as Piecewise Aggregate Approximation (PAA) or Fast Fourier Transforms (FFT) [23, 51] for similarity search, that may not provide the same DR quality as PCA (e.g., may degrade metrics like Euclidean distance further than PCA at the same target dimensionality), but are more efficient to compute.

Recently developed stochastic PCA algorithms are a scalable alternative to those that compute PCA exactly [20, 76]. These algorithms repeatedly process data samples until convergence, reducing PCA's computational overhead. Despite the theoretical promise of these new methods, we are unaware of any empirical study comparing them to alternatives from the database community. Therefore, as a case study, we first extend a highly-cited experimental study of DR methods for time series similarity search from VLDB 2008 [23] and make two observations. First, compared to alternative DR techniques at a target accuracy level, classic PCA via SVD reduces dimensionality by up to 13× (avg: 3×) compared to alternatives, but is up to 700× slower (avg: 138×). Second, sample-based PCA can deliver the *same* quality of dimensionality reduction as classic PCA via SVD while utilizing as little as 1% of the data, providing up to 225× speedups over PCA via SVD. This suggests that new stochastic methods may close the quality-efficiency gap in practice between PCA and today's favored DR methods in end-to-end analytics.

However, the challenge in practically applying stochastic PCA methods is that the amount of sampling required is highly data-dependent, varying from under 1% to over 35% in the time series datasets from the VLDB 2008 study. If we conservatively 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 optimistically 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: how can we efficiently and accurately determine the sampling rate that minimizes total workload runtime while ensuring high accuracy?

In response, we develop DROP, a system that performs whole-workload runtime optimization by dynamically identifying the amount of sampling required for stochastic PCA. As input, DROP takes a high-dimensional dataset (e.g., EKG

SIGMOD'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

data)[1], dataset property to preserve (e.g., pairwise Euclidean distance to 5%), and a runtime model that expresses downstream workload performance as a function of dimensionality (e.g., for k-Nearest Neighbor [k-NN], runtime is linear in dimensionality). As output, DROP returns a low-dimensional transformation of the input data that seeks to minimize the combined runtime of DR and downstream tasks. Thus, DROP obtains a low-dimensional transformation for the input using as few samples as required to minimize the overall workload runtime while satisfying quality constraints.

To achieve the above functionality, DROP addresses the question of how much to sample the input dataset by adapting techniques from the approximate query processing literature: 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. To determine the termination point that minimizes the overall runtime, DROP must overcome three key challenges:

First, given the results of PCA on a data sample, DROP must *evaluate the quality* of the current candidate transformation. While PCA is guaranteed to find the optimal linear transformation with respect to $\mathcal{L}_2$ reconstruction error, popular analytics and data mining tasks (e.g., k-NN [28], k-Means [40], Kernel Density Estimation [81]) instead require approximate preservation of metrics such as average pairwise distances between data points. To overcome this challenge, the system adapts an approach pioneered for deterministic queries in the context of online aggregation: treat quality metrics as aggregation functions and use confidence intervals (either via closed-form or, if unavailable, via bootstrapping) for fast estimation.

Second, DROP must *estimate the marginal benefit of continuing to sample* for another iteration. When running PCA on a series of progressively larger samples, later samples will incur higher computational cost but may in turn return lower-dimensional transformations. To navigate this trade-off between end-to-end runtime and transformation quality, the system performs online progress estimation, using 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* to determine whether to terminate. The system must evaluate if the expected marginal benefit to dimensionality arising from continuing to iterate would reduce total 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 [29], so it is frequently advantageous to trade a slightly higher-dimensional basis for faster preprocessing (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, as well as a useful system for performing whole-workflow optimization of end-to-end data analytics. We make the following contributions in this work:

- We show that the fraction of data required to perform accuracy-achieving PCA on real-world data is often small (as little as 1%), and data-dependent sampling can enable 225× speedup compared to PCA via 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 *xxx*× faster end-to-end execution over PCA via SVD.

2 DIMENSIONALITY REDUCTION FOR END-TO-END WORKLOADS

We provide background on dimensionality reduction (DR) for repeated-query workloads, and revisit a widely cited empirical comparison of DR techniques from VLDB 2008 [23] that we use as a case study. Our study shows 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 [19, 31]. Formally, consider d data vectors (e.g., time series vectors) of length n, $x_i \in \mathbb{R}^n$, with $d > n$. We can represent this as a matrix $X \in \mathbb{R}^{d \times n}$, where each row i corresponds to vector x_i. DR computes a transformation function ($T : \mathbb{R}^n \to \mathbb{R}^k$) 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 $T(X) = \tilde{X} \in \mathbb{R}^{d \times k}$ that preserves some metric of interest.

DR techniques are optimized for various choices of metrics. For instance, DR via Locality Sensitive Hashing [35] can preserve distance metrics such as Hamming distance and Jaccard similarity, and PCA [10] computes a linear transformation that minimizes reconstruction error with respect to the Frobenius norm. In similarity search, a popular metric to preserve is the average Euclidean distances between pairs

[1] Our primary focus for performance evaluation is a case study on time series similarity search, given the amount of study in the database community [23] and the resurgence of interest in time series analytics systems [7, 8, 70]. We explore non-time series data and generalizability in Sections 5 and 6.

of points, which the literature refers to as *tightness of lower bounds* (*TLB*) [23, 29, 51].

2.1.1 Principal Component Analysis (PCA). PCA is a classic, linear DR technique (§7) that identifies a new orthogonal basis for a dataset that captures its directions of highest variance. Of all linear transformation, this basis minimizes reconstruction error in a mean square sense.

Classically implemented PCA uses a Singular Value Decomposition (SVD) routine [78], which provides the matrix decomposition $X = U\Sigma V^\intercal$. Given a data matrix X, PCA via SVD forms the PCA transformation matrix $T : \mathbb{R}^n \rightarrow \mathbb{R}^k$ by first subtracting each column in X by the column's mean to obtain C_X ($\mathbf{1}^\intercal C_X = \mathbf{0}$). The first k right singular vectors of C_X (first k columns of V from the SVD of C_X) comprise T.

2.2 DR for Repeated-Query Workloads

In repeated-query workloads such as similarity search, clustering, or classification, 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 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 [4, 12, 40, 50, 52, 58, 72, 91]; a metric-preserving transformation reduces input dimensionality, and an index is built in this new space for subsequent queries.

2.2.1 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 [23, 35], which we use as a running case study. The *TLB* is useful here to identify the quality of a low dimensional transformation without performing the downstream similarity search task, 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:

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

We focus on Euclidean time series similarity search as our primary means of evaluation given its popularity and large amount of research in the space, but note that there is no requirement that Euclidean distance be used in Equation 1; we futher discuss alternatives in Sections 4.4 and 6.

2.3 Case Study: Speed vs. Quality

To demonstrate the speed-quality trade off inherent in performing DR for repeated-query workloads, we revisit and extend a widely-cited time series similarity search DR study from VLDB 2008 [23]. This work serves as a case study and motivation for this paper since the authors did not evaluate PCA due to it being "untenable for large data sets" despite providing "optimal linear dimensionality reduction."

We first compare PCA via SVD to baseline techniques based on both runtime and DR performance with respect to *TLB* over the largest datasets from [23]. 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× and 3.7× smaller than PAA and FFT for $TLB = 0.75$, and 2.9× and 1.8× smaller for $TLB = 0.99$. While the margin between PCA and alternatives is dataset-dependent (see Table 1 in the appendix), PCA almost always preserves *TLB* with a lower dimensional representation.

Runtime Performance Comparison However, PCA implemented via out-of-the-box SVD routines is on average over 195× slower than PAA and over 82× times slower than FFT when computing the smallest *TLB*-preserving basis (per-dataset breakdown in Table 2). This substantiates [23]'s observation that as classically implemented, PCA is in fact incredibly slow to run compared to alternatives.

As a result of the above, and as noted in Section 5, while improved quality provides faster repeated query execution, the cost of index creation via PCA dominates this speedup—empirically, by over an order of magnitude . This trade off motivates our study of downstream-workload-aware, stochastic, sampling-based PCA methods.

3 SAMPLE-BASED COMPUTATION

Advanced PCA algorithms (Section 7) provide *theoretically efficient* stochastic methods that iterate over small data samples, such as momentum techniques that achieve accelerated convergence rates [20]. However, they either *i*) execute for a pre-specified number of iterations or *ii*) execute until convergence. In the first case, the number of iterations required to compute a basis over a given dataset is highly data-dependent and therefore difficult to specify a priori. In the second case, running to convergence may not be required for many analytics, thus can incur substantial, unnecessary overhead. To our knowledge, existing termination conditions are not suitable when considering users' willingness to trade quality for downstream workload runtime.

Inspired by stochastic methods, we augment our time series case study to show that running PCA on data samples does not sacrifice DR quality, but that number of samples required varies per dataset (i.e., *ii* above). We then show how *progressive sampling*—gradually increasing the number sampled data points—can help dynamically identify how much to sample a given dataset (addressing *i* above).

3.1 Feasibility of Sampling

Many real-world datasets are intrinsically low-dimensional, as evidenced by their rapid eigenvalue spectrum falloff (Section 6). 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 samples required to obtain a TLB-preserving transform with output dimension k equal to input dimension d.

On average, across the considered UCR time series datasets, a sample of under 0.64% of the input is sufficient for a TLB of 0.75, and a sample size under 4.15% is sufficient for a TLB of 0.99 (see Table 3 for a detailed breakdown). Further, in an oracle scenario where this proportion is known, we obtain up to $225\times$ speedup (avg: $57\times$) when using a naïve implementation of PCA via SVD—sans any algorithmic improvement.

3.2 Incremental, Progressive Sampling

As sampling benefit is data-dependent, we must identify how large a sample suffices to compute high-quality transforms. Figure 1 illustrates how the dimensionality required to attain a given TLB changes when we vary dataset and proportion of data sampled. Progressively increasing the number of samples provides lower dimensional transformations of the same quality until convergence to the true PCA solution. This decreases the runtime of downstream applications in exchange for DR time. 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 (a naïve approach, evaluated in Section 5.3) or terminate early (e.g., at 0.3 for SmallKitchenAppliances).

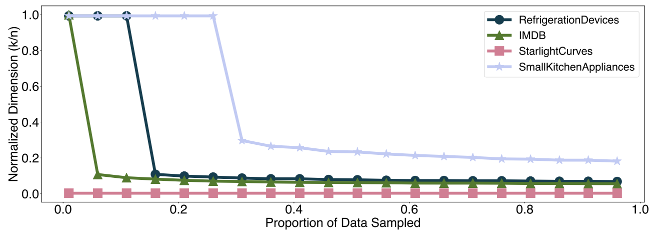


Figure 1: Improvement in representation size for $TLB = 0.80$ across three datasets. Higher sampling rates improve quality until reaching a state equivalent to running PCA over the full dataset ("convergence")

4 DROP: WORKLOAD OPTIMIZATION

Section 3 demonstrated that sampling can reduce the amount of data required to perform DR via PCA, but it is difficult to know a priori how much to sample to attain quality constraints. In response, we 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 many advanced stochastic PCA techniques have traditionally elided: how long should these methods run, and how much computation is actually required to obtain high quality bases? We now discuss workload-aware DR and DROP's architecture.

4.1 Workload-Aware DR

In end-to-end analytics tasks, we wish to minimize the combined runtime of index construction via DR and downstream applications. Similarity search performance heavily depends on the number of workload queries. DR is a fixed cost in index construction, while each query over the dataset incurs a marginal cost that is dependent on DR quality: lower-dimensional data points result in faster queries. Thus, end-to-end runtime is a function of *i*) the time required for DR in index construction (a fixed cost) and *ii*) the benefit of DR as applied to each query (a marginal cost).

In workload-aware dimensionality reduction, we are performing DR to minimize overall workload runtime. As input, we consider a set of data points, desired level of metric preservation (B; default TLB, e.g., $TLB \geq .99$) and, optionally, downstream runtime as a function of dimensionality ($C_d(n)$ for an $d \times n$ data matrix). We seek to use this information to efficiently return a DR function that satisfies the metric constraint with a configurable degree of confidence (default 95%). More formally, denoting DR runtime as R, we define the optimization problem as follows:

Problem 4.1. *Given $X \in \mathbb{R}^{d\times n}$, TLB constraint $B \in (0, 1]$ with 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 $R + C_d(k)$ s.t. $TLB(XT_k) \geq B$ 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. If $C_d(n)$ is unknown, there exist a number of estimation routines for relational workloads that users can use to approximate their downstream runtimes [41, 42, 90]. For general functions that users may only possess black-box knowledge of, such as those used in time series analytics, regression analysis can be used: the user can input datasets of

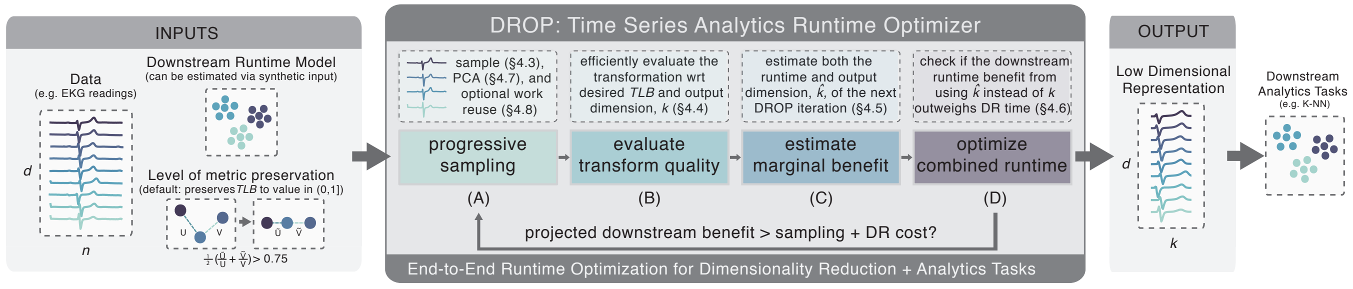

INPUTS
Data
(e.g. EKG readings)
d
n
Downstream Runtime Model
(can be estimated via synthetic input)
Level of metric preservation
(default: preserves TLB to value in (0,1])
u
v
û
v̂
$\frac{1}{2}(\frac{\hat{u}}{u}+\frac{\hat{v}}{v}) > 0.75$
DROP: Time Series Analytics Runtime Optimizer
sample (§4.3), PCA (§4.7), and optional work reuse (§4.8)
efficiently evaluate the transformation wrt desired TLB and output dimension, k (§4.4)
estimate both the runtime and output dimension, k̂, of the next DROP iteration (§4.5)
check if the downstream runtime benefit from using k̂ instead of k outweighs DR time (§4.6)
progressive sampling
evaluate transform quality
estimate marginal benefit
optimize combined runtime
(A)
(B)
(C)
(D)
projected downstream benefit > sampling + DR cost?
End-to-End Runtime Optimization for Dimensionality Reduction + Analytics Tasks
OUTPUT
Low Dimensional Representation
d
k
Downstream Analytics Tasks
(e.g. K-NN)

Figure 2: High-level DROP architecture depicting DROP's inputs, outputs, and core components.

varying dimension, and build a model for the function's runtime. For example, we use polynomial regression to model the cost of our black-box k-NN task in our evaluation. 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.

As shown in Section 3, the more time spent on DR (R), the smaller the resulting transformation (k), thus the lower the downstream task runtime. Hence, to minimize $R + C_d(k)$ in Problem 4.1, we must determine how much time to spend on DR to reduce end-to-end time.

4.2 DROP Architecture

DROP operates over a series of progressively larger data samples, and determines when to terminate via four-step procedure that is repeated for each iteration: progressive sampling, transformation evaluation, progress estimation, and cost-based optimization. To power this pipeline, DROP combines database and machine learning techniques spanning online aggregation (§4.4), progress estimation (§4.5), progressive sampling (§4.3), and PCA approximation (§§4.7,4.8) for the first time that we are aware.

We now provide a brief overview of DROP's sample-based iterative architecture before detailing each.

Step 1: Progressive Sampling (§4.3, Alg 1 L5, Fig 2A) At each iteration, DROP draws a data sample and computes PCA over this sample. Additionally, DROP makes use of a novel means of reusing work across iterations(§4.8).

Step 2: Transform Evaluation (§4.4, Alg 1 L6, Fig 2B) Given the result from PCA computed over a data sample, DROP evaluates quality by identifying the size of the smallest metric-preserving transformation that can be extracted.

Step 3: Progress Estimation (§4.5, 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 (§4.6, 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.

4.3 Progressive Sampling

DROP repeatedly chooses a subset of data and computes PCA on the subsample (via one of several methods described in Section 4.7) at each iteration. By default, we consider a simple uniform sampling strategy where at each iteration, DROP samples a fixed percentage of the data. While we considered a range of alternative sampling strategies, uniform sampling strikes a balance between computational and statistical efficiency. Data-dependent and weighted sampling schemes that are dependent on the current basis may decrease the total number of iterations required by DROP, but may require expensive reshuffling of data at each iteration [17].

DROP provides configurable strategies for both base number of samples and the per-iteration increment, in our experimental evaluation in Section 5, we consider a sampling rate of 1% per iteration. We discuss more sophisticated additions to this base sampling schedule in Section 6.

4.4 Evaluating Transformations

Given a transformation obtained by running PCA over a sample (Section 3), DROP must accurately and efficiently evaluate the performance of this transformation with respect to a metric of interest. To do so, DROP adapts an approach for deterministic queries in online aggregation: treating quality metrics as aggregation functions and utilizing confidence intervals for fast estimation. We first discuss this approach in the context of *TLB*, then discuss how to extend this approach to alternative metrics at the end of this section.

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.

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

```
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($i$))    ▷ § 4.3
        $T_{k_i}$ = EVALUATE($X, X_i, B$)             ▷ § 4.4
        $r_i$ = CLOCK.ELAPSED                         ▷ $R = \sum_i r_i$
        $\hat{k}_{i+1}, \hat{r}_{i+1}$ = ESTIMATE($k_i, r_i$)   ▷ § 4.5
    while OPTIMIZE($C_d, k_i, r_i, \hat{k}_{i+1}, \hat{r}_{i+1}$)   ▷ § 4.6
return $T_{k_i}$
```

Second, brute-force *TLB* computation would dominate the runtime of computing PCA over a sample.

4.4.1 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 also 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.

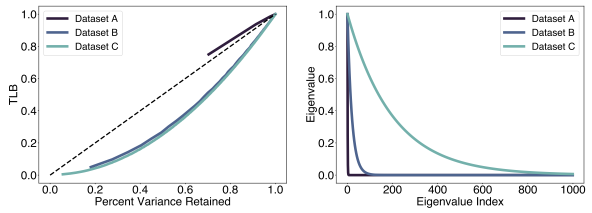
Computing the full n-dimensional basis at every step may be wasteful. To avoid this need, DROP's exploits information from previous iterations: if DROP has previously found a candidate *TLB*-preserving basis of size $n' < n$ in prior iterations, then DROP only computes n' components at the start of the next iteration, as more samples should only allow DROP to better capture data behavior. 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).

4.4.2 TLB Computation. Given a transformation, DROP must efficiently determine if the basis preserves the desired *TLB*. Computing pairwise *TLB* for all data points requires $O(d^2 n)$ 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 adapt techniques from online aggregation and approximate query processing [43, 69], 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 [43]): 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 [43, 67, 86], or bootstrap-based methods [26, 55], which incur higher overhead but can be more generally applied.

4.5 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 $R + C_d(k)$ s.t. $TLB(XT_k) \geq B$, 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 dimensionality-runtime 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 4.6 we show that DROP requires obj_{i+1} to minimize this objective function. Therefore, to estimate obj_{i+1},

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; $k_i = \frac{1}{2}$(low + high); $B_i = 0$
 4:     while (low ! = high) do
 5:         $T_{k_i}, B_i$ = EVALUATE-TLB($X, B, k_i$)
 6:         if $B_i \leq B$ then low = $k_i + 1$
 7:         else high = $k_i$
 8:         $k_i = \frac{1}{2}$(low + 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 ($p <$ numPairs) do
15:         $B_i, B_{lo}, B_{hi}$ = TLB($X, p, k$)
16:         if ($B_{lo} > B$ or $B_{hi} < B$) then break
17:         else pairs ×= 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.
```

DROP must estimate the runtime required for iteration $i+1$ (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 uses linear extrapolation as we found it to be accurate on our workloads, but the architecture can incorporate more sophisticated progress estimation functions (§7).

4.6 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:

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

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 need 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 encountered 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 before entering the convex regime (see SmallKitchenAppliances in Fig. 1). To combat this challenge, optimization is only enabled once a feasible point is attained as we choose to prioritize accuracy over runtime (i.e., at 0.3 for SmallKitchenAppliances in Fig. 1). We describe this decision's performance impact in Section 5.3, and extensions to the streaming setting in Section ??.

4.7 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 (§2.1.1). 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 [38]. 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 [27]), 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 [27], or convergence rate, as recent work in accelerated, momentum-based PCA [20]. However, these techniques can be easily substituted for SVD-Halko in DROP's architecture. We demonstrate this by also implementing and evaluating multiple alternatives in Section 5.6. Further, we also demonstrate that this implementation is competitive with widely used python and C++ libraries.

4.8 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 25% 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) For typical end-to-end, repetitive-query workloads, DROP outperforms PAA and FFT (§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 operator.[2] We implement DROP in Java using the multi-threaded Matrix-Toolkits-Java (MTJ) library for compute-intensive linear algebra operations including matrix multiply and SVD [39]. We use multi-threaded JTransforms [2] to implement 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.

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 [16], 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 [59], the FMA featurized music dataset [22], and a labeled sentiment analysis IMBD dataset [5], which also demonstrate extensibility beyond time series data.

DROP Configuration We employ a runtime cost function for k-NN obtained via linear interpolation on data of varying dimension (implemented via cover trees [9], K-D trees [75],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. We further discuss sampling schedules and properties that make a dataset amenable to DROP in Section 6.

Baselines We report runtime, accuracy, and reduced dimension compared to FFT, PAA, and SVD-Halko (we implemented via MTJ). Each of these methods computes a transformation over the entire data, then performs binary search to identify the smallest dimensional basis that satisfies the target TLB. We further discuss choice of PCA subroutine in Section 5.6.

Similarity Search/k-NN Setup While many methods for similarity search exist, as in [23], 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 [65, 74, 84]. Further, adopting k-NN, 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 it's indexing benefits to each query point.

5.2 DROP Performance

In this section, we evaluate DROP's performance in comparison to PAA and FFT in a repetitive-query workload setting, and demonstrate when using alternatives may be preferable.

k-NN Performance We summarize DROP's results on an end-to-end 1-Nearest Neighbor classification in Figure 3. We display the total, end-to-end runtime of DROP, PAA, and FFT for each of the considered index structures: no index, K-D trees, cover trees. We only 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 possess low intrinsic dimensionality. As a result, DROP frequently identifies this low dimensionality prior to termination, though spends more time doing so depending on the downstream task. We observe some variability in XXX, due to their higher intrinsic

[2]https://github.com/anonimized

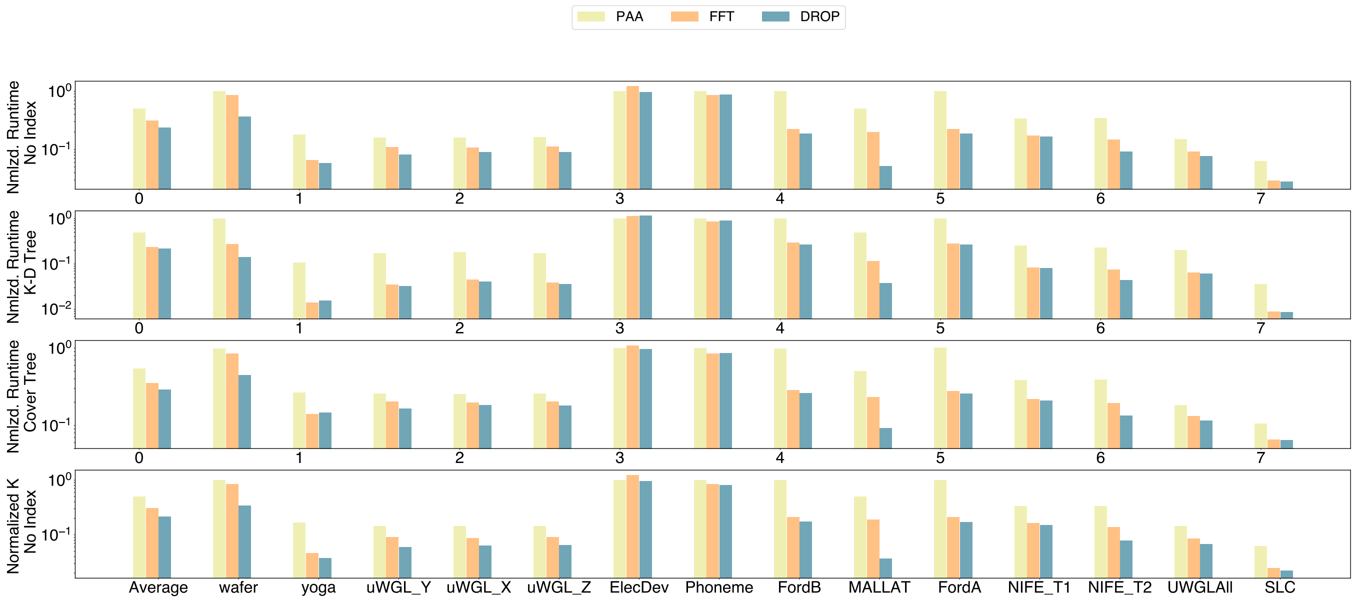


Figure 3: End-to-End DR and k-NN runtime (top three) and returned lower dimension (bottom) over the largest UCR datasets for three different indexing routines, all normalized to no DR. DROP consistently returns lower dimensional bases than conventional alternatives (FFT, PAA), and is on average faster than PAA and FFT.

dimensionality, causing DROP to select different termination points for the different tasks. We do not display k-NN accuracy results as all techniques meet the *TLB* constraint, and achieve the same accuracy rate within 1%.

On average, DROP returns transformations that are 2.3× and 1.4× 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 2.1× and 1.3× faster than PAA and FFT, respectively when using brute force linear search, 2.2× and 1.1× faster when using K-D trees, and 1.9× and 1.2× faster when using cover trees. We demonstrate in our lesion study in Section 5.3 that DROP also significantly outperforms our baseline PCA via SVD implementation, as well as our SVD-Halko implementation.

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 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 this empirically, we obtained estimates of the above and compared DROP against FFT and PAA in two lower-query-volume scenarios when running K-NN using cover trees, and display the results in Figure 4. When compared to PAA, we first found that in the 1:1 index-query ratio setting, DROP should be slightly slower than PAA, as observed in Figure 4. However, as we decrease the ratio, DROP becomes faster, with a break-even point of slightly lower than 1:3. We display that DROP does indeed outperform PAA in the 1:5 index-query ratio case, where it is is on average 1.3× faster than PAA. Similarly, we do so with FFT, where with an index-query ratio of 1:50, DROP is 1.2× faster than FFT and 1.83× faster than PAA.

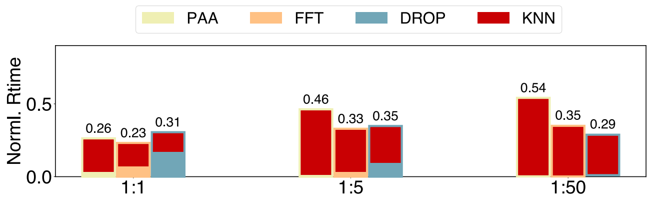


Figure 4: Effect of decreasing the index-query ratio. As an index is queried more frequently, DROP's relative runtime benefit increases.

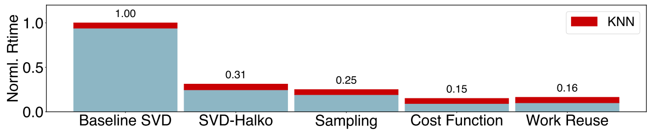


Figure 5: Average result of lesion study over the UCR datasets depicting contribution from each optimization.

5.3 Lesion Study

In this section, 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 to better highlight the effects of each contribution to the DR routine, and display average results over the UCR datasets in Figure 5, excluding Phoneme, which we detail individually.

Figure 5 first demonstrates the boost from using SVD-Halko over a naïve implementation of PCA via SVD. It then shows the runtime boost obtained from running on samples until convergence, where DROP continuously draws 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, followed by 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 7.8× faster (up to 41×) than PCA via SVD, and 1.9× faster than SVD-Halko (up to 3.3×). DROP with cost-based optimization is faster than naïvely sampling to convergence (default) by on average 1.6×, but this default strategy is still 1.2× faster than SVD-Halko on average.

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). However, we observe this behavior on certain

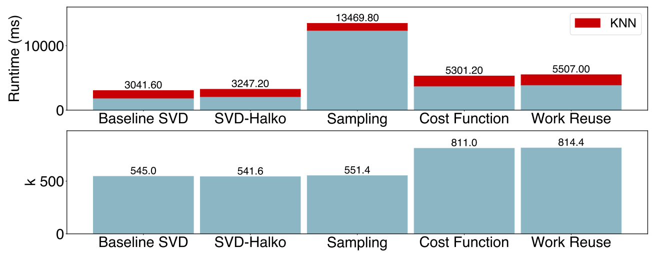


Figure 6: Lesion study of the UCR phoneme, a dataset with high intrinsic dimensionality. The entire dataset is required for satisfactory DR, hence sampling to convergence (default) is orders of magnitude slower than a batch SVD. DROP's cost function enables it to terminate in advance, returning a higher dimensional basis to minimize compute via progressive sampling.

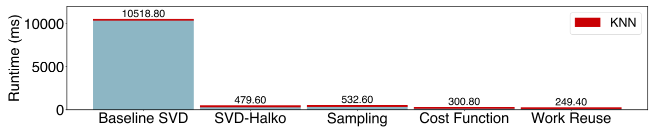


Figure 7: Lesion study over the UCR wafer dataset. Work reuse provides a 25% runtime improvement.

small datasets with moderate intrinsic dimensionality, such as the wafer dataset in Figure 7. Work reuse provides a 25% improvement in addition to cost based optimization.

As noted throughout this work, DROP's sampling operates on the premise that the dataset has row, or data-point-level repetition. However, datasets that do not possess this type of structure are more difficult to reduce the dimensionality over. Phoneme is an example of one such dataset (Figure 6). In this setting, DROP must examine a large proportion of datapoints, resulting in a performance penalty. We discuss potential extensions to DROP to more intelligently mitigate this problem in Section 6, and provide all lesion studies in the Appendix.

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 low 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

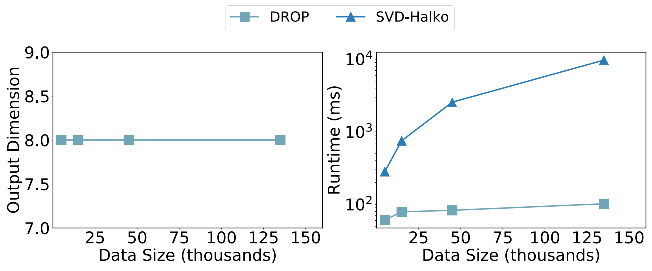


Figure 8: Effect of dataset size on time and output dimension (k), with constant intrinsic data dimensionality of 8. DROP runtime with a fixed schedule remains near constant.

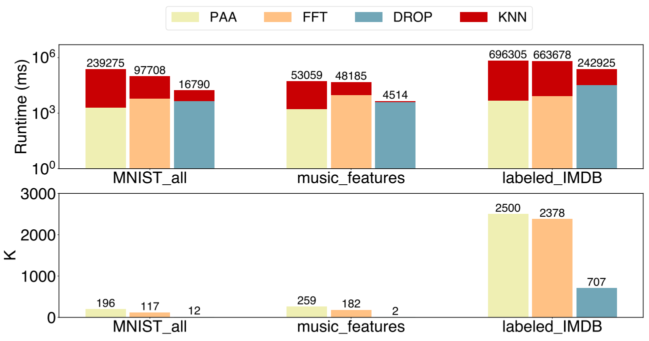


Figure 9: End-to-End k-NN runtime (top) and returned dimension k (bottom) over the entire MNIST dataset and the FMA featurized music dataset.

dataset size. In this experiment, we enable DROP's fixed-size sampling schedule set to increase by 500 datapoints at each iteration. As Figure 8 shows, DROP is able to find a 8-dimensional basis that preserves *TLB* to 0.99 within 102ms for dataset sizes up to 135K data points, and is 95× 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 Preliminary Study: Beyond Time Series

As an application to larger datasets and generalizability beyond time series, we examine workloads from image classification, music analysis, and natural language processing. We repeat the k-NN retrieval experiments above using 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 [59], combining both the provided training and testing datasets), FMA's featurized music dataset, providing 518 features across 106,574 music tracks, and a bag-of-words representation of an IMDB sentiment analysis dataset across 25,000 movies with 5000 features [5]. We present our results in Figure 9. As the given datasets are larger than the ones presented in [16], 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 the IMDB case compared to PAA. However, we also note that FFT is not well suited for this sparse task, hence its poor performance; additional DR baselines must be considered as well for a thorough analysis of application to non-time series. These results provide evidence that the sampling-based dimensionality reduction provided by DROP is effective for other structured, high-volume datasets as well.

5.6 PCA Subroutine Evaluation

TODO: compare instead to numpy and C++ here **Java Comparisons** As stated in Section 4.7, PCA algorithms are optimized for different purposes, with varying convergence, runtime, and communication complexity guarantees. DROP is agnostic to choice of PCA subroutine, and improvements to said routine provide complementary runtime benefits. To select DROP's default algorithm, we implemented and integrated PCA via SVD using MTJ, PCA via SMILE, Probabilistic PCA via SMILE's implementation, and PCA via (stochastic) Oja's method. We then evaluated their performance via oracle experiment over the UCR datasets. Here, we first precompute the size of transformation, required to preserve the dataset's *TLB* to 0.99. We then compare the time taken by each technique to compute this transformation (tuning the the batch size and learning rate for Oja's method). In most instances, SVD-Halko outperformed our alternatives, especially if the required dimensionality was small, as in the largest, StarLightCurves dataset, which required 42.7, 29.2, 8.2, 3.7, and 1.5 seconds for PPCA-SMILE, SVD-MTJ, PCA-Oja's, PCA-SMILE, and SVD-Halko, respectively. This justifies our use of SVD-Halko as a simple-to-implement, parameter-free default.

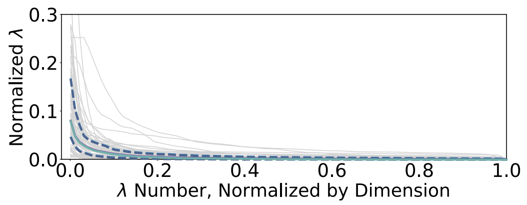
6 EXTENSIONS

In this section, we describe several extensions to DROP.

6.1 Smarter Sampling

We demonstrated that workload-aware approximate PCA can provide large end-to-end speedups in time series tasks, and can extend to structured, non-time series datasets. Our results also revealed the data-dependent nature of dimensionality reduction. Certain datasets are amenable to extreme, aggressive sampling (StarLightCurves), but others are not (Phoneme). Datasets generated via mostly regular, automated processes are prime candidates to obtain runtime speedups via DROP's iterative approach.

The efficiency of sample-based methods is determined by the dataset's eigenvalue spectrum. To demonstrate, we plot the spectrum of each of the 80 UCR time series' [16] eigenvalue spectrum (normalized by each dataset's sum of eigenvalues), together with the 50th, 25th, and 75th percentiles.



For these datasets with poor drop off in spectrum, DROP's default configuration will not provide strong runtime improvements. We instead provide an alternate sampling schedule that identifies when DROP is performing poorly, and increases the sampling rate to more quickly reach a *TLB*-achieving state. Extending DROP to more intelligently determine if a dataset is amenable to aggressive sampling is an exciting area of future work, for instance, by using recent theoretical results from last year that use sampling to estimate spectrum even when the number of samples are small in comparison to the input dimensionality [56]. Such techniques can be run alongside DROP, with minimal alteration.

6.2 Smarter Termination

Here's where the stuff the reviewer spoke about on being more robust to plateus might go...but their comments don't really apply imo. TBH I think I should merge it with above, and call out phoneme as a case where bad termination condition goes wrong.

6.3 Streaming Execution

Streaming stuff goes here—budget a paragraph or two.

6.4 Beyond Time Series

Call out that we focused on this case study heavily. Highlight and recap again all the time series specific stuff we discussed and how to overcome this—budget 3 paragraphs or so for this.

7 RELATED WORK

Dimensionality Reduction Dimensionality reduction is a classic operation in analytics [19, 31, 60, 78] and is well studied in the database [4, 12, 52, 72], data mining [51, 53, 54, 61], statistics and machine learning [21, 76], and theoretical computer science [37, 46] communities, with techniques for use in pre-processing [33, 40, 87], indexing [29, 50, 58, 91], and visualizing [62, 73, 80] datasets.

In this paper, inspired by [23], we study the problem of reducing the dimensionality of increasingly prevalent high-volume time series data [28]. We extend [23] by considering PCA, which was previously eschewed due to its cost at scale.

Recent breakthroughs in the theoretical statistics community provided new algorithms for PCA that promise substantial scalability improvements without compromising result quality [20, 21, 24, 38, 46, 63, 79]. Foremost among these techniques are advanced stochastic methods [20, 76], and techniques for randomized SVD [38]. 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 [27]. As a proof of concept of our method, we provide implementations of full SVD-based PCA, power iteration, as well as Oja's method. 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 [51], especially on real datasets. In addition, DROP *combines* these methods with row-level sampling and optional work reuse to provide benefits similar to using stochastic methods for PCA, regardless of the chosen PCA subroutine.

This setting differs from that of Moving Window (or Rolling) PCA in that the these methods assume overlap among the data samples, whereas here our samples are independently drawn from the same underlying data distribution [82].

Approximate Query Processing A core problem in DROP is determining the appropriate sample size for both basis computation and basis evaluation. To address this challenge, we turned to the approximate query processing literature.

Inspired by approximate query processing engines [69] as in online aggregation [43], DROP performs progressive sampling, drawing only as many samples as required to attain a *TLB* threshold. Similar to work including [32], this threshold-based pruning strategy [45] provides *data-dependent* runtime as opposed to data-agnostic. In contrast with more general data dimensionality estimation methods [11], 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 [6, 13], including sampling strategies that are aware of query histories [34] and storage hierarchies [71]. More sophisticated sampling routines are extremely promising areas of future work, but their runtime cost must be weighed against their potential benefit.

Finally, DROP performs online progress estimation to minimize the end-to-end analytics cost function. This is analogous to query progress estimation [14, 15, 66] and performance prediction [25, 68] in database and data warehouse settings and has been exploited in approximate query processing engines such as BlinkDB [3, 88]. DROP adopts a relatively simple derivative-based estimator but may benefit from more sophisticated techniques from the literature.

Scalable Complex Analytics As a framework for dimensionality reduction, DROP is designed as an operator for analytics dataflow pipelines. Thus, DROP is as an extension of recent results on integrating complex analytics function including signal processing [18, 36, 49, 70], model training [30, 47, 57], and data exploration [64, 77, 83, 85, 89] operators into scalable analytics engines. In the implementation used in this paper, we evaluate DROP as a custom feature transformation dataflow operator, combining it with downstream tasks.

8 CONCLUSION

Advanced data analytics techniques must scale to rising time series volumes. Indexing these time series is increasingly important to provide high performance, as new data is typically repeatedly queried against historical data or models. DR techniques offer a powerful toolkit when indexing these time series, 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 is able to bridge the gap between quality and efficiency in time series dimensionality reduction 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] 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.
[4] Charu C Aggarwal. 2001. On the effects of dimensionality reduction on high dimensional similarity search. In *PODS*.
[5] 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*.
[6] 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.
[7] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*. ACM.
[8] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. 2017. Prioritizing Attention in Fast Data: Principles and Promise.
[9] 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.
[10] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.
[11] Francesco Camastra. 2003. Data dimensionality estimation methods: a survey. *Pattern recognition* 36, 12 (2003), 2945–2954.
[12] Kaushik Chakrabarti and Sharad Mehrotra. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*.
[13] 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.
[14] Surajit Chaudhuri and Vivek Narasayya. 2007. Self-tuning database systems: a decade of progress. In *VLDB*. VLDB Endowment.
[15] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. 2004. Estimating progress of execution for SQL queries. In *SIGMOD*.
[16] 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/.
[17] Kenneth L Clarkson. 2010. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms (TALG)* 6, 4 (2010), 63.
[18] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: a stream database for network applications. In *SIGMOD*.
[19] John P Cunningham and Zoubin Ghahramani. 2015. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research* 16, 1 (2015), 2859–2900.
[20] Christopher De Sa, Bryan He, Ioannis Mitliagkas, Chris Re, and Peng Xu. 2018. Accelerated Stochastic Power Iteration. In *Artificial Intelligence and Statistics*.
[21] Christopher De Sa, Kunle Olukotun, and Christopher Ré. 2015. Global Convergence of Stochastic Gradient Descent for Some Non-convex Matrix Problems. In *ICML*.
[22] 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*.
[23] 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*.
[24] Petros Drineas and Michael W Mahoney. 2016. RandNLA: randomized numerical linear algebra. *Commun. ACM* 59, 6 (2016), 80–90.
[25] Jennie Duggan, Ugur Cetintemel, Olga Papaemmanouil, and Eli Upfal. 2011. Performance prediction for concurrent database workloads. In *SIGMOD*.
[26] Bradley Efron and Robert Tibshirani. 1997. Improvements on cross-validation: the 632+ bootstrap method. *J. Amer. Statist. Assoc.* 92, 438 (1997), 548–560.
[27] Tarek Elgamal, Maysam Yabandeh, Ashraf Aboulnaga, Waleed Mustafa, and Mohamed Hefeeda. 2015. sPCA: Scalable Principal Component Analysis for Big Data on Distributed Platforms. In *SIGMOD*.
[28] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 12.

[29] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. *Fast subsequence matching in time-series databases.*
[30] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD.*
[31] Imola K Fodor. 2002. *A survey of dimension reduction techniques.* Technical Report. Lawrence Livermore National Lab., CA (US).
[32] Edward Gan and Peter Bailis. 2017. Scalable Kernel Density Classification via Threshold-Based Pruning. In *SIGMOD.*
[33] Allen Gersho and Robert M Gray. 2012. *Vector quantization and signal compression.* Vol. 159. Springer Science & Business Media.
[34] Phillip B Gibbons and Yossi Matias. 1998. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD.*
[35] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.
[36] Lewis Girod et al. 2006. Wavescope: a signal-oriented data stream management system. In *ICDE.*
[37] Navin Goyal, Santosh Vempala, and Ying Xiao. 2014. Fourier PCA and Robust Tensor Decomposition. In *STOC.*
[38] 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.
[39] Sam Halliday and BjÃÿrn-Ove Heimsund. 2008. matrix-toolkits-java. (2008). https://github.com/fommil/matrix-toolkits-java.
[40] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques.* Elsevier.
[41] Zhen He, Byung Suk Lee, and Robert Snapp. 2005. Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *ACM Transactions on Database Systems (TODS)* (2005).
[42] Zhen He, Byung S Lee, and Robert R Snapp. 2004. Self-tuning UDF cost modeling using the memory-limited quadtree. In *International Conference on Extending Database Technology.* Springer, 513–531.
[43] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *SIGMOD.*
[44] IDC. 2014. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things. (2014). http://www.emc.com/leadership/digital-universe/.
[45] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 11.
[46] 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.*
[47] 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.*
[48] Ian T Jolliffe. 1986. Principal component analysis and factor analysis. In *Principal component analysis.* Springer, 115–128.
[49] Yannis Katsis, Yoav Freund, and Yannis Papakonstantinou. 2015. Combining Databases and Signal Processing in Plato.. In *CIDR.*
[50] Eamonn Keogh. 2006. A decade of progress in indexing and mining large time series databases. In *VLDB.*
[51] 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.
[52] 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.
[53] Eamonn Keogh, Jessica Lin, and Ada Fu. 2005. Hot sax: Efficiently finding the most unusual time series subsequence. In *ICDM.*
[54] Eamonn Keogh and Michael Pazzani. 2000. A simple dimensionality reduction technique for fast similarity search in large time series databases. *Knowledge Discovery and Data Mining. Current Issues and New Applications* (2000), 122–133.
[55] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. 2012. The big data bootstrap. *arXiv preprint arXiv:1206.6415* (2012).
[56] W. Kong and G. Valiant. 2017. Spectrum Estimation from Samples. *Annals of Statistics* 45, 5 (2017), 2218–2247.
[57] 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.*
[58] Hans-Peter Kriegel, Peer Kröger, and Matthias Renz. 2010. Techniques for efficiently searching in spatial, temporal, spatio-temporal, and multimedia databases. In *ICDE.* IEEE.
[59] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun.com/exdb/mnist/* (1998).
[60] John A Lee and Michel Verleysen. 2007. *Nonlinear dimensionality reduction.* Springer Science & Business Media.
[61] 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.
[62] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.
[63] Michael W Mahoney et al. 2011. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning* 3, 2 (2011), 123–224.
[64] Alexandra Meliou, Sudeepa Roy, and Dan Suciu. 2014. Causality and explanations in databases. In *VLDB.*
[65] 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.
[66] Chaitanya Mishra and Nick Koudas. 2007. A lightweight online framework for query progress indicators. In *ICDE.*
[67] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis.* Cambridge university press.
[68] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. 2010. Estimating the progress of MapReduce pipelines. In *ICDE.*
[69] Barzan Mozafari. 2017. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD.*
[70] Milos Nikolic, Badrish Chandramouli, and Jonathan Goldstein. 2017. Enabling Signal Processing over Data Streams. In *SIGMOD.* ACM.
[71] Frank Olken and Doron Rotem. 1995. Random sampling from databases: a survey. *Statistics and Computing* 5, 1 (1995), 25–42.
[72] KV Ravi Kanth, Divyakant Agrawal, and Ambuj Singh. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD.*
[73] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290, 5500 (2000), 2323–2326.
[74] Ruslan Salakhutdinov and Geoff Hinton. 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics.* 412–419.
[75] 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.
[76] Ohad Shamir. 2015. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *ICML.* 144–152.

[77] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. In *VLDB*.

[78] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.

[79] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. 2016. Generalized low rank models. *Foundations and Trends® in Machine Learning* 9, 1 (2016), 1–118.

[80] Michail Vlachos, Carlotta Domeniconi, Dimitrios Gunopulos, George Kollios, and Nick Koudas. 2002. Non-linear dimensionality reduction techniques for classification and visualization. In *KDD*.

[81] M.P. Wand and M.C. Jones. 1994. *Kernel Smoothing*. Taylor & Francis. https://books.google.com/books?id=GTOOi5yE008C

[82] Xun Wang, Uwe Kruger, and George W Irwin. 2005. Process monitoring approach using fast moving window PCA. *Industrial & Engineering Chemistry Research* (2005).

[83] Abdul Wasay, Xinding Wei, Niv Dayan, and Stratos Idreos. 2017. Data Canopy: Accelerating Exploratory Statistical Analysis. In *SIGMOD*.

[84] 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.

[85] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*.

[86] Taro Yamane. 1973. Statistics: An introductory analysis. (1973).

[87] Jieping Ye. 2004. Generalized low rank approximations of matrices. In *ICML*.

[88] Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. 2014. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*.

[89] E. Zgraggen, 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).

[90] Ning Zhang, Peter J Haas, Vanja Josifovski, Guy M Lohman, and Chun Zhang. 2005. Statistical learning techniques for costing XML queries. In *Proceedings of the 31st international conference on Very large data bases*.

[91] Yunyue Zhu and Dennis Shasha. 2003. Warping indexes with envelope transforms for query by humming. In *SIGMOD*.

APPENDIX

A AUGMENTED RESULTS

In this section, we provide additional information to augment results provided in the measurement study and evaluation. Table **??** displays the proportion of data required to attain a given TLB when using a PCA transformation where output dimension is equal to input dimension. Table **??** illustrates the output dimension required for each algorithm (PAA, FFT, and PCA) to attain a target TLB. Finally, we provide all of the remaining lesion studies for the UCR dataset.

Table 1: Normalized lower dimension for target TLB across DR techniques. PCA admits lower dimension for most UCR time series datasets (additional datasets in the Appendix)

	TLB:0.75			**TLB:0.99**		
Dataset (dimension)	*PAA*	*FFT*	*PCA*	*PAA*	*FFT*	*PCA*
yoga (426)	0.375	0.375	0.281	0.812	0.822	0.770
ElectricDevices (96)	0.126	0.094	0.032	0.594	0.212	0.164
FordA (500)	0.138	0.098	0.038	0.636	0.214	0.17
FordB (500)	0.018	0.029	0.009	0.290	0.177	0.035
MALLAT (1024)	0.084	0.068	0.057	0.898	0.837	0.522
Phoneme (1024)	0.004	0.026	0.001	0.049	0.035	0.034
StarLightCurves (1024)	0.015	0.028	0.019	0.037	0.086	0.062
UWGLAll (945)	0.078	0.065	0.026	0.822	0.736	0.322
wafer (152)	0.014	0.030	0.007	0.103	0.049	0.037

Table 2: Runtime (in ms) of 4 DR techniques. PCA is slowest, and can be over 700× slower than PAA. Running SVD over a sample (sampling) can bridge this gap

Dataset	**PAA (× SVD)**	**FFT**	**SVD**	**SVD-Halko**	**sampling**
ElectricDev	75 (713×)	113	53483	420	237
FordA	42 (120×)	224	9039	1742	517
FordB	44 (144×)	173	6339	1270	338
MALLAT	46 (66×)	327	3023	3058	486
Phoneme	49 (50×)	270	2465	2987	3032
SLC	155 (227×)	770	35212	8350	472
UWGLAll	205 (11×)	769	8382	3766	498
wafer	60 (215×)	134	12895	343	97
yoga	36 (89×)	200	3201	770	128

Table 3: A small proportion of data is needed to obtain a TLB-preserving transform with full PCA (output = input dimension)

	TLB		
Dataset (number of datapoints)	**0.75**	**0.90**	**0.99**
ElectricDevices (16637)	0.0026	0.0043	0.0088
FordA (4921)	0.0054	0.0114	0.0198
FordB (4446)	0.008	0.0146	0.0248
MALLAT (2400)	0.0031	0.009	0.0197
Phoneme (2110)	0.0547	0.1346	0.3875
StarLightCurves (9236)	0.001	0.0011	0.0039
UWaveGestureLibraryAll (4478)	0.0025	0.0056	0.024
wafer (7164)	0.001	0.0032	0.0097
yoga (3300)	0.0017	0.0028	0.0096