A Unified Approach to Coreset Learning
Alaa Maalouf, Gilad Eini, Ben Mussay, Dan Feldman, and Margarita Osadchy

Abstract—Coreset of a given dataset and loss function is usually a small weighted set that approximates this loss for every query from a given set of queries. Coresets have shown to be very useful in many applications. However, coresets’ construction is done in a problem-dependent manner and it could take years to design and prove the correctness of a coreset for a specific family of queries. This could limit coresets’ use in practical applications. Moreover, small coresets provably do not exist for many problems. To address these limitations, we propose a generic, learning-based algorithm for construction of coresets. Our approach offers a new definition of coreset, which is a natural relaxation of the standard definition and aims at approximating the average loss of the original data over the queries. This allows us to use a learning paradigm to compute a small coreset of a given set of inputs with respect to a given loss function using a training set of queries. We derive formal guarantees for the proposed approach. Experimental evaluation on deep networks and classic machine learning problems show that our learned coresets yield comparable or even better results than the existing algorithms with worst case theoretical guarantees (that may be too pessimistic in practice). Furthermore, our approach applied to deep network pruning provides the first coreset for a full deep network, i.e., compresses all the networks at once, and not layer by layer or similar divide-and-conquer methods.

Index Terms—Coresets, data summarization, generalization, learning.

I. INTRODUCTION

CORESET is usually defined as a small weighted subset of the original input set that provably approximates the given loss (objective) function for every query in a given set of queries. Coresets are useful in machine learning applications as they offer significant efficiency improvements. Namely, the traditional (possibly inefficient, but provably optimal) algorithms can be applied on coresets to obtain an approximation of the optimal solution on the full dataset using time and memory that are smaller by order of magnitudes. Moreover, the existing heuristics that already run fast can be improved in terms of accuracy by running them many times on the coreset in the time it takes for a single run on the original (big) dataset. In addition, coresets can be maintained for a distributed and streaming data, where the stream is distributed in parallel from a server to m machines (e.g., cloud), and the goal is to maintain the optimal solution (or an approximation to it) for the whole input seen so far in the stream using small update time, memory, and communication to the server. This is very useful when we are working on a machine with a limited memory which can store only a small number of input items at once. In this case, by maintaining a small coreset for the whole (big) data on this limited machine (and solving the given problem on the coreset), we can approximate the solution for the whole data or infinite stream. Finally, coresets support constraint optimization, where the goal is to solve an optimization problem under some constraints (e.g., requiring a sparse classifier/solution, or requiring that the output is a point from the input data).

In the recent decade, coresets, under different formal definitions, were applied to many machine learning algorithms e.g., logistic regression [1], [2], support vector machine (SVM) [3], [4], [5], [6], clustering problems [7], [8], [9], matrix approximation [10], [11], [12], [13], ℓp-regression [14], [15], [16], and others; see surveys [17], [18], [19].

Some attempts of using coresets were recently suggested in application to deep networks. Apart from the standard use of coresets for reducing the amount of computations in training, e.g., by replacing full data [20] or a batch [21] with a coreset, there are other applications that motivate the use of the summarization methods in deep networks, e.g., model compression, continual learning, domain adaptation, federated learning, and neural architecture search (NAS). We discuss some of the them below.

Model Compression: Deep networks are highly overparametrized, resulting in high memory requirements and slower inference. While many methods have been developed for reducing the size of a previously trained network with no (or small) accuracy loss [22], [23], [24], [25], [26], [27], [28], [29], [30], most of them relied on heuristics, which performed well on known benchmarks, but diverged considerably from the behavior of the original network on specific subsets of input distribution [31]. Few previous works [32], [33], [34], [35] tried to resolve this problem by deriving a coreset for a fully connected or a convolutional layer with provable tradeoff between the compression rate and the approximation error for any future input. However, since these works construct a coreset for a layer, the full network compression is performed in a layer-by-layer fashion.

Limited Data Access: Problems such as continual/incremental learning [36], [37], [38], [39], [40], [41], domain adaptation [42], [43], and federated learning [44] do not have access to the full data (due to memory limitation or privacy issues) and only a small summary of it can be used. Coresets offer a natural solution for these problems.
NAS: Another important application that could benefit from coresets is NAS. Evaluating different architectures or a choice of parameters using a large dataset is extremely time-consuming. A representative, small summary of the training set could be used for a reliable approximation of full training, while greatly speeding up the search. Few recent works [45], [46] (inspired by the work of [47]) tried to learn a small synthetic set that summarizes the full set.

Previous attempts of summarizing a full training set with a small subset or a synthetic set showed a merit of using coresets in modern artificial intelligence (AI) (e.g., for training deep network). However, the summarization methods that they suggested were based on heuristics with no guarantees on the approximation error. Hence, it is not clear that the existing heuristics for data summarization could scale up to real-life problems. On the other hand, theoretical coresets that provably quantify the tradeoff between data reduction and information loss for an objective function of interest are mostly limited to simple, shallow models due to the challenges discussed below in Section I-A. From the theoretical perspective, it seems that we cannot have coresets for a reasonable neural network under the classic definition of the worst case query (e.g., see Theorem 6 [32]). In this article, we try to find a midway between these two paradigms.

A. Coreset Challenges

In many modern machine learning problems, obtaining non-trivial theoretical worst case guarantees is usually impossible (due to a high complexity of the target model, e.g., deep networks or since every point in the input set is important in the sense of high sensitivity [48]). Even for the simple problems, it may take years to design a coreset and prove its correctness for a specific problem at hand.

Another problem with the existing theoretical frameworks is the lack of generality. Even the most generic frameworks among them [49], [50] replace the problem of computing a coreset for n points with n new optimization problems (known as sensitivity bound), one for each of the n input points. Solving these, however, might be harder than solving the original problem. Hence, different approximation techniques are usually tailored for each and every problem.

B. Our Contribution

The above observations suggest that there is a need in a more generic approach that can compute a coreset automatically for a given pair of dataset and loss function and can be applied to hard problems, such as deep networks. It seems that this would require some relaxation in the standard coreset definition. Would such a relaxed coreset produced by a generic algorithm yield comparable empirical results with the traditional coresets that have provable guarantees? We affirmatively answer this question by providing:

1) A new definition of a coreset, which is a relaxation of the traditional definition of the strong coreset.

2) AutoCL: a generic and simple algorithm that is designed to compute a coreset (under the new definition) for almost any given input dataset and loss function.

3) Example applications with highly competitive empirical results for: (a) problems with known coreset construction algorithms, namely, linear regression, logistic regression, and SVMs where the goal is to summarize the input training set, and (b) model compression, i.e., learning a coreset of all the training parameters of a deep neural network (DNN) at once (useful for model pruning). To our knowledge, this is the first algorithm that aims to compute a coreset for the network at once, and not layer by layer or similar divide-and-conquer methods. It is also the first approach that suggests to represent the coreset itself as a small (trainable) network that keeps improving on each iteration. In this sense, we suggest “coreset for deep learning using deep learning.”

4) Open code for reproducing our results.1 We expect that it would be the baseline for producing “empirical” coresets for many problems in the future. Mainly, since it requires very little familiarity with the existing theoretical research on coresets.

II. Preliminaries

Notations: For a set P of n items, we use |P| to denote the number of items in P (i.e., |P| = n). For an event B, we use Pr(B) as the probability that event B occurs, and for a random variable x with a probability measure μ, we use Eμ(x) to denote its mean (expected value). Finally, for a loss function loss and an input set of variables C (from any form), we use Vloss(C) to denote a standard gradient computation of loss with respect to the set of variables C, and C − αVloss(C) to denote a standard variables update (C) using a gradient step, where α > 0 is the learning rate.

Remark: Throughout the literature, the term “coreset” usually refers to a small weighted subset of the input set (data). However, in other works (and in ours), this requirement is relaxed [18], [51]. In many applications, this relaxation gives a significant benefit as it supports a much larger family of instances as coreset candidates. It is also common to use the term “sketch” for the kind (relaxed definition) of coresets.

The following (generic) definition of a query space encapsulates all the ingredients required to formally define an optimization problem.

Definition 1 (Query Space; See Definition 4.2 in [52]): Let P be a (possibly infinite) set called ground set, Q′ be a (possibly infinite) set called query set, and let f : P × Q′ → [0, ∞) be a loss (or cost) function. Let P ⊆ P be a finite set called input set, and let w : P → [0, ∞) be a weight function. The tuple (P, w, Q′, f) is called a query space over P.

Typically, in the training step (of machine learning model), we solve the optimization problem, i.e., we aim at finding the solution q* that minimizes the sum of fitting errors ∑p∈P w(p)f(p, q) over every q ∈ Q′.

Definition 2 (Query Cost): Let (P, w, Q′, f) be a query space over P. Then, for a query q ∈ Q′ we define the total cost of q as f(P, w, q) = ∑p∈P w(p)f(p, q).
In the next definition, we describe formally a (strong) coreset for a given optimization problem.

**Definition 3 (Traditional Coresets):** For a query space \((P, w, Q', f)\), and an error parameter \(\varepsilon \in (0, 1)\), an \(\varepsilon\)-coreset is a pair \((C, u)\) such that \(C \subseteq P\), \(u : C \to \mathbb{R}\) is a weight function, and for every \(q \in Q'\), \(f(C, u, q)\) is a \(1 + \varepsilon\) multiplicative approximation for \(f(P, w, q)\), that is,

\[
|f(P, w, q) - f(C, u, q)| \leq \varepsilon f(P, w, q). \tag{1}
\]

### III. Method

In this section, we first explain our approach in general, emphasizing its novelty and then we present our suggested framework including all the details.

#### A. Novel Framework

We propose a practical and generic framework for coreset construction to a wide family of problems via the following steps.

1) **Make a problem simpler by relaxing the definition of a coreset.** Namely, we propose a new \((\varepsilon, \mu)\)-coreset for the average loss (in Definition 5) that is a relaxation of the standard definition (in Definition 3) and is more suited for the learning formalism.

2) **Define coreset construction as a learning problem.** Here, the coreset (under the new definition in Definition 5) is the training variable.

3) **Find the coreset that optimizes the empirical risk over a training set of queries.** We assume that we are given a set of queries, chosen i.i.d. from an unknown distribution and we find a coreset that approximates the average loss of the original input data over the training set of queries.

4) **Show that the optimized coreset generalizes to all the members in the query set.** Namely, the expected loss on the coreset over all queries approximates the expected loss on the original input data.

#### B. \((\varepsilon, \mu)\)-Coreset for the Average Loss

We relax the definition of a coreset by observing that in data mining and machine learning problems, we are usually interested in approximating the average loss over the whole set of queries rather than approximating the loss of a specific query. To this end, we define a distribution over the set of queries in Definition 4, and then focus on approximating the expected loss in Definition 5.

**Definition 4 (Measurable Query Space):** Let \((P, w, Q', f)\) be a query space over the ground set \(P\), and let \(\mu\) be a probability measure on a probability space \((Q', 2^{Q'})\). Then, the tuple \((P, w, Q', f, \mu)\) is called a measurable query space over \(P\).

**Definition 5 ((\(\varepsilon, \mu\))-Coreset for the Average Loss):** Let \((P, w, Q', f, \mu)\) be a measurable query space over \(P\). Let \(\varepsilon \in [0, \infty)\) be an error parameter, \(C \subset P\) a set, and \(u : C \to \mathbb{R}\) be a weight function such that

\[
|\mathbb{E}_\mu(f(P, w, q)) - \mathbb{E}_\mu(f(C, u, q))| \leq \varepsilon
\]

\(^{i.e.,}\) the expected loss of the original set \(P\) over the randomness of sampling a query \(q\) from the distribution \(\mu\) is approximated by the expected loss on \(C\).

Then, the pair \((C, u)\) is called an \((\varepsilon, \mu)\)-coreset for the measurable query space \((P, w, Q', f, \mu)\).

While \((P, w)\) is also an \((\varepsilon, \mu)\)-coreset of \((P, w, Q', f, \mu)\), coreset \((C, u)\) is efficient if the cardinality of \(C\) is significantly smaller than \(P\), i.e., \(|C| \ll |P|\), hopefully by order of magnitude.

#### C. Coreset Learning

We propose to learn a coreset (and its weights) as in Definition 5 using the gradient-based methods. We assume that we are given a set \(P\), its weights such that \(\sum_{p \in P} w(p) = 1\), and a set \(Q\) of \(|Q| = k\) queries sampled i.i.d. from \(Q'\) (according to the measure \(\mu\)). First, we aim to compute an \((\varepsilon, \mu)\)-coreset \((C, u)\) of \((P, w)\) with respect to the finite set of queries \(Q\). Formally speaking, \((C, u)\) should satisfy

\[
\left|\sum_{q \in Q} \frac{1}{k} f(P, w, q) - \sum_{q \in Q} \frac{1}{k} f(C, u, q)\right| \leq \varepsilon. \tag{2}
\]

To do so, we can treat \(Q\) as our training data and learn coreset \((C, u)\) of \((P, w)\) with respect to the objective \(f\) by minimizing the left-hand side of (2) as our loss.

This will guarantee that \((C, u)\) is an \((\varepsilon, \mu)\)-coreset for the measurable query space \((P, w, Q, f, \mu)\), where \(Q : Q \to [0, 1]\) is the uniform distribution over the finite set \(Q\), i.e., \(U(q) = 1/k = 1/|Q|\) for every \(q \in Q\).

However, we wish that the constraint in (2) would hold for the whole set of queries \(Q'\) to obtain an \((\varepsilon, \mu)\)-coreset for our desired (original) measurable query space \((P, w, Q', f, \mu)\).

To obtain a generalized solution (as we show in Section III-D), we need to bound \(\sup_{q \in Q'} f(C, u, q)\). To do so, we should guarantee that the sum of coreset weights approximates the original sum of weights, that is,

\[
\left|\sum_{p \in P} w(p) - \sum_{p \in C} u(p)\right| \leq \varepsilon. \tag{3}
\]

The motivation behind bounding (3) is as follows. Recall that \(\mathbb{P}\) is the ground set, i.e., \(P, C \subset \mathbb{P}\). Let \(M = \sup_{q \in Q'} f(p, q)\), so that enforcing (3) yields for every \(q \in Q'\): \(f(C, u, q) \leq \sum_{p \in C} w(p) f(p, q) \leq \left(\sum_{p \in \mathbb{P}} w(p) + \varepsilon\right) M = (1 + \varepsilon) M\). Hence, we “force” our coreset to have a bounded loss over the whole query space, \(\sup_{q \in Q'} f(C, u, q) \leq (1 + \varepsilon) M\); furthermore, this bound is proportional to the bound of the loss on the original input \(P\), i.e., it is proportional to

\[
\sup_{q \in Q'} f(P, w, q) \leq \sum_{p \in P} w(p) M = M
\]

and the approximation error \(\varepsilon\).

To summarize, we learn an \((\varepsilon, \mu)\)-coreset \((C, u)\) of \((P, w)\) with respect to the objective \(f\) given a training data (set of
Then, by (2), we have that 

\[ (\beta, \mu) \]

and 2 with (2) yields that

\[ \mu (|C| - \lambda) \sum_{p \in P} w(p) - \sum_{p \in C} u(p) \]

The approximation error that we wish to minimize, \( \lambda > 0 \) is a hyperparameter to balance the two losses.

7: \( C := C - \alpha \nabla \text{loss}(C) \)

(Update \( C \), where \( \alpha > 0 \) is the learning rate.)

8: \( u := \max\{0, u - \alpha \nabla \text{loss}(u)\} \) (Update \( u \).)

9: end for

10: return \((C, u)\)

Algorithm 1 AUTOCL\((P, w, Q, f, C_{size})\)

Input: A finite input set \( P \), and its weight function \( w : P \to \mathbb{R} \), a finite set of queries \( Q \), a loss function \( f : P \times Q \to [0, \infty) \), and an integer \( C_{size} \geq 1 \).

1: \( C := C_{size} \) is an arbitrary set of \( C_{size} \) vectors in \( P \).

2: \( u(c) := 1/|C_{size}| \) for every \( c \in C \).

3: for \( i := 1 \) to epochs do

4: \( f_C := \frac{1}{k} \sum_{q \in Q} f(C, u, q) \) {The average loss on \( C \).

5: \( f_P := \frac{1}{k} \sum_{q \in Q} f(P, w, q) \) {The average loss on \( P \).

6: \( \text{loss} := |f_P - f_C| + \lambda |\sum_{p \in P} w(p) - \sum_{p \in C} u(p)| \)

This claim states that with high probability, the average loss on the set \( P \) over the i.i.d. sampled set \( Q \) of \( k \) queries approximates the expected loss on the set \( P \) over all the queries in \( Q' \) [i.e., \( \mathbb{E}_\mu(f(P, w, q)) \)].

Now, recall that \( P \) is the ground set, i.e., \( P, C \subseteq P \), and \( M = \sup_{q \in Q'} |f(p, w, q)| \). As we formally show in Section B, since \( \varepsilon \) and \( \delta \) are fixed, and finally, the maximum loss over every \( q \in Q' \), i.e., \( \sup_{q \in Q'} |f(P, w, q)| \) (see Claim 1).

To show that \( \delta \) holds, we can also use Hoeffding’s inequality as follows.

**Claim 1 (Mean of Losses):** Let \( (P, w, Q', f, \mu) \) be a measurable query space such that \( \sum_{p \in P} w(p) = 1 \), and let \( M = \sup_{q \in Q'} |f(P, w, q)| \). Let \( \varepsilon \in (0, \infty) \) be an approximation error, and let \( \delta \in (0, 1) \) be a probability of failure. Let \( Q \) be a sample of \( k \geq (2M^2 \ln(2/\delta)/\varepsilon^2) \) queries from \( Q' \), chosen i.i.d., where each \( q \in Q' \) is sampled with probability \( \mu(q) \). Then, with probability at least \( 1 - \delta \)

\[
\mathbb{E}_\mu(f(P, w, q)) - \mathbb{E}_\mu(f(P, w, q)) \leq \varepsilon.
\]

Then, we obtain that with probability at least \( 1 - \delta \)

\[ \mathbb{E}_\mu(f(P, w, q)) - \mathbb{E}_\mu(f(C, u, q)) \leq 3\varepsilon. \]

**Proof:** See proof in Section C in the Appendix.
E. Bridging the Gap Between Theory and Practice

We take one more step toward deriving effective, practical coresets and replace the loss in 4 (and Line 6 in Algorithm 1) with a formulation that is more similar to the standard coreset definition, namely, \( \text{loss}(q; C, \mu) = |1 - (f(C, \mu, q)/f(P, \mu, q))| + \lambda |\sum_{p \epsilon P} w(p) - \sum_{p \epsilon C} u(p)| \) and we minimize this loss on average over the training set of queries \( Q \); see Algorithm 2 in the Appendix.

A solution obtained by Algorithm 2 aims to minimize the average approximation error over every query \( q \) in the sampled set \( Q \) and thus is very similar to Definition 3 with the modification of average instead of the worst case. This enables us to obtain a better coreset in practice that approximates the loss of every query \( q \) (as the minimization is on the average approximation error over all the queries and not only on the difference between the average losses of the coreset and the original data over all the queries). Our empirical evaluation in Section IV verifies that the coreset obtained by running Algorithm 2 generalizes to unseen queries, i.e., the average approximation error of the coreset over all the queries is small compared with other coreset construction algorithms. Moreover, we show below that the solution obtained by Algorithm 2 satisfies Definition 5.

Let \((C^*, \mu^*)\) be a solution that minimizes the average loss in Algorithm 2. We can find a constant \( \epsilon' > 0 \), such that

\[
\frac{1}{k} \sum_{q \in Q} \left| 1 - \frac{f(C^*, \mu^*, q)}{f(P, \mu, q)} \right| \leq \epsilon'.
\]

For a constant \( \epsilon \) from Definition 5, let \( M = \sup_{q \in Q} |f(P, \mu, q)| \), and let \( \epsilon = \epsilon'M \). By simple derivations (see Section C-A in the Appendix), we can show that

\[
\frac{1}{k} \sum_{q \in Q} f(P, \mu, q) - \frac{1}{k} \sum_{q \in Q} f(C^*, \mu^*, q) \leq \epsilon.
\]

Hence, by Claim 2 the solution obtained by Algorithm 2 generalizes to the whole measurable query space \((P, \mu, Q, f, \mu)\), and thus it satisfies the definition of \((\epsilon, \mu)\)-coreset, while simultaneously satisfying (5) which is closely related to the original definition of coresets as in Definition 3.

Remark: Observe that in the context of DNN, in most cases and especially in the classification tasks, the used loss function is (softmax) cross-entropy. In such cases, the loss function is always bounded between 0 and 1, i.e., \( 0 \leq f(p, q) \leq 1 \) for every \( p \in P \) and \( q \in Q \), which implies that the defined parameter \( M \) in 6 is actually 1, and thus \( \epsilon = \epsilon' \) is this case. Furthermore, in many machine learning known problem such assumption indeed holds, e.g., logistic regression. Finally, when we aim at solving (or suggesting an approximation to) some problem by computing a coreset, and the queries that we wish to approximate are close to the optimum which means that there is loss is indeed small and bounded.

IV. EXPERIMENTAL RESULTS

We proposed a unified framework for coreset construction that allows us to use the same algorithm for different problems. We demonstrate this on the examples of training set reduction for linear regression, logistic regression, and SVM in Section IV-A and on the examples of model size reduction a.k.a. model compression of multilayer perceptron (MLP) and convolutional neural network (CNN) in Section IV-B. We show that in both the cases our unified framework yields comparable or even better results than previous coresets, which are specifically fit to the problem at hand.

A. Training Data Coresets

We demonstrate the practical strength of our coreset construction scheme in the context of data reduction for linear regression, logistic regression, and SVMs.

1) Setup: For linear regression, we ran our experiments on the 3-D Road Networks datasets\(^3\) (North Jutland, Denmark)\(^5\) that contains 434 874 records. We used two attributes: “Longitude” [Double] and “Latitude” [Double] to predict the third attribute “Height in meters” [Double]. We created a set of queries by sampling models from training trajectories of linear regression computed using the full dataset from 20 random starting points. We split the sampled models into training, validation, and test sets of sizes 20 000, 20 000, and 2000 correspondingly. We computed coresets of different sizes, from 50 to 140. For each coreset size, we invoked Algorithm 2 with the Adam optimizer\(^5\) for ten epochs with a batch size of 25 and a learning rate of 0.01. The results were averaged across ten trials. In this experiments, we used \( \lambda = 1 \).

For logistic regression, we performed the experiments on the high time resolution universe (HTRU)\(^3\) dataset, comprising 17 898 radio emissions of the pulsar star represented by eight features and a binary label\(^5\). We created a set of queries similar to linear regression and we sampled from this set training, validation, and test sets of sizes 8000, 1600, and 800 correspondingly. The results were averaged across five trials. To make the optimization simpler, we removed the weight fitting term from the loss in Algorithm 2 and assumed that all the members of the coreset have the same weight \( 1/|C| \). We ran the optimization for 1000 epochs with the batch size of 100 using the Adam optimizer and a learning rate of 0.001. Using this modification, we computed coresets of different sizes ranging from 100 to 500.

Similar to logistic regression, in the context of SVM we performed the experiments on the HTRU dataset. We adapt the settings of\(^6\) and use 16 107 out of 17 898 as data points. We created a set of queries in the same fashion as logistic regression, and we then sampled from this set training, validation, and test sets of sizes 10 000, 2000, and 1000 correspondingly. The results were averaged across five trials. We ran the optimization for 200 epochs with the batch size of 100 using the Adam optimizer and a learning rate of 0.001. Using this, we computed coresets of different sizes ranging from 800 to 1800 (jumps of 200).

The differences in hyperparameters and the coreset sizes between, e.g., the logistic and linear regression experiments

\(^3\)https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+(North+Jutland,+Denmark)

\(^5\)https://archive.ics.uci.edu/ml/datasets/HTRU2
are due to the higher complexity of the problem for logistic regression. First, computing a coreset for logistic regression is known to be a complex problem where (high) lower bounds on the coreset size exist [2]. The second (and probably less significant) reason is the dimension of the input data, where we used a higher dimensional input in logistic regression.

2) Results: We refer to a weighted labeled input dataset by \((P, w, b)\), where \(P\) is the dataset, \(b : P \rightarrow \mathbb{R}\) and \(w : P \rightarrow [0, \infty)\) are the labeling function and weight function, respectively, i.e., each point \(p\) in \(P\) is a sample in the dataset, \(b(p)\) is its label, and \(w(p)\) is its weight. Similarly, we refer to the compressed labeled dataset (coreset) by \((C, u, y)\). We report the results using the two measures as explained below.

1) Approximation Error for the Optimal Solution: Let \(q^*\) be the query that minimizes the corresponding objective loss function, e.g., in linear regression:  
\[
q^* \in \arg\min_{q \in \mathbb{R}^d} f(P, w, b, q) = \sum_{p \in P} w(p)(p^T q - b(p))^2.
\]
For each coreset \((C, u, y)\), we compute \(q^*_C \in \arg\min_{q \in \mathbb{R}^d} f(C, u, y, q)\), and then we calculate the approximation error for the optimal solution as  
\[
Err_{opt} = |1 - \frac{f(P, w, b, q^*)}{f(P, w, b, q^*_C)}|.
\]

2) Average Approximation Error: For every coreset \((C, u, y)\), we report the average case approximation error over every query \(q\) in the test set \(Q_{test}\), i.e.,  
\[
Err_{avg} = \frac{1}{|Q_{test}|} \sum_{q \in Q_{test}} |1 - \frac{f(C, u, y, q)}{f(P, w, b, q)}|.
\]
We compare our coresets for linear regression with uniform sampling and with the coreset from [6]. Error of the three methods is shown in Fig 1(a) and Error in Fig 1(b).

We compare our coreset for both logistic regression and SVM with uniform sampling and with the coreset from [6]. For logistic regression, \(Err_{opt}\) of the compared methods is shown in Fig 2(a) and \(Err_{avg}\) in Fig 2(b). For SVM, \(Err_{opt}\) is reported in Fig 3(a) and \(Err_{avg}\) in Fig 3(b).

In all the experiments, we observe that our learned coresets outperform uniform sampling and the theoretical counterparts. Our method yields very low average approximation error, because it was explicitly trained to derive a coreset that minimizes the average approximation error on the training set of queries, and the learned coreset succeeded to generalize to unseen queries.

3) Coreset Visualization: In this section, we visualize our computed coresets from Section IV-A2. We plot the original high-dimensional data and its coreset in a 3-D by projecting them onto the subspace of highest variance [obtained by principal component analysis (PCA)]. For every experiment (linear regression, logistic regression, and SVM), we plot three figures visualizing three different coreset sizes; see Figs 4–6.

B. Model Coreset for Structured Pruning

The goal of model compression is reducing the run time and the memory requirements during inference with no or little accuracy loss compared with the original model. Structured pruning reduces the size of a large trained deep network by reducing the width of the layers (pruning neurons in fully connected layers and filters in convolutional layers). An alternative approach is sparsification, which zeros out unimportant parameters in a deep network. The main drawback of sparsification is that it leads to an irregular network.
structure, which needs a special treatment to deal with sparse representations, making it hard to achieve actual computational savings. Structured pruning simply reduces the size of the tensors, which allows running the resulting network without any amendment. Due to the advantage of structured pruning over sparsification, we perform structured pruning of a deep networks in our experiments.

In general, model compression is divided into two steps: 1) determining a more compact architecture and 2) updating the network parameters to approximate the original model.
Some model compression approaches look for a reduced architecture and weight approximation (both the steps) at the same time. Others (including ours) decouple the architectural search from network approximation. Finding the compact architecture can be done using NAS (a significant amount of research has been conducted on this topic). The second step of finding the weights of the miniature model could be done in different ways.

1) Train the compact architecture from the ground up.
2) Keep the top K weights/filters (determined by the compact model) and fine-tune the network.
3) Determine the weights of the small architecture that best approximates the original network’s output, and then fine-tune the weights (if necessary).

We are following the latter approach here. The key advantage is that a well-approximated network is close to the original one in terms of accuracy and hence requires substantially less fine-tuning, if any at all [e.g., visual geometry group (VGG)19 on Canadian Institute for Advanced Research (CIFAR)10 required no fine-tuning].

To this end, we assume that the target small architecture is given, and our task is to compute the training parameters of the small architecture that best approximate the original large network. We view filters in CNN or neurons in a fully connected network as items in the full set $P$, and the training data as the query set $Q$. We use the small architecture to define the coreset size in each layer and we learn an equally weighted coreset $C$ (the small network) using Algorithm 2 and setting $\lambda = 0$. We report the experiments for structured pruning of a fully connected network in Section IV-B1 and of channel pruning in Section IV-B2.

In all the experiments, we present the compression ratio, which is the percentage of parameters deleted from the original model, as well as the error, which is the percentage of misclassified instances from the test set. We compared our method with other pruning methods including coreset-based pruning techniques. We report the reduction in percentage of floating point operations (FLOPs) for the pruned model obtained by our method and for baselines if published. In our experiment, we only compressed and fine-tuned the network once; no repeated/iterative pruning was used. In some (reported) cases, we (even) do not fine-tune the network.

1) Neuron Pruning: We used the LeNet$–300–100$ model with 266,610 parameters trained on Modified National Institute of Standards and Technology dataset (MNIST) [59] as our baseline fully connected model. It comprises two fully connected hidden layers with 300 and 100 neurons correspondingly, each followed with an ReLu activation. After training the baseline model with the Adam optimizer for 40 epochs and a batch size of 64, it achieved a test accuracy of 97.93% and loss $= 0.0917$. The target small architecture included 30 neurons in the first layer and 100 in the second, resulting in 89.63% compression ratio. We applied the training procedure in Algorithm 2 to learn the weights of this network using the Adam optimizer with the $L_2$ regularization for 400 epochs with the batch size of 500.

Results: The coreset (compressed) model achieved 97.97% accuracy and 0.0911 loss on the test data, i.e., improvement in both the terms. Next, we compare our results to a pair of other coreset-based compression methods in Table I and to noncoreset methods: filter thresholding (FT) [60], SoftNet [61], and ThiNet [62] implemented in [35]. We observe that the learned coreset performs better than most compared methods and comparatively to the algorithm derived from the theoretical coreset framework. Note that previous coreset methods [32], [35] are designed for a single layer, while our algorithm does not have this limitation and can be applied to compress all the layers of the network in a single run. Moreover, applied to DNN compression, our framework can work on individual weights (sparcification), neurons (as shown above), and channels (as we show next).

2) Channel Pruning: We compressed the convolutional layers of: 1) VGG19 [63]; 2) VGG16 [63] on CIFAR10; and 3) ResNet50 [64] on imagenet large scale visual recognition challenge (ILSVRC)2012 [65].

VGG Architectures Setup: We used PyTorch implementation of the visual geometry group network (VGGNet)-19 and VGGNet-16 networks for CIFAR10 from [66]. Our VGG-19 baseline model had about 20M parameters, while the VGG-16 baseline model consists of roughly 14.7M parameters. The VGG-19 and VGG-16 baselines' accuracy was 93.25% and 92.89%, respectively.

The target architecture of VGG-19$^9$ of the small network (see Table I in the supplementary material) corresponds to a reduction in the parameters by roughly 88%. We ran Algorithm 2 to learn the small architecture for 180 epochs with a batch size of 500 using the Adam optimizer and $L_2$ regularization. In VGG-16, the target architecture of the small network (see Table II in the supplementary material) corresponds to 95.2% reduction in the parameters. We ran Algorithm 2 to learn the small model for 160 epochs with a batch size of 64 using the stochastic gradient descent (SGD) optimizer.

VGG Architectures Results: In the context of VGG-19, our compressed model improved the baseline network and achieved 93.51% accuracy and 0.32 loss. Table III shows

---

9https://github.com/Eric-mingjie/network-slimming/blob/master/models/vgg.py
10https://github.com/foolwood/pytorch-slimming

---

### Table I: Neural Pruning of LeNet-300-100 for MNIST

| Pruning Method   | Baseline Error(%) | Small Model Error(%) | Compression Ratio(%) | FLOPs Compression Ratio(%) |
|------------------|-------------------|----------------------|----------------------|--------------------------|
| FT [60]          | 1.59              | 1.94                 | 81.68                | NA                       |
| SoftNet [61]     | 1.59              | 2.02                 | 81.69                | NA                       |
| ThiNet [62]      | 1.59              | 1.17                 | 75.01                | NA                       |
| Sample-based Coreset [35] | 1.59 | 2.03 | 84.32 | NA |
| Pruning via Coreset [33] | 2.16 | 2.03 | ~ 90 | NA |
| Learned Coreset [ours] | 2.07 | 2.02 | 89.63 | 89.63 |

### Table II: Channel Pruning of VGG-16 for CIFAR-10

| Pruning Method   | Baseline Error(%) | Small Model Error(%) | Compression Ratio(%) | FLOPs Compression Ratio(%) |
|------------------|-------------------|----------------------|----------------------|--------------------------|
| SoftNet [61]     | 7.11              | 3.92                 | 64.35                | 64.34                    |
| ThiNet [62]      | 7.11              | 2.94                 | 63.95                | 64.02                    |
| Sample-based Coreset [35] | 7.11 | 7.61 | 94.32 | 85.03 |
| FT [60]          | 7.11              | 8.22                 | 80.09                | 80.14                    |
| Learned Coreset [ours] | 7.11 | 7.2 | 95.22 | 85.09 |
comparison of the small network accuracy of the newly suggested learned coreset with the channel pruning coreset from [33] and several noncoreset methods. While the results are comparable, our algorithm is much simpler and is not tailored to the problem at hand. The coreset reported in [33] was constructed by applying a channel pruning coreset in a layer-by-layer fashion, while our learned coreset is computed in one-shot for the entire network. On VGG-16, as Table II reports, we compressed 95.22 from the baseline model while achieving a nearly zero drop in accuracy, improving upon the newly suggested coreset result of [35] in all the aspects, i.e., compression rate, FLOP compression rate, and accuracy.

Resnet-50 on ImageNet Results and Setup: Here, we compressed the ResNet50 [64] baseline model roughly by 62% (reduction in the number of parameters). The base model includes 50 convolutional layers with batch normalization and a dense layer with 2048 neurons. Table IV provides comparison between our method and other baselines, where we are either better than the competing methods or achieve comparable results. Finally, we remind the reader that our framework is generic and could be applied to many other applications in addition to compressing DNNs.

3) Adversarial Distribution Experiment: While the compressed networks are usually tested on a test set, sampled from the same distribution as the training set, there is a concern that the compressed network would deteriorate from the original network on out-of-distribution inputs. This is more likely to happen in networks compressed using heuristic approaches. When using a coreset with provable guarantees on the approximation error for any input, we expect the gap in the performance of the original and the compressed network to be much smaller, as shown in Fig 7.

We proposed a novel unified framework for coreset learning that is theoretically motivated and can address problems for which obtaining theoretical worst case guarantees is impossible. Following this framework, we suggested a relaxation of the coreset definition from the worst case to the average loss approximation. We proposed a learning algorithm that inputs a sample set of queries and a loss function associated with the problem at hand and outputs an average-loss coreset that holds for the training set of queries and generalizes to unseen queries. We showed that if the sample set of queries is sufficiently large, then the average loss over the coreset closely approximates the average loss over the full set for the entire query space. We then showed empirically that our learned coresets are capable to generalize to unseen queries even for arbitrary sampling sizes. Our experiments demonstrated that coresets learned by our approach yielded comparable or better approximation of the optimal solution loss and average loss over the unseen queries than coresets that have worst case guarantees. Moreover, our method applied to the problem of deep networks pruning provides the first full-network coreset with excellent performance.

V. CONCLUSION

As our approach is a learning-based method, its construction time is slower than that of the nonlearning (classical) algorithms for data reduction. However, our algorithm is generic, easily implementable, and produces smaller coresets for the same (or even) better approximation error as

from [66]. We gradually moved the input distribution away from the original one by increasing the magnitude of the adversarial noise in small steps. Obviously, all the tested networks showed vulnerability to adversarial inputs, as no protection was included. However, in this test we are more interested in the accuracy gap with the original network, rather than adversarial robustness. Our learned-coreset-based pruning is closer to the original network even better at the beginning and retains the same degradation level (where there is a degradation), which corresponds to the approximation error bound, as shown in Fig 7. Similarly, the coreset-based method of [33] is close to us and to the original network, but it is not as robust as our approach. The performance of the technique from [66] is the same for the initial distribution of inputs, but it degrades significantly quicker with each distribution change and eventually sinks to the chance level.

| Pruning Method | Baseline Error(%) | Small Model Error(%) | Compression Ratio(%) | FLOP Compression Ratio(%) |
|---------------|------------------|---------------------|----------------------|--------------------------|
| Unstructured Pruning [67] | 6.3 | 6.48 | 80 | NA |
| Structured Pruning [68] | 6.33 | 6.32 | 88 | NA |
| Pruning via Coresets [33] | 6.33 | 6.43 | 88 | NA |
| Learned Coreset (ours) | 6.75 | 6.43 | 88.2 | 48.02 |

| Pruning Method | Baseline Error(%) | Small Model Error(%) | Compression Ratio(%) | FLOP Compression Ratio(%) |
|---------------|------------------|---------------------|----------------------|--------------------------|
| ResNet-50 [62] | 24.72 | 28.97 | 33.72 | 46.78 |
| ResNet-50 [63] | 23.74 | 28.8 | 34.74 | 35.82 |
| SetNet [61] | 23.85 | 25.39 | 49.35 | 51.36 |
| CCP [68] | 23.85 | 25.16 | 46 | 58.8 |
| POM [35] | 23.85 | 25.11 | 41.9 | 55.3 |
| Sample-Based Coreset [33] | 23.77 | 25.79 | 44.04 | 50.05 |
| Pruning via Coresets [33] | 23.75 | 25.11 | 62 | NA |
| Learned coreset (ours) | 23.78 | 25.09 | 62 | 61.5 |

Fig. 7. Adversarial examples’ experiment.
Appendix A
Hoeffding Theorem

Theorem 1 (Hoeffding): Let $X_1, \ldots, X_k$ be independent random variables, when it is known that for every $i \in k$, $X_i$ is strictly bounded by the intervals $[a_i, b_i]$. Define the empirical mean of these variables by $\mu = \frac{1}{k} \sum_{i=1}^{k} (1/k)X_i$, then

$$\Pr( |\mu - E(\mu)| \geq \varepsilon ) \leq 2 e^{-2\varepsilon^2 / \sum_{i=1}^{k}(b_i - a_i)^2}.$$ 

Appendix B
Proof of Claim 1

Proof: First, observe that: 1) the probability distribution $\mu$ is defined over the set $Q'$ and 2) the function $f$ in our case is a function of $q \in Q'$, since $P$ and $w$ are fixed (given). Thus, we can define the corresponding probability distribution $\mu'$ for the (multi)-set $F' = \{f(P, w, q) \mid q \in Q'\}$ as follows: For every $x = f(P, w, q) \in F'$ (where $q \in Q'$), we have that $\mu'(x) = \mu'(f(P, w, q)) = \mu(q)$. Moreover, the sampled set $Q$ has its corresponding sampled loss set $F = \{f(P, w, q) \mid q \in Q\}$. Hence, we have that

$$\Pr \left( \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \mathbb{E}_\mu(f(P, w, q)) \leq \varepsilon \right) = 1 - \Pr \left( \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \mathbb{E}_\mu(f(P, w, q)) \geq \varepsilon \right).$$

By applying Hoeffding’s inequality (see Theorem 1 in the Appendix), we have

$$1 - \Pr \left( \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \mathbb{E}_\mu(f(P, w, q)) \geq \varepsilon \right) \geq 1 - 2 e^{-2\varepsilon^2 \sum_{q \in Q} (w_q)^2} = 1 - 2 e^{-2\varepsilon^2 |\mu|/4M^2}.$$ 

where $a_i$ and $b_i$ are the lower and upper bounds on the loss of the $i$th sampled query, respectively. Since, by the definition of $M$ we have that for every $q \in Q'$, $-M \leq f(P, w, q) \leq M$, we obtain that

$$1 - 2 e^{-2\varepsilon^2 \sum_{q \in Q} (w_q)^2} = 1 - 2 e^{-2\varepsilon^2 \frac{\sum_{i=1}^{k}(b_i - a_i)^2}{2\sum_{q \in Q} (w_q)^2}} = 1 - 2 e^{-2\varepsilon^2 |\mu|/4M^2}.$$ 

Finally, combining (8), (9), and (10) proves the claim as

$$\Pr \left( \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \mathbb{E}_\mu(f(P, w, q)) \leq \varepsilon \right) \geq 1 - \delta.$$ 

Appendix C
Proof of Claim 2

Proof: Let $M_1 = \sup_{q \in Q} f(P, w, q)$, and let $M_2 = \sup_{q \in Q} f(C, u, q)$. First, we observe that

$$M_1 = \sup_{q \in Q} f(P, w, q) = \sup_{q \in Q} \sum_{p \in P} w(p)f(p, q) \leq \sum_{p \in P} w(p)M = M$$

where the third derivation holds by the definition of $M$, and the fourth holds since $\sum_{p \in P} w(p) = 1$. We also have

$$M_2 = \sup_{q \in Q} f(C, u, q) = \sup_{q \in Q} \sum_{p \in C} u(p)f(p, q) \leq \sum_{p \in C} u(p)M \leq (1 + \varepsilon)M$$

where the third inequality holds by the definition of $M$, the fourth by Assumption 1, and the last holds since $\sum_{p \in P} w(p) = 1$. By combining (11) and (12), we get that $(1 + \varepsilon)M \leq M_1, M_2$. Now, we note that Claim 1 holds for any measurable query space. Hence, for the pair of measurable query spaces $(P, w, Q', f, \mu)$ and $(C, u, Q', f, \mu)$, if the sampled set $Q \subset Q'$ satisfies that $k = |Q| \geq (2(1 + \varepsilon)M^2 \ln(2/\delta)/\varepsilon^2)$, then by Claim 1 we get that

$$\left| \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \mathbb{E}_\mu(f(P, w, q)) \right| < \varepsilon,$$

and

$$\left| \frac{1}{k} \sum_{q \in Q} f(C, u, q) - \mathbb{E}_\mu(f(C, u, q)) \right| < \varepsilon.$$ 

By the triangle inequality, we have that

$$|\mathbb{E}_\mu(f(P, w, q)) - \mathbb{E}_\mu(f(C, u, q))| \leq \left| \mathbb{E}_\mu(f(P, w, q)) - \frac{1}{k} \sum_{q \in Q} f(P, w, q) \right| + \left| \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \frac{1}{k} \sum_{q \in Q} f(C, u, q) \right|.$$
Algorithm 2 PRACTICAL-AUTO CL(P, w, Q, f, Csize)

Input: A finite set P, its weight function w, a finite set of queries Q, a loss function \( f : P \times Q \rightarrow [0, \infty) \), and an integer Csize \( \geq 1 \).

1: \( C := \{c_{i}\}_{i=1}^{Csize} \) is an arbitrary set of Csize vectors in \( \mathbb{P} \).
2: \( u(c) := 1/Csize \) for every \( c \in C \).
3: for \( i \in \{1, \ldots, \text{epochs}\} \) do
4: for every \( q \in Q \) do
5: \( f_c := f(C, u, q) \)
   \{The cost of the query \( q \) on \( C \)\}
6: \( f_P := f(P, w, q) \)
   \{The cost of the query \( q \) on \( P \)\}
7: \( \text{loss} := |1 - \frac{f_c}{f_P}| + \lambda \sum_{p \in P} w(p) - \sum_{p \in C} u(p) \)
   \{The approximation error that we wish to minimize\}
   \{\( \alpha \) is the learning rate.\}
8: \( C := C - \alpha \text{Vloss}(C) \) \{Update \( C \)\}
9: \( u := \max\{0, u - \alpha \nabla \text{loss}(u)\} \) \{Update \( u \)\}.
10: end for
11: end for
12: return \((C, u)\)

\[ + \frac{1}{k} \sum_{q \in Q} f(C, u, q) - \mathbb{E}_u f(C, u, q). \] (17)

Using (13) and (14), and by the assumption (2) on the output
\( C, u \), we have that (15), (16), and (17) are bounded by \( \epsilon \). Hence
\[ |\mathbb{E}_u f(P, w, q) - \mathbb{E}_u f(C, u, q)| \leq 3\epsilon. \]

A. Proof of Equation 6

For a constant \( \epsilon \) from Definition 5, let \( M = \sup_{q \in Q} |f(P, w, q)| \), and let \( \epsilon = \epsilon' M \). We show that
\[ \left| \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \frac{1}{k} \sum_{q \in Q} f(C^*, u^*, q) \right| \leq \epsilon. \] (18)

Proof:
\[ \left| \frac{1}{k} \sum_{q \in Q} f(P, w, q) - \frac{1}{k} \sum_{q \in Q} f(C^*, u^*, q) \right| \]
\[ \leq \frac{1}{k} \sum_{q \in Q} |f(P, w, q) - f(C^*, u^*, q)| \]
\[ = \frac{M}{k} \sum_{q \in Q} |f(P, w, q) - f(C^*, u^*, q)| \]
\[ \leq \frac{M}{k} \sum_{q \in Q} \frac{1}{k} |f(P, w, q) - f(C^*, u^*, q)| \]
\[ = \frac{M}{k} \sum_{q \in Q} \frac{1}{k} = \frac{M}{k} = M \epsilon' = \epsilon \]

where the first derivation holds since \( |\sum_{i=1}^{k} a_i| \leq \sum_{i=1}^{k} |a_i| \) for any set of number \( \{a_i\}_{i=1}^{k} \), the derivation in (6) follows

from the definition of M, i.e., since \( M > \int |f(P, w, q)| \) for every \( q \in Q \), the one after holds since \( |a||b| = \{|ab|\} \) for any pair \( a, b \in \mathbb{R} \), and the last derivation holds by (5).

APPENDIX D

PRACTICAL IMPLEMENTATION

While the training of Algorithm 2 is formalized as a stochastic process, i.e., sequentially, for every \( q \in Q \), we compute the approximation error \( |1 - (f(C, u, q)/f(P, w, q))| \) for this one query \( q \), and we then update the learned variables based on this error. However, it can be implemented using a minibatch of several queries \( Q \subseteq Q \). Here, the approximation error with respect to the current batch \( Q \) is \( \sum_{q \in Q} |1 - (f(C, u, q)/f(P, w, q))| \) and the learned variables are updated based on this error.

REFERENCES

[1] J. Huggins, T. Campbell, and T. Broderick, “Coresets for scalable Bayesian logistic regression,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 4080–4088.
[2] A. Munteanu, C. Schwiegelshohn, C. Sohler, and D. Woodruff, “On coresets for logistic regression,” in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 6561–6570.
[3] S. Har-Peled, D. Roth, and D. Zimak, “Maximum margin coresets for active and noise tolerant learning,” in Proc. IJCAI, 2007, pp. 836–841.
[4] I. W.-H. Tsang, J.-Y. Kwok, and J.-M. Zurada, “Generalized core vector machines,” IEEE Trans. Neural Netw., vol. 17, no. 5, pp. 1126–1140, Sep. 2006.
[5] I. W. Tsang, J.-T. Kwok, and P.-M. Cheung, “Core vector machines: Fast SVM training on very large data sets,” J. Mach. Learn. Res., vol. 6, pp. 363–392, Apr. 2005.
[6] M. Tukan, A. Maalouf, and D. Feldman, “Coresets for near-convex functions,” in Proc. Adv. Neural Inf. Process. Syst., 2020, pp. 997–1009.
[7] I. Jubran, M. Tukan, A. Maalouf, and D. Feldman, “Sets clustering,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 4994–5005.
[8] M. Lucic, O. Bachem, and A. Krause, “Strong coresets for hard and soft Bregman clustering with applications to exponential family mixtures,” in Proc. 19th Int. Conf. Artif. Intell. Statist., in Proceedings of Machine Learning Research, vol. 51, A. Gretton and C. C. Robert, Eds, Cadiz, Spain, May 2016, pp. 1–9. [Online]. Available: http://proceedings.mlr.press/v51/lucic16.html
[9] M. Schmidt, C. Schwiegelshohn, and C. Sohler, “Fair coresets and streaming algorithms for fair k-means,” in Proc. Int. Workshop Approximation Online Algorithms. Cham, Switzerland: Springer, 2019, pp. 232–251.
[10] D. Feldman, M. Schmidt, and C. Sohler, “Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering,” in Proc. 24th Annu. ACM-SIAM Symp. Discrete Algorithms, Jan. 2013, pp. 1434–1453.
[11] A. Maalouf, I. Jubran, and D. Feldman, “Fast and accurate least-mean-squares solvers,” in Proc. Adv. Neural Inf. Process. Syst., 2019, pp. 8305–8316.
[12] A. Maalouf, A. Statman, and D. Feldman, “Tight sensitivity bounds for smaller coresets,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, Aug. 2020, pp. 2051–2061.
[13] T. Sarlos, “Improved approximation algorithms for large matrices via random projections,” in Proc. 47th Annu. IEEE Symp. Found. Comput. Sci. (FOCS), Oct. 2006, pp. 143–152.
[14] M. B. Cohen and R. Peng, “Lq row sampling by Lewis weights,” in Proc. 47th Annu. ACM Symp. Theory Comput., Jun. 2015, pp. 183–192.
[15] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney, “Sampling algorithms and coresets for \( l_{1}\) regression,” SIAM J. Comput., vol. 38, no. 5, pp. 2060–2078, 2009.
[16] C. Sohler and D. P. Woodruff, “Subspace embeddings for the \( L_1\) norm with applications,” in Proc. 43rd Annu. ACM Symp. Theory Comput. (STOC), 2011, pp. 755–764.
[17] D. Feldman, “Introduction to core-sets: An updated survey,” 2020, arXiv:2011.09384.
[18] J. M. Phillips, “Coresets and sketches,” 2016, arXiv:1601.00617.
[19] I. Jubran, A. Maalouf, and D. Feldman, “Introduction to core-sets: Accurate coresets,” 2019, arXiv:1910.08707.
Alaa Maalouf received the M.Sc. and Ph.D. degrees in computer science from the University of Haifa, Haifa, Israel, in 2019 and 2022, respectively. Soon, he will be starting his Postdoc with MIT, Cambridge, MA, USA. His research interests are machine/deep learning, robotics, and big data.

Gilad Eini received the B.Sc. and M.Sc. degrees in computer science from the University of Haifa, Haifa, Israel, in 2017 and 2021, respectively. His research interests are machine/deep learning and big data.

Ben Mussay received the B.Sc. and M.Sc. degrees in computer science from the University of Haifa, Haifa, Israel, in 2019 and 2020, respectively. His research interests are sublinear algorithms and deep learning.

Dan Feldman received the Ph.D. degree from the University of Tel-Aviv, Tel Aviv, Israel, in 2010. During his Ph.D. degree with the University of Tel-Aviv, he developed data reduction techniques based on computational geometry known as coresets. He is an Associate Professor and the Head of the Robotics and Big Data Laboratory, University of Haifa, Haifa, Israel, after returning from a three years Postdoc with Caltech, Pasadena, CA, USA, and MIT, Cambridge, MA, USA. Since then, coresets are applied for main problems in machine learning, computer vision, electroencephalogram (EEG), and robotics. His group in Haifa continues to design and implement provable coresets for real-time systems.

Margarita Osadchy received the Ph.D. degree (Hons.) in computer science from the University of Haifa, Haifa, Israel, in 2002. She is an Associate Professor with the Department of Computer Science, a member of the Data Science Research Center, and a member of the Scientific Committee of the Center for Cyber Law and Policy with the University of Haifa. She was a Visiting Research Scientist with the NEC Research Institute, Princeton NJ, USA, and then a Post-Doctoral Fellow with the Department of Computer Science, Technion, Haifa. Her research interests are deep/machine learning, computer vision, and computer security and privacy.