CryptoML: Secure Outsourcing of Big Data Machine Learning Applications

ABSTRACT

We present CryptoML, the first practical framework for provably secure and efficient delegation of a wide range of contemporary matrix-based machine learning (ML) applications on massive datasets. In CryptoML a delegating client with memory and computational resource constraints wishes to assign the storage and ML-related computations to the cloud servers, while preserving the privacy of its data. We first suggest the dominant components of delegation performance cost, and create a matrix sketching technique that aims at minimizing the cost by data pre-processing. We then propose a novel interactive delegation protocol based on the provably secure Shamir's secret sharing. The protocol is customized for our new sketching technique to maximize the client's resource efficiency. CryptoML shows a new trade-off between the efficiency of secure delegation and the accuracy of the ML task. Proof of concept evaluations corroborate applicability of CryptoML to datasets with billions of non-zero records.

1. INTRODUCTION

Machine learning (ML) is an indispensable tool for extracting knowledge from contemporary massive datasets. While complex ML algorithms can serve a significant number of applications, it is well-known that they incur expensive matrix-based storing and computing as well as iterative optimization costs. Examples of such costly ML approaches are the widely popular decent-based algorithms such as deep learning [7], Ridge, Elastic Nets, and Lasso [20], power method [11], and support vector machines [10]. For resource-limited clients, an increasingly popular practice is to delegate the required storage and computations of ML algorithms to cloud server providers (CSPs). This practice raises critical privacy concerns, since ML applications typically involve sensitive personal, medical, and financial records.

We introduce CryptoML, a new framework for secure delegation of generic iterative, matrix-based ML algorithms from resource-constrained clients to CSPs. For instance, consider a scenario where a health clinic wants to utilize the cloud power to detect and predict tumor propagation trends among patients by holistically analyzing their private diagnostic images. Such problems require complex iterative ML updates on large data dependency matrices. While secure task delegation has been an active research subject in recent years, privacy preserving ML with provable security has only been addressed for specific applications such as classification [5], Ridge regression [18], genome mapping [6], and/or for clients capable of performing costly cryptographic protocols such as fully homomorphic encryption (FHE) [13], and Garbled Circuits (GC) [21]. Real implementation of such approaches to datasets with billions of records has not been feasible. More generic secure ML delegation methods such as [15] are not scalable to large datasets.

CryptoML's delegation is designed to minimize the client's incurred cost, both in terms of its local and remote used resources while preserving the privacy of the data and the solution to ML problem. We exploit the fact that many ML applications are tolerant to output solution variations and suggest creating an approximate compact representation of data a.k.a., a *sketch matrix*. Our sketching algorithm is designed to greatly reduce the client's delegation cost. Forming the sketch is a precursor to the execution of the ML algorithm. The (pre-processing) cost of building and updating the sketch is readily amortized over several runs of the iterative ML updates.

The CryptoML delegation of the client's data and computing to the cloud is provably secure in the classic honest but curious security model, guaranteeing privacy of data and parameters of the ML algorithm as well as the results. The approach is practically efficient and does not rely on expensive cryptographic primitives such as FHE, GC or even public-key-cryptography. The building block of our novel delegation protocol is Shamir's secret sharing (SS) [8].

To the best of our knowledge, CryptoML is the first to propose a systematic and provably secure approximate computing approach to enable delegation of generic iterative ML tasks on large data. Our explicit contributions are as follows.

The first framework for scalable privacy-preserving delegation of a broad-range of complex ML algorithms with provable security (CryptoML): In our model, a resource-limited device delegates storage and computation of the ML algorithm that entails iterative multiplications on massive and often non-sparse matrices. On the client side, the execution overhead is constant, while on the CSPs side, the complexity is the same as the best known scalable method for solving the ML problem on a single machine without security. CryptoML's delegation can be done with the lowest possible number of CSPs, where the communication between the client and each CSP is provably minimal.

A novel Delegation-optimized Sparse Sketching (DSS): Our sketching approach is scalable and practical for resource-limited clients. This sketching is a precursor for ML training phase and its overhead is readily amortized over the intensive runs of the iterative ML method. The sketch is formed as an ensemble of lower dimensional subspaces with sparsified inter-subspace connections. The aim of DSS is to minimize the overall delegation cost in terms of stored and communicated words, as well as floating point operations (FLOPs). As such, the proposed sketching can be utilized outside of our framework for other privacy-preserving protocols including but not limited to GC or FHE.

Proof of concept evaluations of CryptoML on massive datasets as well as quantitative performance analysis: We evaluate CryptoML on various datasets for the Principle Component Analysis (PCA) application – the building block for several learning applications. CryptoML introduces a new trade-off between the efficiency and desirable accuracy of the ML algorithm which is of interest in contemporary approximate computing paradigms. To our knowledge, our evaluations are the first to show provably secure delegation of sophisticated iterative ML algorithms on massive content with billions of records.

2. CryptoML'S OVERVIEW

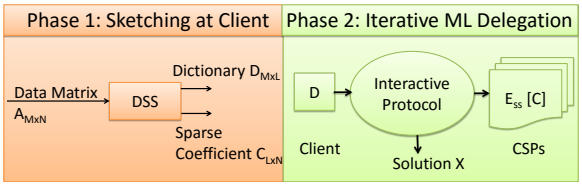

Figure 1: Overview of CryptoML.

The overall flow of CryptoML is shown in Figure 1. We denote the large $M \times N$ matrix on which we run the ML algorithm as A, where M is the dimension of a signal and N is the total number of samples being processed, which can be arbitrarily large. CryptoML consists of two phases: pre-processing and execution. During the pre-processing phase, the client applies DSS, our novel resource-efficient sketching approach to transform A to a compact *dictionary matrix* D and a *sparse coefficient matrix* C. The sketching can be customized to minimize the target delegation performance costs. The client keeps D locally and delegates encrypted shares of C, i.e., $E_{ss}(C)$, to CSPs. DSS requires a fixed limited memory on the client. During the secure ML delegation phase, client uses an interactive secure protocol to outsource computations on the large matrix C to the CSPs.

Design Goals. CryptoML's design satisfies the following goals: (i) Approximate computing: Our design takes advantage of the flexibility of ML algorithms in handling variations in solution to gain runtime, energy, and memory efficiency. For a given user-defined approximation error, the computational workload is reduced to the limits imposed by the data sketch. (ii) Resource Awareness: Our design assigns the majority of memory and computational workload to the powerful CSPs. As such, during processing the ML algorithm, the client's memory and computational workload is limited to $\mathcal{O}(N)$ as opposed to $\mathcal{O}(MN)$. (iii) Privacy Preserving: Our design protects the privacy of the client's data. The original large database and the solution to the ML problem are *not* revealed to the CSPs.

Attacker and Security Model. We focus on a resource-constrained client and N_s independent CSPs: $CSP_1, \ldots, CSP_{N_s}$, where client has a separate personal user account with each CSP. The client authenticates itself to each independent server before uploading or downloading data. This model describes the capabilities provided by the existing CSPs, such as Microsoft Azure and Amazon Elastic Compute Clouds. Our protocol is suitable for the honest but curious threat model where the CSPs perform the computation correctly but may attempt to infer information from the delegated data or computations. Security of our protocol can then be guaranteed even when the adversary has unbounded computing power. In our setting, the CSPs are not required to communicate with each other. Our main tool to build the secure outsourcing protocol is Shamir's secret sharing that is based on polynomials over a fixed finite field (denoted by $\mathbb{K}$ here) [8]. A value $s \in \mathbb{K}$ is shared by choosing a random polynomial $P_s(X) \in \mathbb{K}[X]$ of degree at most t such that $P_s(0) = s$. The share $s_j = P_s(j)$ is then sent privately to player j. It is well known that any set of t or fewer shares contain no information about s, and s can be reconstructed from any $t+1$ or more shares.

Target ML Algorithms. The key to many important ML algorithms is exploring the dependency between various data points. In particular, the pairwise correlation (*Gram*) matrix of variables or signals, computed with $G = A^T A$, is widely used for representing dependencies. Using the Gram matrix, most learning algorithms iteratively update a vector of unknown parameters until they converge to a solution. Each update process requires matrix multiplications in the form of $Gx = A^T Ax$, where x is the unknown parameter vector. Big data ML analysis, often performed on distributed processors, incur a high computational and communication cost because of iterative multiplications on large matrices. Due to their universal applicability, CryptoML targets iterative algorithms that operate on massive and dense (non-sparse) Gram matrices. For our evaluations, we consider power method, which can be used for solving a variety of applications in image processing, visualization, exploratory data analysis, pattern recognition, and time series [16, 12].

3. CryptoML'S DATA SKETCHING

We introduce DSS, our proposed sketching approach that reduces the computation workload by producing low-dimensional and sparse factors. We exploit the well-known fact that many learning applications are tolerant to output solution variations, offering the opportunity to trade exact solutions with improvement in resource cost [9, 14]. More formally, for a given sketch approximation error ϵ, DSS seeks to find a suitable dictionary matrix D and a sparse coefficient matrix C such that:

$$\min \|C\|_0 \ s.t. \ \|A - DC\|_F \leq \epsilon \|A\|_F, \tag{1}$$

where D is $M \times L$, C is $L \times N$, and $L \leq M$ is the reduced dimension; $\|C\|_0$ is the total number of non-zeros in C ($nnz(C)$), and $\|\cdot\|_F$ is the Frobenius norm. We are interested in reducing $nnz(C)$ as each non-zero element in C incurs memory access, FLOPs, and communication overhead, which adversely affect the performance. Parameter L contributes to communication cost and can be specified by the client. The optimally minimum value for L to achieve a desired approximation error can be found via a heuristic divide and conquer method. In Section 4, we formally quantify our delegation protocol's performance cost and discuss how a client can use this quantification to customize the sketch with respect to the underlying resource limitations. D is often much smaller than A (i.e., $ML \ll MN$), since the number of signals N in big data applications can be arbitrary large. Our results show that for real-world applications, and within guaranteed user-defined approximation errors, D can consume up to two orders of magnitude less memory than A.

Algorithm 1 demonstrates CryptoML's sketching. First the dictionary D is created by sub-sampling columns of A uniformly at random (Steps 1-2). Once D is created, Equation 1 becomes equivalent to a generic sparse approximation problem; each column c_i of C is a sparse approximation of the column a_i of A with respect to D and the user-specified projection error ϵ. We adopt Matching Pursuit (MP) which is a greedy sparse approximation routine [19] that iteratively selects the highest correlated columns in the dictionary to the signal (Step 3). Each iteration produces a non-zero coefficient in c_i. MP stops either when the error tolerance criteria is met, or when a pre-specified number of iterations (denoted by K in Algorithm 1) is passed.

Our approach is motivated by recent results for sparse subspace clustering [9], where the goal is to discover multiple low-dimensional subspaces present in a collection of data and then cluster signals according to their subspace membership.

Algorithm 1 : DSS

Input: Normalized data matrix $A \in \mathbb{R}^{M \times N}$, approximation error tolerance ϵ, number of columns in dictionary L, and (optional) maximum number of non-zeros per C column K (otherwise $K = L$).

Output: A sparse matrix $C \in \mathbb{R}^{L \times N}$ and a dictionary $D \in \mathbb{R}^{M \times L}$ such that $\|A - DC\|_F \leq \epsilon \|A\|_F$.

1. Client creates a random subset of in dices of size L denoted by I_L
2. Client creates $D = A(:, I_L)$.

3. Client applies Matching Pursuit to solve $a_i = Dc_i$ for the approximation error ϵ:
   3.0. Initialize $r = a_i$,$\phi = \emptyset$
while $\|r\|_2 < \epsilon \|a_i\|_2$ or $|\phi| < K$ **do**
   3.1. $k = argmax_j |d_j.r|$
   3.2. $\phi = (\phi, k)$.
   3.4. $r = r - |d_k.r| d_k$
end while

The key intuition is that a sparse representation of a data signal (columns of A) ideally corresponds to a combination of a few other signals. By introducing the approximation, the sparsity level increases. In our evaluations, we show how moderate increasing of ϵ produces a more compact output, resulting in further reduction in resource-consumption on both the client and server side.

Complexity Analysis. Algorithm 1 is highly efficient and hence suitable for a resource-limited client. The main computing task is executing MP sparse coding routine. Since each column of C is created independently, this algorithm is highly parallel. The computational complexity for computing a column of C is $\mathcal{O}(LMK)$. As we show in our experiments, L (and subsequently K as $K \leq L$) can be up two orders of magnitude lower than M within reasonable approximation errors. The infrequent pre-processing cost is quickly paid off due to the savings in the overall cost of secure iterative Gram matrix based updates.

4. CryptoML'S DELEGATION

We propose CryptoML's customized and resource-aware protocol for secure and effective delegation of ML computations. Our protocol takes advantage of the iterative nature of target ML algorithms. In the following we discuss secret-sharing, the building block of our secure protocol, and provide its ML-specific analysis, which leads to determining the minimum number of participating CSPs required for secret-sharing. Given this minimum number, we propose a delegation protocol that incurs the provably minimum communication overhead between the client and CSPs while simultaneously reducing the overall cost of secure ML execution in terms of runtime, energy, and memory usage.

4.1 Secret Sharing in CryptoML

In our protocol, we are interested in offloading massive matrix multiplications to CSPs using Shamir's model. Recall that the complexity of iterative ML algorithms arise from large matrix multiplications. To get the multiplication result of two matrices using secret sharing, if elements of each matrix are encoded with a degree t polynomial, the result would be a degree $2t$ polynomial. Thus, to recover the matrix product, $2t+1$ shares of each matrix are needed. In the following, we first identify a bound on the minimum number of required CSPs. Given this bound, we design a protocol that provably reaches the minimum communication overhead between the client and CSPs. We demonstrate that our communication-minimizing design also results in minimized overall ML execution cost.

Proposition 1. Consider a scenario where a client aims to outsource the multiplication of 2 or more matrices using Shamir's SS. In this case, the minimum number of required CSPs (such that not all of them collude) is 3.

Proof. According to Shamir's scheme, in order to secret share the multiplication of two values, each encoded with a degree t polynomial, $2t+1$ shares are needed. If less than 3 CSPs are used, at least one CSP receives more than t shares, violating the condition for secrecy.

Based on proposition 1, to securely outsource a Gram matrix based update $A^T Ax$ using Shamir's scheme, at least 3 CSPs such that not all of them collude are required. ■

Proposition 2. Consider a scenario where a client aims to compute the multiplication of n matrices, i.e., $Z = \prod_{1 \leq i \leq n} Z_i$, using Shamir's SS for secure delegation. If the client desires to limit its interactions with the CSPs to only once, i.e., a one-time offloading of shares and a one-time reception of results from the CSPs, at least $n+1$ (not-all colluding) CSPs are required.

Proof. Assume that each matrix Z_i is secret shared using a polynomial of degree t_i. We denote the minimum number of CSPs by n_s. Then:

$$\frac{(\min_{1 \leq i \leq n} t_i)n}{n_s} < \frac{\sum_{1 \leq i \leq n} t_i + 1}{n_s} \leq \min_{1 \leq i \leq n} t_i. \quad (2)$$

The left-side inequality holds by definition. We show that the right-side equality is also true. For now, let us consider a case where the client asks each CSP to contribute in multiplication of all Z_i $(1 \leq i \leq n)$ matrices, i.e., each CSP holds shares from all the Z_is. The client has to outsource $\sum_{1 \leq i \leq n}(t_i) + 1$ shares to be able to recover product matrix Z, however, none of the CSPs can have more than $i_{min} = \min_{1 \leq i \leq n} t_i$ shares. Otherwise, according to Shamir's SS, they can recover matrix $Z_{i_{min}}$. Thus, the inequality holds. From the inequalities, and since n_s and n must be integer values, we conclude that $n_s \geq n+1$.

Now consider another case where the client sends shares of a *subset* of matrices to each CSP, say shares of $Z_1, \ldots, Z_j$ to some and $Z_{j+1}, \ldots, Z_n$ to others. In this case Z is computed as $Z = (\prod_{1 \leq i \leq j} Z_i)(\prod_{j+1 \leq i \leq n} Z_i)$. Due to the limit on the number of interactions between the client and CSPs, the problem becomes an instance of the previous case: At least $(j+1) + (n-j+1) = n+2$ distinct CSPs are required. Similarly, it can be shown that breaking the product into more sub-products increases the number of required CSPs. Thus, Proposition 2 holds. ■

In an iterative ML algorithm, each iteration requires a $A^T Ax$ update which is a product of 3 matrices. Thus, if we limit the number of per iteration interactions between the CSP and client to only once, at least 4 (not-all colluding) CSPs are required. Therefore, to securely outsource a Gram matrix based iterative update with the minimum number of CSPs (i.e., 3) the number of per iteration interactions between the CSP and client has to be more than one.

4.2 Resource-Optimized Delegation

Algorithm 2 outlines our 3-CSP communication-minimizing sketch delegation protocol. Instead of secret sharing the original matrix A, the sparse coefficient matrix C is outsourced. The computations associated with the limited-size dictionary matrix D is done on the client (Step 3). The communication in this protocol is limited to

Algorithm 2 : Secure Delegation of Iterative Analysis on Sketch.

Input: Vector $x_{N\times 1}$ and sketch matrices $D_{M\times L}$ and $C_{L\times N}$.

Output: $C^T D^T DCx$.

0. Client encodes C with degree t polynomials and sends $2t+1$ shares to CSPs.

One Iteration of Gram Matrix Update:

1. Client encodes x with degree t polynomials and sends $2t+1$ shares to CSPs.
2. CSPs compute $L\times 1$ vectors $v_1 = Cx$ corresponding to each share and send the $2t+1$ results to client.
3. Client decodes results to find Cx. It then computes $L\times 1$ vector $v_2 = (D^T D)Cx$. It encodes v_2 with degree t polynomials and sends $2t+1$ shares to CSPs.
4. CSPs compute $N\times 1$ vectors $v_3 = C^T((D^T D)Cx)$ corresponding to each share and send $2t+1$ results to client.
5. Client updates x by decoding results to find $C^T(D^T D)Cx$.

$(2N(2t+1)+2L(2t+1))$ per iteration.

The pre-processing results in a lower dimensional and spasre coefficient matrix C. Since each element in matrix C contributes to the cost associated with communication, computation, and memory access, to achieve performance efficiency, we limit the computation and secret sharing in Algorithm 2 to the non-zero elements of C and ignore the zero values. One can obfuscate the location of non-zero elements by randomly permuting rows and columns of C. If C $(M\times N)$ has K non-zeros per column, there are at least $\binom{L}{K}K!$ distinct ways to permute rows of C. Thus, a brute force attack is computationally intractable for real-world datasets. Note that even if the location of non-zeros in C is revealed to the CSPs, such information does not reveal anything about the values of non-zero elements. This is because according to Shamir's scheme, unless the 3 CSPs collude, it is absolutely impossible to recover a value given the insufficient number of shares.

4.3 Protocol Performance Quantification

Bounds on Number of FLOPs. The cost of FLOPs arises from the number of operations required for secret sharing (encryption/decryption) and performing arithmetics to compute $C^T D^T DCx$. In general, based on Lagrange interpolation, the cost of encrypting an element with a polynomial of degree t, is $\mathcal{O}(t^2)$. In Algorithm 2, given that the computation is limited to the non-zero elements, the cost of matrix multiplication on each share, i.e., Steps 2 and 4, becomes equal to the number of non-zeros in C or $nnz(C)$. Recall that according to Algorithm 1, each column of C has at most K non-zeros, hence $nnz(C) = KN$. The matrix multiplication cost of Step 3, is L^2.

Bounds on Communication. The cost of communications arises from sending and receiving data in each step. Matrix C is transmitted only once to the CSP, resulting in communication of KN *words*, where a word is a unit of data (i.e., a 64-bit unit). Each non-zero element is stored as a triple (row-index, column-index, value). In each iteration, shares of $N\times 1$ vectors x and $C^T(D^T D)Cx$, and $L\times 1$ vectors v_1 and v_2 are communicated.

Bounds on Memory. Table 1 summarizes the costs for our protocol. To better understand the performance improvements achieved by CryptoML, we also provide the costs for the case where the iterative protocol is run on the original data A. The protocol corresponding to $A^T Ax$ is achieved by simply replacing C with A and skipping Step 3 in Algorithm 2. These rows are marked by *w/o CryptoML*. Since the number of multiplications and additions are asymptotically the same, for brevity we only report the number of multiplications under FLOPs.

As mentioned in Section 2, we are in the big data regime where $M < N$. We will show in our evaluations that for a variety of real-world datasets L (and subsequently K as $K < L$) can be more than an order of magnitude lower than M. Our protocol not only reduces the client-side computations, but also it significantly reduces the memory and computation cost on the CSPs, which results in a higher overall performance and lower dollar cost.

Given the asymptotic secure delegation cost analysis in Table 1, a client can customize the sketching parameters (including L, approximation error ϵ, or sparsity-level K) in Algorithm 1 to meet the constraints imposed by its resource limitations. For example, client can reduce parameters K or L (at the cost of a higher approximation error) to achieve a desired memory or communication overhead.

4.4 Protocol Properties

CryptoML successfully realizes our crucial design objective (Section 2), i.e., to delegate the majority of memory and computation workload to the CSPs. As seen in Table 1, for each iteration, the client requires $\mathcal{O}(t^2)(N+M)$ and $\mathcal{O}(t^2)(N+L)+L^2$ computation for the original and the sketch, respectively. Each CSP requires $\mathcal{O}(t)MN$ and $\mathcal{O}(t)KN$, respectively. In our protocol, CSPs only interact with the client and not with each other; this naturally lowers the possibility of collusion.

Shamir's secret sharing requires operations over a finite field $\mathbb{K}$. The specific nature of our delegation protocol enables efficient ways for processing floating-point data while benefiting from Shamir's scheme. Since the CSP computations are limited to matrix multiplications, the delegated memory and operations can be readily done in fixed-point format. After each iteration, the client (if necessary) applies type conversion on the much more compact (multiplication) result and executes further (light-weight) ML-specific operations on it. It then converts back the compact result to fixed-point and sends it to CSPs for next iteration.

Verification of the results and detection of cheating CSPs is straightforward in our secure delegation scenario; The client can at any moment compute the actual element-wise products on portions of the data locally and compare it against those received from the CSPs.

5. RELATED WORK

Performing privacy-preserving ML algorithms on the cloud has been the subject of intensive research in the past with great progress, e.g., [3, 22, 4, 15, 18, 5]. The available methods range from solutions relying on fully homomorphic encryption (FHE) and Garbled Circuits (GC), to secret sharing [3], trusted hardware [4] and to secure compilers [22]. However, most earlier works in privacy preserving ML are inappropriate for small-client cases. Secure ML solutions based on FHE or GCs are still not at the efficiency level required for our delegation scenario. For example, a recent work GraphSC [17], reported using ORAM and GCs on GraphLab for iterative matrix updates. In their evaluations, a single iteration update on a connectivity matrix with 1 million non-zeros took more than 13 hours on a powerful server

Table 1: Computation, communication, and memory costs of performing one iteration of Gram matrix based update. The one-time costs (Step 0 of protocol) represent the communication overhead of sending data to CSP before iterative updates start. However, the per-iteration costs (Steps 1-5) are repeated many times until the ML algorithm converges to a solution. CryptoML enables significant performance improvements as for many real-world datasets $K < L \ll M < N$.

		Computation (FLOPs)		Memory (Words)		Communication (Words)
		Client	CSPs	Client	CSPs	Client-CSPs
One-time	w/o CryptoML	$\mathcal{O}(t^2)MN$			$\mathcal{O}(t)MN$	$\mathcal{O}(t)MN$
One-time	CryptoML	$\mathcal{O}(t^2)KN$		ML	$\mathcal{O}(t)KN$	$\mathcal{O}(t)KN$
Per-iteration	w/o CryptoML	$2\mathcal{O}(t^2)(N+M)$	$2\mathcal{O}(t)(MN)$	$\mathcal{O}(t)(N+M)$	$\mathcal{O}(t)(MN+N+M)$	$2\mathcal{O}(t)(N+M)$
Per-iteration	CryptoML	$2\mathcal{O}(t^2)(N+L)+L^2$	$2\mathcal{O}(t)(KN)$	$\mathcal{O}(t)(N+L)+L^2$	$\mathcal{O}(t)(KN+N+L)$	$2\mathcal{O}(t)(N+L)$

using 128 AMD Opteron processors. Such performance is significantly less efficient compared to our work which enables processing of matrices with billions of nonzeros.

Several practical methods for secure delegation of specific applications have been developed. For example, (privacy-preserving) large-scale mapping of genomes [6], MapReduce framework [22], and linear programs [3]. The available solutions are either based on security in a non-standard (weak) model [3], or they are inapplicable to massive data learning scenarios that are of interest to our work. For instance, general parallel programming frameworks such as MapReduce [22] are not suitable for addressing most large-scale ML problems as dense data dependencies prohibit task parallelization that is the basic assumption of the MapReduce framework. While Shamir's SS is a widely known and acknowledged method for provably secure computation, we have not found any work that uses it for secure delegation of ML algorithms. In fact, [3] is the only relevant work for delegating computations based on a heuristic approximation of Shamir's SS. This work is not concerned with sketching or iterative ML updates; it only introduces a new security protocol for matrix multiplication on single and double server cases with weak security guarantees.

In summary, little work has been done to address the generic problem of secure and scalable delegation of complex ML algorithms in practice; prior works either focus on non-proven secure settings [3] or utilize costly non-scalable cryptographic operations for a different delegation scenario [17, 15, 5] (such as multi-party computing). Our protocol hides the data, model, and the final results from the CSPs. It is provably secure, and it is the first-ever secure ML delegation work that reports complex iterative functions over datasets with billions of non-zeros in their correlation matrix. One strong point of the our solution is that it covers a wide range of iterative matrix-based ML algorithms.

6. EXPERIMENTS

To evaluate our work, we answer the following questions: (i) What is the impact of CryptoML on memory reduction of real datasets for various introduced approximation errors, (ii) How does our secure delegation protocol improve the overhead in terms of memory usage, processing time, and communication, and (iii) What is the impact of introducing different approximation errors in DSS on accuracy of solutions to the ML problems for various real applications.

Implementation and API. We implement CryptoML in C++ using the standard message passing system (MPI). Data is stored using a 64-bit fixed-point representation, with 20-bit for the fractional part. Our API takes the following user inputs: dataset A, sketch error ϵ, and the learning algorithm as an iterative update function on Gram matrix. Our modular library can be easily used to implement various ML update functions on top of the Gram matrix multiplication. All of our evaluations are done on IBM iDataPlex cluster. Each node in the cluster is an Intel-Xeon-X5660@2.80GHz. We set the client node's memory to $8GB$ and each of the 3 server nodes memory to $48GB$. Since the client and server nodes are on the same network, the packet transmission time is faster compared to an out-of network communication setup. However, a slower network only increases the gap between the (lower) communication overhead in CryptoML, versus the (larger) overhead of baseline. Our results along with numerical quantifications in Table 1 can be used to find the overall communication cost for any network delay.

Datasets. Our experiment datasets are shown in the first row of Table 2; Salina, is derived from hyperspectral imaging [2], Cancer Cell is derived from MRI medical imaging, and Light Field is derived from plenoptic cameras [1].

6.1 CryptoML Evaluation

Memory Reduction Capability. We verify the capability of CryptoML to create low-dimensional and sparse transformations. Table 2 shows the effect of varying the transformation approximation error ϵ. The $M \times N$ input matrix A is transformed to an $M \times L$ matrix D and a sparse $L \times N$ matrix C, whose average number of non-zeros per column is K. The memory reduction is computed by $\frac{nnz(D)+nnz(C)}{nnz(A)}$, where $nnz(.)$ measures number of non-zeros. Significant memory reductions are achieved. For example, for Light Field data, the reduction is more than 25 folds. Increasing ϵ, further reduces the memory footprint.

The approximation error in CryptoML can be customized to meet a given performance budget. Given the data-dependent transformation results and quantifications in Table 1, one can estimate the actual delegation cost in terms of runtime, memory usage, and communication on the pertinent CSP platforms and tune the ϵ accordingly.

Runtime Analysis. We apply CryptoML to real-world datasets and report the actual measured runtime of the algorithms. In these experiments, all tests are done on IBM iDataPlex platform where the client node has 4 cores at $8GB$ RAM (which emulates a typical portable laptop) and the 3 CSPs each have 32 cores at 48GB RAM. Figure 2 compares the runtime performance of power method for finding the first 100 eigenvalues of different datasets. The results show significant runtime improvement of up to 30× over baseline, which is the case where we apply the protocol on the original data A. As it was expected from quantifications in Table 2, CryptoML achieves runtime improvements for all datasets over baseline. Our timing results indicate applicability of CryptoML to large-scale problems. To the best of our knowledge, CryptoML is the first to demonstrate a practical secure iterative delegation on datasets with billions of non-zero elements (Light Field data).

Accuracy Analysis. The bottom row in Table 2 shows the learning error in finding singular-values. The learning error is the normalized cumulative error of the first 100 eigenvalues

[0]This dataset consists of cancer tumor morphologies collected in MD-Anderson cancer center.

Table 2: CryptoML's performance evaluation. $M \times N$ matrix is transformed to a $M \times L$ dictionary matrix and a $L \times N$ coefficient matrix, whose average number of non-zeros per column is K.

$M \times N$	Salina 203×54129 (87.9MB)			Cancer Cells 1024×111296 (911.7MB)			Light Field 18496×272320 (40.3GB)		
Transformation error	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$
K	148	128	97	889	704	380	780	560	208
L	150	150	100	900	800	650	800	600	600
Memory reduction (×)	1.57	1.37	2.04	1.14	1.28	1.58	25.0	30.7	33.3
Application error	0.0003	0.0003	0.0018	0.0001	0.0004	0.0004	0.00128	0.0029	0.0029

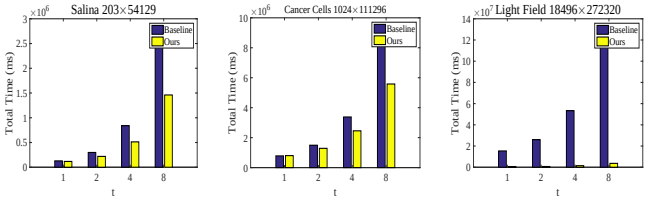

Figure 2: Runtime comparison of PCA application for various t's, the degree of secret sharing polynomial.

found by running power method using CryptoML. The baseline singular values are derived from running power method on A. A notable observation is that although a higher ϵ can result in meaningful runtime and memory improvements, it may not drastically affect the ML solution.

6.2 Discussions

Our evaluations demonstrate the capability of CryptoML in reducing the cost of immensely expensive ML algorithms in secure delegation scenarios. For example, consider a configuration set up with a 4-core ($8GB$) client and three 32-core (each $48GB$) CSPs. For SS of degree $t = 8$, power method on a Light Field data with size 18496×272320 takes more than 36hrs to converge. CryptoML reduces this to only slightly more than one hour (Figure 2), while the client's one-time pre-processing overhead is less than 31 minutes. What further makes the pre-processing cost negligible is that several learning algorithms (e.g., regularized regressions) require tuning for model selection. This means the learning algorithms are run several times for finding the best learning parameters (e.g., regularization variables). The runtime improvement also translates to dollar-cost reduction on CSPs.

CryptoML introduces a new paradigm in large scale privacy-preserving delegation by leveraging approximate computing. Future directions include developing novel data transformation schemes as well as improving the interactive delegation protocol by exploiting other security primitives. Note that while CryptoML's security protocol is built based on Shamir's secret sharing model, other primitives such as Garbled Circuits and Homomorphic Encryption can also take advantage of our approach. In other words, existing security protocols based on matrix-based analytics can benefit from the reduction in the required memory, FLOPs, and communication achieved by CryptoML. However, as suggested by recent results [17], performance overhead of those primitives make them unsuitable to large matrices.

7. CONCLUSION

We propose CryptoML, a novel customizable framework for secure delegation of iterative ML algorithms to (untrusted) cloud servers. A major novelty of CryptoML is devising a systematic way to realize resource efficiency via approximate computing. We propose a new projection (sketching) approach to create compact approximations of the data. We devise a novel secure delegation protocol on the sketch, based on secret sharing, that takes advantage of the iterative nature of the underlying ML algorithms to achieve performance efficiency. Our protocol delegates the majority of memory and computational workload to the cloud while provably minimizing the (cloud-server) communication overhead. Our work is the first to enable secure delegation of iterative ML algorithms on datasets with billions of non-zeros. The memory, communication, and workload overhead reduction achieved in CryptoML is not limited to a specific data compaction, security primitive, or delegation scenario and can be readily extended to other secure computing tasks.

8. REFERENCES

[1] Stanford light field archive. *lightfield.stanford.edu* (2014).

[2] Aviris hyperspectral data. *www.ehu.es/ccwintco* (2015).

[3] Atallah, M. J., and Frikken, K. B. Securely outsourcing linear algebra computations. In *AsiaCSS* (2010).

[4] Bajaj, S., and Sion, R. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *TKDE* (2014).

[5] Bost, R., Popa, R. A., Tu, S., and Goldwasser, S. Machine learning classification over encrypted data. In *NDSS* (2015).

[6] Chen, Y., Peng, B., Wang, X., and Tang, H. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *NDSS* (2012).

[7] Coates, S. snd Huval, B., Wang, T., Wu, D., Catanzaro, B., and Ng, A. Deep learning with COTS HPC systems. *ICML* (2013).

[8] Cramer, R., and Damgård, I. Secure distributed linear algebra in a constant number of rounds. In *CRYPTO* (2001).

[9] Elhamifar, E., and Vidal, R. Sparse subspace clustering: algorithm, theory, and applications. *TPAMI'13*.

[10] Ferris, M. C., and Munson, T. S. Interior-point methods for massive support vector machines. *SIAM J. on Optimization* (2002), 783–804.

[11] Figueiredo, M., Nowak, R., and Wright, S. Gradient projections for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE J. Select. Top. Signal Processing 1*, 4 (2007), 586–597.

[12] Fowlkes, C., Belongie, S., Chung, F., and Malik, J. Spectral grouping using the nystrom method. *TPAMI'04*.

[13] Gentry, C., et al. Fully homomorphic encryption using ideal lattices. In *STOC* (2009), vol. 9, pp. 169–178.

[14] Gittens, A., and Mahoney, M. Revisiting the nystrom method for improved large-scale machine learning. *JMLR'13*.

[15] Graepel, T., Lauter, K., and Naehrig, M. Ml confidential: Machine learning on encrypted data. In *ICISC*. 2013.

[16] Journee, M., Nesterov, Y., Richtárik, P., and Sepulchre, R. Generalized power method for sparse pca. *JMLR* (2010).

[17] Nayak, K., Wang, X. S., Ioannidis, S., Weinsberg, U., Taft, N., and Shi, E. *GraphSC*: Parallel secure computation made easy. In *IEEE S&P* (2015).

[18] Nikolaenko, V., Weinsberg, U., Ioannidis, S., Boneh, D., and Taft, N. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE S&P* (2013).

[19] Pati, Y. Recursive function approximation with applications to wavelet decomposition. *Proc. Asilomar Conf. Signals, Systems, and Computers* (1993).

[20] Tibshirani, R. Regression shrinkage and selection via the lasso. *Royal Statist.* (1996).

[21] Yao, A. Protocols for secure computations. In *FOCS* (1982).

[22] Zhang, K., and Ruan, Y. Sedic: privacy-aware data intensive computing on hybrid clouds. In *CCS* (2011).