International Journal of Leading Research Publication (IJLRP)

E-ISSN: 2582-8010 ● Website: www.ijlrp.com ● Email: editor@ijlrp.com

Implementation & Analysis of Coded Machine Unlearning Protocols

Uttej Thandu[1], Athulya Reddy[2], Prasanth Tirumalasetty[3], RajKumar Edury[4], Srija Virigineni[5]

[1,2,3,4,5]CECS Department University of Michigan Dearborn, USA.

Abstract:
There are some applications that may necessitate removing the trace of a sample from the system, such as when a user requests that their data be deleted or when corrupted data is discovered. Simply removing a sample from storage does not necessarily remove its entire trace, because downstream machine learning models may store some information about the samples used to train them. If a sample is completely unlearned, it can be completely unlearned. We retrain all models that used it from scratch, removing that sample from their training dataset. When multiple such unlearning requests are anticipated, unlearning by retraining becomes prohibitively costly. The training data can be divided into smaller disjoint shards and assigned to non-communicating weak learners using ensemble learning. Each shard is used to create a faulty model. These models are then combined to form the final central model. In this paper, we propose a coded learning protocol in which the training data is encoded into shards prior to the learning phase using linear encoders. In addition, we present the corresponding unlearning protocol and demonstrate that it meets the perfect unlearning criterion.

Keyword: Coded machine learning, machine learning, unlearning.

I. INTRODUCTION

Machine learning (ML) has grown commonplace in the last decade because of the quantity of data. After an ML model has been trained, some samples in the training dataset may need to be unlearned for a variety of reasons, such as to satisfy users' requests for data removal, or due to the discovery of corrupt low-quality samples or adversarial modified samples designed to harm the ML model's performance. Because machine learning models can be arbitrarily complicated and trained on vast datasets and have the capacity to memo rize training data, it's critical to develop efficient unlearning methods that can handle any model. This issue is unrelated to privacy-preserving machine learning, because guaranteeing differential privacy does not eliminate the requirement for an unlearning process. Indeed, while differentially private algorithms guarantee a limit on how much individual training points contribute to the model and keep this contribution minimal, each point nevertheless contributes something. The model would be unable to learn if this were not the case. Forgetting, on the other hand, necessitates that a given training point contributes 0 to the model, which runs counter to the differential privacy promise. Knowing how each training points contributed to model parameter updates is required to
have models forget. When the dataset is queried adaptively, that is, when a given query is based on previous queries, the approach's convergence is no longer guaranteed. Only for models that require a minimal number of iterations for learning are the divergences caused by this approach bounded in the adaptive setting. While it is true that any PAC algorithm can be transformed to its counterpart in the SQ learning context, there are no effective SQ learning algorithms for complicated models like DNNs. Retraining such models from scratch is a foolish method to make them forget. Instead of fully retraining models affected by training data deletion, our study tries to hold ML to standards such as the right to be forgotten through the ability to unlearn, which has a huge computational and time overhead. Unlearning ensures that a trained model is no longer taught with data that the user has chosen to delete. To put it

another way, unlearning ensures that training on a point and then unlearning it will result in the same model distribution as not training on the point at all.

II. RELATED WORKS

In the literature, several works have been offered to provide effective unlearning solutions. The removal of data from learning models is guaranteed with perfect unlearning. For example, to speed up the unlearning process, employs statisti cal query learning. A broader paradigm for perfect unlearning involves an ensemble learning configuration in which a master node shard the training dataset and distributes the shards to noncommunicating weak learners who are trained separately, and then aggregates their results. Statistical unlearning is an alternative approach to the unlearning problem, in which data removal protocols provide statistical guarantees of removal, analogous to statistical approaches for calculating leave-one out cross validation. For example, a statistical formulation of data deletion from machine learning is presented, which is comparable to differential privacy, and a method to achieve deletion in linear and logistic regression scenarios is described. With a brief discussion on deletion in ML models, a formula tion of data deletion problems using cryptographic notations is offered and also contains other papers on statistical unlearning. Statistical unlearning is often appropriate for well-behaving convex models; however, it often provides no guarantees for non-convex models. Another closely related line of work is
concerned with individual samples' privacy; this is relevant in cases where the samples contain highly sensitive information, and they need to be kept secret even from the ML model.

III. EXPERIMENTS

In this part, we report simulation results from certain experiments comparing performance versus unlearning cost on genuine and synthetic datasets for two protocols: the proposed coded machine unlearning protocol and the uncoded machine unlearning protocol published in [1]. The experiments replicate the unlearning of a sample from the training dataset, with performance expressed as a mean. The unlearning cost and squared error are calculated in terms of the time required to retrain the impacted weak learner. To generate simulation results for all of the tests, we use the sklearn.linear model package, especially the Linear Regression and Ridge mod ules. We sweep the variable s while fixing the rate for the programmed scenario and monitor performance since the cost of unlearning is proportional to the size of the shards. Each point in the plots represents the average of a number of runs, with each run simulating the experiment on a randomly shuffled dataset that is then divided into training and testing datasets according to the sizes provided. During each run, after separating the dataset into training and testing, Algorithm 1 is run first, using s and r for a specific code with rate t = s/r and density = 1/r for a certain code. After the model has been trained, Algorithm 3 is used to unlearn a random sample of the training dataset. After all of the runs have been completed, the performance is calculated as the average mean squared error of the testing dataset, while the unlearning cost is calculated as the average time necessary to retrain the impacted weak learners, because removing a sample from the dataset is free. The datasets are pre-processed in the following way for the simulations: each column of the original feature matrix and the response vector is normalized to be in the range [0, 1]. The projections are done on the normalized features using a cosine activation function and the following parameters if the random projections approximation is employed as mentioned in [2].

IV. IMPLEMENTATION

Data Set Information: The dataset contains 9568 data points collected from a Combined Cycle Power Plant over 6 years (2006-2011), when the power plant was set to work with full load. Features consist of hourly average ambient variables Temperature (T), Ambient Pressure (AP), Relative Humidity (RH) and Exhaust Vacuum (V) to predict the net hourly electrical energy output (EP) of the plant [3]. A combined cycle power plant (CCPP) is composed of gas turbines (GT), steam turbines (ST) and heat recovery steam

generators. In a CCPP, the electricity is generated by gas and steam turbines, which are combined in one cycle, and is transferred from one turbine to another. While the Vacuum is collected from and has effect on the Steam Turbine, he other three of the ambient variables effect the GT performance [4]. For comparability with our baseline studies, and to allow 5x2 fold statistical
tests be carried out, we provide the data shuffled five times. For each shuffling 2-fold CV is carried out and the resulting 10 measurements are used for statistical testing. We provide the data both in .ods and in .xlsx formats. [5]

A. Implementation

1)We used Jupyter Notebook for the implementation of algorithms using Python as statistical programming. 2)The dataset described above is then loaded into the Jupyter notebook.
3)Imported all the necessary libraries.
4)The dataset has been divided into 5 shards(encoded). 5)A weak linear regression model has been trained on each shard.
6)Each model outputs 4 coefficients as we have 4 features. 7)Took the mean of each coefficient and used it for prediction. 8)To unlearn samples from this model we track the index of the sample in the coded shards and delete it.
9)We train the weak learners again and take the mean of the coefficients to get the final parameters.
10)We use the generator matrix to track these samples. More sparse the matrix is for the particular sample, lesser are updates in weak learners. [6]

V. PROBLEM SETUP

Consider the following scenario: the training dataset is a matrix denoted as [X, y], the rows of which are the indepen dent and identically distributed samples xi and their responses yi for i = 1, 2,..., n, where xi E X and yi E Y. The number of samples is denoted by n, and the number of features is denoted by d. The columns of X are referred to as the features, while the variable y is referred to as the response variable.
yi = f(xi) + e (1)
A Training dataset is used to train a learning model, which results in a model, i.e., a function f: X –¿ R that minimizes a loss function. The loss is a function that measures the goodness of fit of the model f E F on the training dataset for regression problems [7].
For the regression problems, Representer Theorem is gen erally used, It states that for the regularized loss, when using a strictly increasing function and a kernel k with F as its associated Reproducing Kernel Hilbert Space (RKHS). This powerful theorem translates any regression problem, even nonlinear problems, as a linear problem in the RKHS. [8] In regimes with large training datasets, these kernel methods suffer greatly. For a dataset with a fixed number of features, computations of the elements in the kernel matrix result in an additional complexity of O(n 2) on top of the optimiza tion method used to solve the problem. This gives a good approximation of the function f* using random projections to a D-dimensional space where d ¡ D n, allowing efficient linear regression methods to be used to solve the regression problems [9].
The model is utilized for prediction once it has been trained until an unlearning sample request is received. The model stops processing prediction queries and starts the unlearning
protocol once unlearning requests come. When a request to unlearn sample [x, yu] from the training dataset is received, the model must be updated quickly to remove any influence of this sample, i.e., unlearn it, from Mh (X, y). Unlearning techniques ensure that samples are completely removed from the model, although they may be inefficient. Perfect unlearning is achieved by removing samples from the training dataset and starting over with a new model. However, because retraining is the process that causes most of the delay in this technique, it is necessary to build efficient unlearning protocols that can be employed for big datasets.

A. Proposed solution

The solution for this problem is described in two parts, Learning and Unlearning. We provide a method

for encoding the training dataset prior to training and define a specific coding scheme for a regression learning model during the learning phase. We go onto the unlearning phase once the model has been trained, and we propose an efficient technique for processing unlearning requests using the coded training dataset, as well as updating the model to precisely unlearn the requested samples.
model M, it is delivered directly to each of the weak learners, resulting in weak predictions f* j (x) for j = 1, 2,..., r, and then the model M computes the final prediction f* (x) using an aggregation function a: R r-¿ R, such as averaging, a majority vote, and so on.
We know that the resulting model f* j is the corresponding weights w* j for linear regression models or nonlinear regres sion models with random projections. M generates a matrix W whose columns are the estimated coefficients from the weak learners after all weak learners Lj's have been taught. Once W is available, the model M computes the aggregate prediction weights by averaging the weak learners' weight vectors to give w agg, which is immediately employed at prediction time, avoiding the weak predictions calculations.

The training dataset encoding is a method for creating a new training dataset with the purpose of lowering learning and unlearning costs. Coding is a technique for combining numerous samples from an uncoded training dataset into a single sample of a coded training dataset to facilitate learning and unlearning. In other words, whereas weak learners have fewer coded samples, each of these coded samples is made up of several uncoded samples, allowing the model to learn these uncoded data indirectly. To begin, consider the following definition of an encoder:

Algorithm 1 Learning (Learn)

1: Input: $[\boldsymbol{X}, \boldsymbol{y}], s, r, \rho$.
2: Output: $\boldsymbol{W}^*, \{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}, \boldsymbol{G}$.
3: **At master node** $\mathfrak{M}$, **do**
4: **if** $s \neq 1$ **then**
5: $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}, \boldsymbol{G} \leftarrow \text{LinearEnc}([\boldsymbol{X}, \boldsymbol{y}], s, r, \rho)$.
6: **else**
7: $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\} = \{[\boldsymbol{X}, \boldsymbol{y}]\}$
8: $\boldsymbol{G} = [1]$
9: **end if**
10: Send $[\overline{\boldsymbol{X}}_i, \overline{\boldsymbol{y}}_i]$ to weak learner $\mathfrak{L}_i$
11: **At weak learner** $\mathfrak{L}_i$, **do**
12: $\boldsymbol{w}_i^* \leftarrow \text{argmin}_{\boldsymbol{w}} \ell([\overline{\boldsymbol{X}}_i, \overline{\boldsymbol{y}}_i], \boldsymbol{w})$
13: Send $\boldsymbol{w}_i^*$ to $\mathfrak{M}$
14: **At master node** $\mathfrak{M}$, **do**
15: $\boldsymbol{W}^* = [\boldsymbol{w}_1^*, \boldsymbol{w}_2^*, ..., \boldsymbol{w}_r^*]$
16: $\boldsymbol{w}_{\text{agg}}^* = \frac{1}{r} \sum_{i=1}^{r} \boldsymbol{w}_i^*$

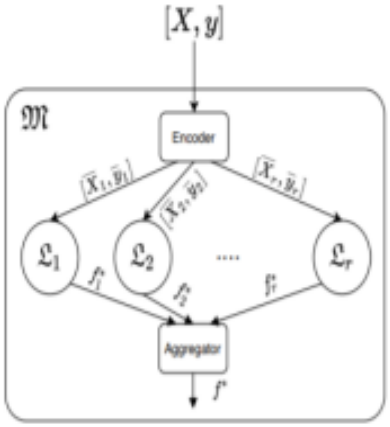


Fig. 1. *P roposed coded learning setup*

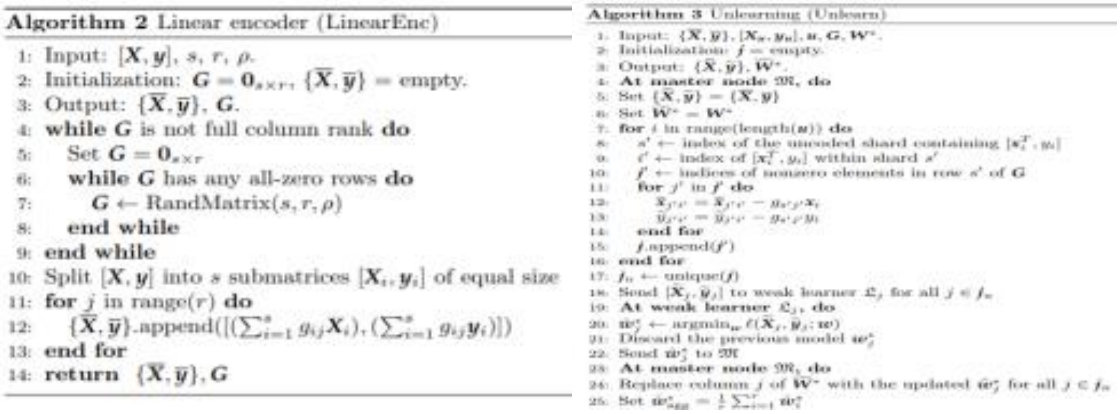
B. Learning the model

As illustrated in Figure 1, the suggested procedure adds the concept of data encoding prior to training the ensemble model. The learning model M, also known as the master node, is used to train a regression model using a training dataset that has been preprocessed. The model begins by running the training dataset through an encoder, which results in a sharded coded training dataset with r coded shards. Then, for each coded shard j, a weak learner Lj is trained to build a model designated as f* j. The model M is ready for prediction once these weak learners have been trained. When a sample x is supplied to the

C. Encoder

The rate may alternatively be seen as the ratio of the number of uncoded shards to the number of coded shards when employing shards of equal size, as in this study. The encoder's rate and design are now new model parameters that must be tuned while training a model. It's worth mentioning that, as we'll see later, the design of the encoder for a given rate has a direct impact on the entire model's unlearning cost as well as its performance. As a result, while creating a model, it should be carefully studied. Algorithm 1

describes the suggested coding learning model for linear regression. The learning algorithm accepts as inputs the training dataset [X, y] and code parameters s, r, and outputs the coded training



Algorithm 2 Linear encoder (LinearEnc)

Input: $[\boldsymbol{X}, \boldsymbol{y}]$, s, r, ρ.
Initialization: $\boldsymbol{G} = \boldsymbol{0}_{s\times r}$, $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}$ = empty.
Output: $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}$, $\boldsymbol{G}$.
while $\boldsymbol{G}$ is not full column rank **do**
  Set $\boldsymbol{G} = \boldsymbol{0}_{s\times r}$
  while $\boldsymbol{G}$ has any all-zero rows **do**
    $\boldsymbol{G} \leftarrow \text{RandMatrix}(s, r, \rho)$
  end while
end while
Split $[\boldsymbol{X}, \boldsymbol{y}]$ into s submatrices $[\boldsymbol{X}_i, \boldsymbol{y}_i]$ of equal size
for j in range(r) **do**
  $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}$.append$([(\sum_{i=1}^{s} g_{ij}\boldsymbol{X}_i), (\sum_{i=1}^{s} g_{ij}\boldsymbol{y}_i)])$
end for
return $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}, \boldsymbol{G}$

Algorithm 3 Unlearning (Unlearn)

Input: $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}, [\boldsymbol{X}_u, \boldsymbol{y}_u], u, \boldsymbol{G}, \boldsymbol{W}^*$.
Initialization: f = empty.
Output: $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}, \overline{\boldsymbol{W}}^*$.
At master node $\mathfrak{M}$, **do**
Set $\{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\} = \{\overline{\boldsymbol{X}}, \overline{\boldsymbol{y}}\}$
Set $\overline{\boldsymbol{W}}^* = \boldsymbol{W}^*$
for i in range(length(u)) **do**
  $s' \leftarrow$ index of the uncoded shard containing $[\boldsymbol{x}_i^T, y_i]$
  $i' \leftarrow$ index of $[\boldsymbol{x}_i^T, y_i]$ within shard s'
  $f' \leftarrow$ indices of nonzero elements in row s' of $\boldsymbol{G}$
  for j' in f' **do**
    $\overline{\boldsymbol{X}}_{j'i'} = \overline{\boldsymbol{X}}_{j'i'} - g_{s'j'}\boldsymbol{x}_i$
    $\overline{\boldsymbol{y}}_{j'i'} = \overline{\boldsymbol{y}}_{j'i'} - g_{s'j'}y_i$
  end for
  f.append(f')
end for
$f_u \leftarrow$ unique(f)
Send $[\overline{\boldsymbol{X}}_j, \overline{\boldsymbol{y}}_j]$ to weak learner $\mathfrak{L}_j$ for all $j \in f_u$
At weak learner $\mathfrak{L}_j$, **do**
$\overline{\boldsymbol{w}}_j^* \leftarrow \text{argmin}_{\boldsymbol{w}}\, \ell(\overline{\boldsymbol{X}}_j, \overline{\boldsymbol{y}}_j; \boldsymbol{w})$
Discard the previous model $\overline{\boldsymbol{w}}_j^*$
Send $\overline{\boldsymbol{w}}_j^*$ to $\mathfrak{M}$
At master node $\mathfrak{M}$, **do**
Replace column j of $\overline{\boldsymbol{W}}^*$ with the updated $\overline{\boldsymbol{w}}_j^*$ for all $j \in f_u$
Set $\overline{\boldsymbol{w}}_{\text{ave}}^* = \frac{1}{r}\sum_{j=1}^{r} \overline{\boldsymbol{w}}_j^*$

dataset, coding matrix, and trained model. The model M encodes the training dataset using the coding parameters given by a linear encoder defined in 7 Algorithm 2.

D. Unlearning

We'll present a process to unlearn samples from the model now that it's been trained on the coded training dataset. Our objective is to erase such samples from the coded shards as well as any trace of such samples from the weak learners who are harmed by such samples. Algorithm 3 describes the unlearning protocol for the aforementioned learning process. The coded dataset X, y, the unlearned samples [Xu, yu], their indices u in the uncoded training dataset [X, y], the matrix G, and the original model calculated coefficients W are the algorithm's inputs. The samples must first be eliminated from the coded shards by subtracting them from the corresponding coded samples in all coded shards where they appear to have no impact. Once all of the coded shards have been updated, the associated weak learners are updated to unlearn these samples from the weak learner models, and the final aggregate model is updated using the revised weak learners' estimates. Other sophisticated models, such as the over-parameterized multi layer perceptron (MLP), cannot be employed since the training loss for these models might be zero. As a result, when a sample is deleted and the model is started from the prior model, it will converge quickly since the training loss is already zero, although this solution was obtained in part because of the removed sample. As a result, in the over-parameterized case, our technique does not completely unlearn the sample.

VI. DISCUSSION

The suggested protocol's success can be shown in cir cumstances when uncoded machine unlearning's performance degrades significantly as the cost of unlearning falls. One theory for why this happens is that it has something to do with the samples used to train each of the weak learners. Influential samples have been extensively studied in the literature [10]. Coding is a method of merging samples into a coded dataset, which includes these influential examples, as we previously described. Let's look at the impact of individual samples on the trained model's performance. The Computer Activity dataset [11] and the Combined Cycle Power Plant dataset are two of the previously considered datasets. We run the following experiment: we shuffle the data at random and divide it into training and testing datasets of the same size as previously, then we delete samples from the training dataset based on certain criteria, then train a single learner on the remaining samples and watch its test MSE. We utilize two removal criteria: the first is that a sample is removed if any of its original attributes fall outside defined percentiles. This criterion eliminates what we call outliers. The other criterion is to eliminate samples whose initial features fall under specific percentiles, which we refer to as inliers. Outliers are samples that are towards the tails of the probability distribution function (PDF), whereas inliers are samples that are near the median. We symmetrically adjust these percentiles on both ends and evaluate performance on the testing data for numerous runs before computing the observed average test MSE. We argue that the existence of these

influential samples, i.e. outlier samples, is one explanation for the behavior of uncoded machine unlearning. If there are important samples in the dataset, the uncoded machine unlearning suffers significantly as the number of shards grows, and the proposed procedure can give a better trade-off. If such important samples do not exist, however, uncoded machine unlearning does not show any loss in performance as the number of shards grows. The suggested technique has no positive impact on uncoded machine unlearning and has no negative impact. It's worth mentioning that these important samples have heavy-tailed distributions, which are frequent in a variety of real-world areas where machine learning is increasingly used, such as technology, social sciences and demography, medicine, and so on. We found that if the probability distribution functions of some of the features have big tails, we have a trade-off for uncoded machine unlearning, and coding gives a better trade off. However, if the probability distribution functions do not
have heavy tails, we do not see this trade-off, and hence coding does not give a better trade-off.

VII. CONCLUSION

We have trained the weak learner after deleting the sample to be unlearned and tested the model using the same unlearned sample. The model predicted a similar result, not the same test result as the model unlearned the sample. We looked at the challenge of perfect machine unlearning for ensem ble learning situations with a master node and several non communicating weak learners trained on disjoint shards of the training dataset in this paper. We concentrated on the trade off between performance and the expense of unlearning. in the case of regression models, We introduced a novel learning approach known as coded learning, which has the potential to improve learning outcomes enable more efficient unlearning while demonstrating a superior performance versus cost trade off When compared to uncoded machine unlearning, the cost of unlearning is lower. We provided a coded learning method ology as well as a linear encoder for regression datasets, as well as its associated unlearning methodology demonstrated its ability to ensure flawless unlearning. We demonstrated that the suggested approach may provide a superior trade-off for a variety of actual datasets with varying values of the underlying parameters through a series of tests. Alternatively, We looked at datasets where uncoded machine unlearning does not demonstrate a trade-off between performance and unlearning cost, and found that coding keeps performance on par with uncoded machine unlearning in these circumstances. In the trials, we demonstrated that by utilizing proper codes, the unlearning cost for a single learner trained on the full dataset may be reduced to a fraction of the cost for a single learner trained on the complete dataset while maintaining comparable performance. Finally, we look at whether we should anticipate the proposed protocol to outperform uncoded machine unlearning based on the presence of important sam ples in the dataset and aspects of the dataset's probability distribution function.

This research is a first step toward better understanding the function of coding in machine unlearning. Extending the suggested protocol to idea classes with larger capacity, such as deep neural networks, is one viable avenue for future research. Another route for study is to look into alternative classes of codes other than linear random codes for supervised learning issues. Another topic for future research is developing proce dures for almost flawless machine unlearning in convex/non convex models when only statistical assurances are necessary. Finally, theoretical investigation of the interaction between influential samples and random coding, as well as their influ ence on the final learnt model, is a promising path for future research.

REFERENCES:

[1] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 141–159.

[2] A. Rahimi, B. Recht *et al.*, "Random features for large-scale kernel machines." in *NIPS*, vol. 3, no.

4. Citeseer, 2007, p. 5.
[3] Y. Sun, Y. Liu, G. Wang, and H. Zhang, "Deep learning for plant identification in natural environment," *Computational intelligence and neuroscience*, vol. 2017, 2017.
[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.
[5] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948. [6] M. Abadi, U. Erlingsson, I. Goodfellow, H. B. McMahan, I. Mironov, N. Papernot, K. Talwar, and L. Zhang, "On the protection of private information in machine learning systems: Two recent approches," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 1–6.
[7] A. Sinha and J. C. Duchi, "Learning kernels with random features," *Advances in Neural Information Processing Systems*, vol. 29, pp. 1298– 1306, 2016.
[8] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Compu tation and Machine Learning Series, 2018.
[9] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 463–480.
[10] D. A. Belsley, E. Kuh, and R. E. Welsch, *Regression diagnostics: Identifying influential data and sources of collinearity*. John Wiley & Sons, 2005, vol. 571.
[11] S.-M. Chen, W.-C. Hsin, and Y.-C. Chang, "A new method for fuzzy interpolative reasoning based on the slopes of fuzzy sets," in *2011 International Conference on Machine Learning and Cybernetics*, vol. 1. IEEE, 2011, pp. 137–141.