ModelPred: A Framework for Predicting Trained Model from Training Data

Yingyan Zeng
Grado Department of Industrial and Systems Engineering
Virginia Tech
Blacksburg, USA
yingyanzeng@vt.edu

Jiachen T. Wang
Department of Electrical and Computer Engineering
Princeton University
Princeton, USA
tianhaowang@princeton.edu

Si Chen
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, USA
chensi@vt.edu

Hoang Anh Just
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, USA
just@vt.edu

Ran Jin
Grado Department of Industrial and Systems Engineering
Virginia Tech
Blacksburg, USA
jran5@vt.edu

Ruoxi Jia
Bradley Department of Electrical and Computer Engineering
Virginia Tech
Blacksburg, USA
ruoxijia@vt.edu

***Abstract*—In this work, we propose ModelPred, a framework that helps to understand the impact of changes in training data on a trained model. This is critical for building trust in various stages of a machine learning pipeline: from cleaning poor-quality samples and tracking important ones to be collected during data preparation, to calibrating uncertainty of model prediction, to interpreting why certain behaviors of a model emerge during deployment. Specifically, ModelPred learns a parameterized function that takes a dataset S as the input and predicts the model obtained by training on S. Our work differs from the recent work of Datamodels [1] as we aim for predicting the trained model parameters directly instead of the trained model behaviors. We demonstrate that a neural network-based set function class is capable of learning the complex relationships between the training data and model parameters. We introduce novel global and local regularization techniques to prevent overfitting and we rigorously characterize the expressive power of neural networks (NN) in approximating the end-to-end training process. Through extensive empirical investigations, we show that ModelPred enables a variety of applications that boost the interpretability and accountability of machine learning (ML), such as data valuation, data selection, memorization quantification, and model calibration.**

I. Introduction

What are the training points with the most or least contribution to the model? How to select data that benefit model performance? Is a training data point memorized by the model? How to produce accurate estimates of uncertainty on model predictions? Which point is the most responsible for learning a given parameter in the model?

Answering these questions is central to building trust in machine learning (ML). Prior work shows that addressing these questions requires analyzing the input-output behavior of a learning process and gaining an understanding of how models (and the resulting predictions) might have differed if different training data points are observed [2, 3, 4, 5].

However, due to the complexity of the learning process (e.g., end-to-end training), analyzing how its input—a dataset—affects the output—the trained model—is challenging. Recent work has made significant progress in approximating how a small change (e.g., removing one or a handful of training points) on the full training set would change the trained model [6, 7, 8, 9], but the existing techniques remain limited in estimating the effects of removing a large group of training points [10]. Ilyas et al. [1] propose to learn a model that predicts the classification performance at a given test point directly from the training data. However, this learning procedure will need to be redone if the classification performance at a different test point is of interest to the evaluation, which is common for practical applications with streaming-in testing data; therefore, it lacks the flexibility to accommodate different evaluation goals.

In this paper, we propose a framework, **ModelPred**, to analyze the dependence of the trained models on training data by building an explicit model for the trained models in terms of training data. Compared to [1], **ModelPred** estimates the trained model parameters instead of the prediction performance at a certain test point, thereby providing flexibility to switch to different test points for evaluation. At the same time, the test performance estimates based on **ModelPred** are more accurate than those from [1] because **ModelPred** effectively leverages the knowledge of how test performance is calculated from a trained model.

Our contributions are summarized as follows:
#1: Introducing ModelPred. ModelPred is a framework designed to approximate the input-output behavior of a learning algorithm, $\mathcal{A}$, when applied to a training set S.

The learning algorithm takes the training set and returns a trained model with parameters $\theta := \mathcal{A}(S)$. To accomplish this, **ModelPred** introduces a cheap-to-evaluate parametric model $\widehat{\mathcal{A}}$. We treat the search for the best $\widehat{\mathcal{A}}$ as a supervised learning problem. Specifically, we generate the labeled data for training $\widehat{\mathcal{A}}$ via executing $\mathcal{A}$ on different data subsets, and then use standard techniques in supervised learning to learn the best $\widehat{\mathcal{A}}$ that approximates $\mathcal{A}$. Our framework differs from the recent work of Datamodels [1] in the sense that we directly predict the trained model parameters instead of some behavior of the trained models.

#2: Instantiating ModelPred. There are several critical design choices of **ModelPred**: What should be used as the input for $\widehat{\mathcal{A}}$? What is the right model class to learn $\widehat{\mathcal{A}}$? How to design a proper objective function to fit $\widehat{\mathcal{A}}$? Ilyas et al. [1] encodes the presence of each training sample of an input subset S' within S as a binary vector. While this reduces the input complexity, this approach ignores the contents of an input subset. As a result, it can only estimate the outcome of learning for a subset *within* the training dataset yet fails to estimate the learning result for, new *unseen* points. In this paper, we use a neural-network-based *set function* to approximate $\mathcal{A}$, which takes in the actual contents of a subset and predicts the trained model. This design enables the learned $\widehat{\mathcal{A}}$ to generalize to subsets involving unseen points. To mitigate overfitting, we propose two novel regularization techniques: the global regularizer encourages that the models predicted by the fitted $\widehat{\mathcal{A}}$ predict the labels in the held-out set as effectively as those obtained from the original $\mathcal{A}$; and the local one ensures that the gradient of the classification or regression learning objective function with respect to these models is close to zero.

#3: Approximability of a learning algorithm with neural networks. When instantiating **ModelPred**, we choose to approximate $\mathcal{A}$ with a neural network. While a neural network seems a reasonable first choice given its popularity in various vision and language tasks, it remains unclear whether a neural network can approximate complex training processes from end to end. This paper presents the *first* formal study of the approximability of a learning process with neural networks. In particular, our results suggest that learning processes associated with strongly convex or smooth problems can be efficiently approximated by neural networks with ReLU units.

#4: Applications of ModelPred. We explore a variety of applications of **ModelPred** towards answering the questions posed at the beginning of the section. Our findings are summarized as follows.

- **ModelPred functions successfully predict trained models from the training data (Table II and III).** The Spearman correlation between the accuracy of the predicted models and the ground-truth is mostly greater than 97%.
- **ModelPred functions enable the discovery of the connection between data and model parameters (Figure 3).** The level of sensitivity of a ground-truth model's parameter to different points matches the results obtained from a predicted model.
- **ModelPred functions successfully boost the accuracy of data valuation—measuring the contribution of individual points to learning (Table V).** While accurately measuring the value of data has a great potential to improve model performance and transparency, being more accurate comes with significant computational costs. In particular, existing data value notions require retraining a target model from scratch on many subsets of the training set and evaluating the corresponding model performance scores. We leverage the ability of **ModelPred** functions to predict models directly from training data and significantly improve the accuracy of data valuation.
- **ModelPred functions lead to improved data selection in the presence of bad data (Figure 5).** We leverage the Shapley value (SV) estimated through **ModelPred** functions to rank the contributions of all training points and select the ones with the highest contributions. **ModelPred** functions lead to a significant improvement in data selection effectiveness while maintaining a similar computation cost to conventional Shapley value estimation methods.
- **ModelPred functions lead to more efficient and accurate quantification of model memorization (Figure 6).** To assess whether a given training point is memorized or not, we evaluate **ModelPred** functions on different subsets with the point included and excluded and then compare the prediction at the point. We find that **ModelPred** functions enable the successful discovery of memorized training samples.
- **ModelPred functions improve the accuracy of prediction confidence (Table VI).** We incorporate the predictions of the model's output by **ModelPred** functions into an ensemble, and it leads to an improvement in the accuracy of prediction confidence on various datasets.

II. Approach

A. Preliminaries

Denote a data point as (x, y), where $x \in \mathbb{R}^d$ is the feature vector and $y \in \mathbb{R}$ is the label. We define $f(x;\theta)$ as the **Base Model**, which is a parametric function that is parameterized by $\theta \in \mathbb{R}^{d_{\text{param}}}$ and maps an input feature to a label (*i.e.*, logistic regression model). The loss function $\ell(\theta;(x,y)) = \ell(f(x;\theta), y)$ is defined to measure the discrepancy between the prediction $\widehat{y} = f(x;\theta)$ and the true label y, which can be a classification or regression learning objective function. Denote the full training dataset containing n training data points as $D = \{(x_i, y_i)\}_{i=1}^n$. We further denote S as a subset with n_S data points sampled from D. A *learning algorithm* (*i.e.*, *solver*) $\mathcal{A}$ is defined as the mapping from an input dataset S to the parameters of the trained model $\widehat{\theta}_S$ *i.e.*, $\widehat{\theta}_S = \mathcal{A}(S)$. A typical example of a learning algorithm is empirical risk minimization (ERM) with ℓ_2 penalty. Specifically, the parameters are optimized to minimize the training loss averaged over the subset S:

$$\widehat{\theta}_S := \operatorname*{argmin}_{\theta} L(\theta; S) = \frac{1}{n_S} \sum_{i \in S} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2} \|\theta\|_2^2, \tag{1}$$

where λ is a hyperparameter that controls the degree of penalization on the ℓ_2-norm of the estimated parameters to increase the model's generalizability.

B. Algorithm

1) Problem Formulation.

As mentioned in Section I, our goal is to predict the outcome of the learning algorithm $\mathcal{A}$ given training set S. While $\mathcal{A}(S)$ is a complicated function that typically involves loss function optimization, we leverage a classic technique in machine learning: picking a suitable function class and finding the function in the class that best approximates the target function. To this end, we construct **ModelPred** as a surrogate function $\widehat{\mathcal{A}}$ to obtain outputs that match $\mathcal{A}(\cdot)$ given an arbitrary training set S.

Definition 1 (**ModelPred** function)**.** *Given a data domain $\mathcal{Z}$, a learning algorithm $\mathcal{A}$, a distribution $\mathcal{D}$ over the space of* data *sets $2^{\mathcal{Z}}$, for any dataset S drawn from $\mathcal{D}$, let $\widehat{\theta}_S = \mathcal{A}(S)$ be the model trained on S using $\mathcal{A}$. The proposed **ModelPred** function is a parametric function $\widehat{\mathcal{A}}$ optimized to predict the $\widehat{\theta}_S$ from a training set $S \sim \mathcal{D}$,* i.e.,

$$\widehat{\mathcal{A}} : 2^{\mathcal{Z}} \to \mathbb{R}^{d_{\text{param}}}, \tag{2}$$

$$\text{where} \quad \widehat{\mathcal{A}} = \operatorname*{argmin}_{\widetilde{\mathcal{A}}} \mathbb{E}_{S \sim \mathcal{D}} \left[L\left(\widetilde{\mathcal{A}}(S), \mathcal{A}(S) \right) \right], \tag{3}$$

where $L(\cdot, \cdot)$ is a loss function.

2) Instantiating

To develop a generic framework that not only can predict a model from the subsets within the training dataset but also from *unseen* subsets drawn from the same distribution $\mathcal{D}$, the contents of input data are preserved to be used as the input for **ModelPred** function, which makes $\mathcal{A}(\cdot)$ a set function. Taking this into consideration, we choose deep neural networks (DNNs) as the function class for **ModelPred** function $\widehat{\mathcal{A}}$ given their strong expressive power. Moreover, since $\mathcal{A}(\cdot)$ is a set function, the architectures of neural networks are restricted to those that are invariant to input permutations. Finding the best neural network that approximates $\mathcal{A}(\cdot)$ becomes a *supervised learning* problem when there are samples of $(S, \mathcal{A}(S))$ available.

3) Design of the Loss Function L

Despite the strong expressiveness, with the large size of parameters, the DNN adopted as the surrogate model can be easily overfitted on the training set if we only push the model to minimize the discrepancy between the predicted and actual model parameters. To prevent the DNN from overfitting, we propose two novel regularization techniques, where the utility loss is proposed to maintain the utility of the predicted model as a global regularizer, while the optimality of the predicted parameters is imposed by the Karuch-Kuhn-Tucker (KKT) loss as a local regularizer.

Global Regularizer. From an overall perspective, an accurately predicted model should maintain a utility close to the ground-truth model when evaluated on any test points, intuitively. For the utility loss as a global regularizer, denote $\mathcal{U}(S; \widehat{\theta})$ as the utility function that evaluates the utility of optimized model parameters $\widehat{\theta}$ on a dataset S. For one training sample, $(\widehat{\theta}_S; S)$, the utility loss is specified as:

$$L_U = \left| \mathcal{U}(S; \widehat{\mathcal{A}}(S)) - \mathcal{U}(S; \widehat{\theta}_S) \right|.$$

Local Regularizer. From a local perspective, an accurately predicted model should satisfy the optimality conditions, which constrains the gradient of the predicted parameters with respect to the learning objective function. With this in mind, we proposed to employ KKT conditions in the loss function to enhance the generalization of **ModelPred** by leveraging the optimality condition.

KKT conditions are found to guarantee the optimality of a solution [11] for convex problems. For an unconstrained convex problem (1), the optimal parameters $\theta_S^* \in \operatorname{argmin}_{\theta \in \mathbb{R}^{d_{\text{param}}}} L(\theta; S)$ should satisfy the stationary KKT condition as follows:

$$\nabla L(\theta_S^*; S) = 0. \tag{4}$$

If the learning algorithm can find a near-optimal solution, then the stationary KKT condition is almost satisfied by the optimized parameter $\widehat{\theta}_S$. Therefore, the KKT loss of one training sample is added as: $L_{KKT} = \left\| \nabla L\left(\widehat{\mathcal{A}}(S); S \right) \right\|$. This condition also applies to convex problems with hard constraints where the KKTstationary condition can be enforced on the Lagrangian of the optimization problem. Besides, the stationary KKT condition can be treated as the first-order optimality condition for nonconvex optimization problems. Therefore, by adding KKT loss, we enhance the generalization of **ModelPred** for generic optimization problems, including neural networks.

Combining the loss on the discrepancy between prediction and groud-truth with proposed regularizers, the total loss of one training sample for the proposed DNN is:

$$L_{\text{DNN}} = \left\| \widehat{\theta}_S - \widehat{\mathcal{A}}(S) \right\| + L_{KKT} + L_U. \tag{5}$$

4) Algorithm Details

The proposed **ModelPred** framework proceeds in two phases: *offline training* and *online estimation* (Figure 1.)

Offline Training. The offline training of **ModelPred** consists of the parameters sampling step and DNN training step. With the set of training samples $\Phi = \{(S_1, \widehat{\theta}_1), (S_2, \widehat{\theta}_2), \ldots\}$, the objective of the offline training phase is to train an effective parameter model which can predict the optimized parameters accurately given new input data points.

The very first question encountered during the training phase is how to generate training subsets and what distribu-

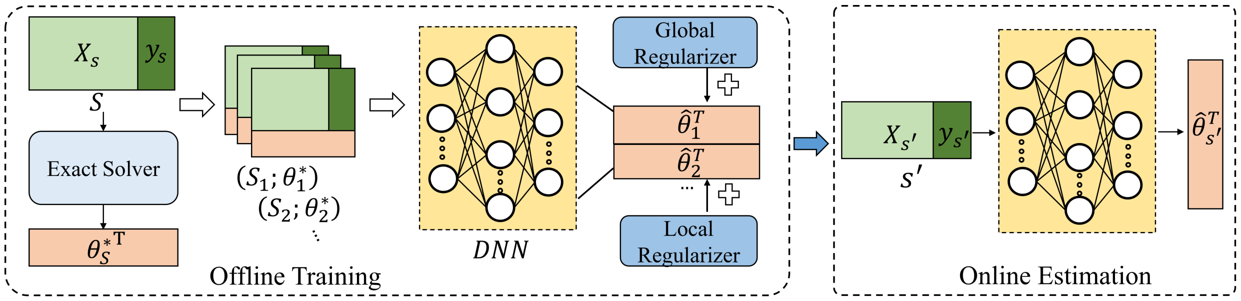

Fig. 1: The offline training phase and online estimate of the proposed **ModelPred**.

tion should be used.

Denote π as a permutation of all training data points from the full set and $D_{\pi[:i]}$ as a set of points that precedes the i-th point in π. We adopt permutation sampling to sample the trained parameters $\mathcal{A}(D_{\pi[:i]})$ for training DNN-based $\widehat{\mathcal{A}}$. The offline training workflow is summarized in Algorithm 1.

Algorithm 1: ModelPred Offline Training

```
input : Full dataset $D = \{(x_i, y_i)\}_{i=1}^n$, learning algorithm
        $\mathcal{A}$, the number of permutations $T$
output: Trained $\widehat{\mathcal{A}}$
1  $\Phi \leftarrow \emptyset$
2  for $t = 1, \ldots, T$ do
3      $\pi_t \leftarrow$ GenerateUniformRandomPermutation($D$)
4      for $i = 1, \ldots n$ do
5          $\widehat{\theta}_i^{\pi_t} = \mathcal{A}(D_{\pi_t[:i]})$
6          $\Phi = \Phi \cup \{(D_{\pi_t[:i]}; \widehat{\theta}_i^{\pi_t})\}$
7      end
8  end
9  Trained $\widehat{\mathcal{A}}$ with $\Phi$
10 return $\widehat{\mathcal{A}}$
```

Compared with other kinds of subset sampling strategies such as sampling uniformly at random, permutation sampling enables the evaluation of the change of model parameters with slightly varied subsets. Therefore, the training samples collected by permutation can better reflect the effect of individual data points, thus, allowing the DNN to better learn their influences in the trained parameters. Thus, permutation sampling is used to generate the training subsets in the offline training of **ModelPred**. The empirical experiment results show a slight advantage of permutation sampling over uniform sampling when we evaluate the performance of the DNN trained by subsets. Note that if the distribution of the subset to be estimated in the testing phase (*i.e.*, test distribution) is known a prior, it is suggested to directly sample the training samples from the test distribution. For instance, in data valuation by SV, the definition of SV requires the distribution to assign uniform probability to all data sizes and uniform probability to all subsets of a given size. Then, in the offline phase, the subsets should be sampled from the distribution above.

Online Estimation. After the offline training phase, the trained DNN $\widehat{\mathcal{A}}$ can be used for efficient model parameters prediction given a new training subset or a batch of subsets through an evaluation of $\widehat{\mathcal{A}}$. Note the subsets are not limited to those containing observed data points.

III. Characterization of Efficiently Approximatable Learning Algorithms

In this paper, we try to use a neural network to fit the function $\widehat{\theta} = \mathcal{A}(D)$. When the context is clear, we omit the learning algorithm and simply write the function as $\widehat{\theta}(D)$. We denote the function associated with kth parameter as $\widehat{\theta}_k(D)$, and thus $\widehat{\theta}(D) = (\widehat{\theta}_1(D), \ldots, \widehat{\theta}_{d_\text{param}}(D))$, where d_param is the number of model parameters. Although D is a set of data points, it can be equivalently viewed as a vector concatenation of all (x_i, y_i). We assume we always normalize input features to $[0, 1]$. Therefore, we have $D \in [0, 1]^{n(d+1)}$. In order to fit $\widehat{\theta}(D)$ with a neural network, the very first question is *whether* $\widehat{\theta}(D)$ *could be efficiently expressed or approximated by neural networks of certain structures.* Here, the efficiency is measured by the total number of computational units in the neural networks.

Despite the strong expressive power of neural networks, not every function could be efficiently approximated by them. Particularly, while the famous universal approximation theorem (UAT) includes many functions, there are requirements regarding the continuity of the functions and compactness of the domain. A recent study [12] shows that *functions* $g : \mathbb{R}^d \to \mathbb{R}$ *that have a smaller upper bound of gradient norm* $\|\nabla g\|$ *could be more efficiently approximated by neural networks with ReLU activation.* As long as $\widehat{\theta}_k(D)$ could be efficiently approximated by certain neural network architectures, we could put d_param such networks in parallel to approximate $\widehat{\theta}(D)$. The parallel networks share the same input but have no internal connections or computational units. Therefore, we can *reduce the problem of understanding the efficient approximability of a function to that one of bounding its gradient norm.* Particularly, this section aims to understand *sufficient* conditions for which $\left\|\frac{\partial \widehat{\theta}_k}{\partial D}\right\|$ could be upper bounded.

Our first result is under the setting that the learning algorithm $\mathcal{A}$ is able to find an *optimal parameter* $\theta^* \in \operatorname{argmin}_\theta L(\theta; D)$. There are three major assumptions in our result: (1) the loss function $\ell(\theta)$ is α-strongly convex, (2) $\left\|\frac{\partial}{\partial\theta\partial z_i}\ell(\theta, z)\right\|$ is upper bounded by some constant B_1, and (3) $\|\theta^*\| \leq B_2$.

Definition 2 (α-strongly convex)**.** *A differentiable function f is α-strongly convex if*

$$f(y) \geq f(x) + \nabla f(x)^T(y-x) + \frac{\alpha}{2}\|y-x\|^2, \quad (6)$$

for some $\alpha > 0$ and for all x, y in the domain.

Since $\ell(\theta; (x, y))$ is α-strongly convex, the training loss $L(\theta; D) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$ is $(\alpha+\lambda)$-strongly convex. Together with the condition that $\|\theta^*\|$ is finite, it implies L has a unique global minimum $\theta^* = \mathrm{argmin}_\theta L(\theta; D)$. Our main result for strongly convex functions is below:

Theorem 1. *If for all $z \in [0,1]^d$, the loss function $\ell(\theta; z)$ is α-strongly convex in θ, and $\left\|\frac{\partial}{\partial\theta\partial z_i}\ell(\theta, z)\right\| \leq B_1$ for $i \in [d]$, then for $\theta_k(D) = [\mathrm{argmin}_\theta L(\theta; D)]_k$, where $L(\theta; D) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$ and $[\cdot]_k$ means the kth element in the vector, then we have*

$$\left\|\frac{\partial\theta_k}{\partial D}\right\| \leq B_1 \frac{\sqrt{d_{\mathrm{param}}(d+1)}}{\sqrt{n}(\alpha+\lambda)}. \quad (7)$$

We then analyze the case where the learning algorithm $\mathcal{A}$ is gradient descent. Gradient descent is a commonly used optimization algorithm in machine learning. Formally, suppose the parameter θ is initialized by 0 and denoted by $\theta^{(0)}$. For the ith iteration, we update $\theta^{(t)} = \theta^{(t-1)} - \eta\nabla L(\theta^{(t-1)}; D)$ with a learning rate η. Given the maximum step number T, we have $\widehat{\theta}(D) = \theta^{(T)}$. The only different assumption for bounding $\left\|\frac{\partial\widehat{\theta}_k}{\partial D}\right\|$ in this case is that the loss function $\ell(\theta)$ is β-smooth instead of α-strongly convex.

Definition 3 (β-smoothness)**.** *A differentiable function f is β-smooth if*

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta\|x-y\| \quad (8)$$

for some $\beta > 0$, and for all x, y in the domain.

Theorem 2. *If for all $z \in [0,1]^d$, the loss function $\ell(\theta; z)$ is β-smooth in θ, and $\left\|\frac{\partial}{\partial\theta\partial z_i}\ell(\theta, z)\right\| \leq B_1$ for $i \in [d]$, then for $\theta_k^{(t)}(D)$ defined by the kth entry of $\theta^{(t)}$, which is iteratively computed by $\theta^{(t)} = \theta^{(t-1)} - \eta\nabla L(\theta^{(t-1)}; D)$ and $\theta^{(0)} = 0$ with a learning rate $\eta > 0$, and $L(\theta; D) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$, we have*

$$\left\|\frac{\partial\theta_k^{(t)}}{\partial D}\right\| \leq \left(1 - (1 - \eta\lambda - \eta d_{\mathrm{param}}\beta)^t\right) \frac{B_1\sqrt{(d+1)}}{\sqrt{n}(\lambda + d_{\mathrm{param}}\beta)}.$$

Theorem 1 and 2 demonstrate that $\left\|\frac{\partial\widehat{\theta}_k}{\partial D}\right\|$ could be upper bounded by the dimension of data domain d and model parameter dimension d_{param}, and the dependency is only $O(\sqrt{d_{\mathrm{param}} d/n})$ for the case of $\mathcal{A}$ being able to find the optimal parameter, and $O(\sqrt{d}/d_{\mathrm{param}}\sqrt{n})$ for the case of gradient descent.

IV. Experiments

A. Experimental setup

1) Computational Considerations

The retraining of a large base model, such as a neural network on a large dataset is computationally intensive. To apply the proposed method efficiently to large base models, we propose to employ the technique of transfer learning, where the weights in all layers except the last one are fixed as a feature extractor [13]. To retrain a neural network on large size subsets, only the last layer in the neural network will be modified and adjusted to the training dataset, This technique has been widely used in natural language processing and computer vision domains [14]. Besides, by removing the data loading bottleneck, *fast neural network training* techniques such as FFCV[15] make it possible to train large amouts of models on diverse subsets in an efficient manner. Our paper does not incorporate such advanced techniques, and we expect that doing so can further improve efficiency.

2) Base Model

To demonstrate that our proposed **ModelPred** is a model-agnostic approach, three kinds of base models (*i.e.*, Logistic Regression (LR), Support Vector Machine (SVM)), and a neural network (Resnet-18)) are chosen, and **ModelPred** will learn to predict the parameters of these models. For LR and SVM, a regularization term with ℓ_2-norm coefficient $\lambda = 1$ is added to the loss function. We adopt L-BFGS-B [16] as the learning algorithm (*i.e.*, solver) for LR and LIBLINEAR [17] as the solver for SVM with a squared hinge loss. Stochastic Gradient Descent (*i.e.*, SGD with learning rate as 0.001, and momentum as 0.9) is adopted as the solver to train NN.

3) Dataset

We evaluate the proposed **ModelPred** on five datasets:

Iris([18]). We use the binary version of Iris dataset with the first two classes of data. The binary version contains 100 samples in total with feature dimension $d = 4$. 67 data points are randomly selected to generate training subsets. The remaining 33 data points are used for SV calculation, and 17 of them are used for dataset addition.

SPAM([19]). SPAM dataset is a collection of spam and non-spam e-mails with feature dimension $d = 215$. 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for SV calculation, and 150 of them are used for dataset addition.

HIGGS([20]). HIGGS dataset is produced using Monte Carlo simulations with feature dimension $d = 30$. After preprocessing, we keep 25 features. 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for SV calculation, and 150 of them are used for dataset addition.

MNIST([21]). MNIST dataset is a collection of grayscale handwritten digits with size 28×28 and 10 classes. We use a CNN in the preprocessing stage to extract the features and reduce the dimension to 128. 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for SV calculation, and 150 of them are used for dataset addition.

CIFAR-10([22]) and Hymenoptera([23]). CIFAR-10 is a collection of 60,000 3-channel images in 10 classes. Hymenoptera is a small 3-channel dataset used to classify ants and bees, which consists of 245 training images and 153 testing images. These two datasets are used for applying **ModelPred** to large NNs by transfer learning. In the experiment, we fine-tune the weights of the last layer of a pre-trained Resnet-18 model on CIFAR-10 dataset for the binary classification of hymenoptera images and use **ModelPred** to estimate the parameters of the new model. In particular, a Resnet-18 model is pre-trained on CIFAR-10. The dimension of the weight in the last layer is 512 (*i.e.*, $d = 512$). 208 images are randomly selected to generate training subsets and the remaining 190 are treated as the testing set, where 100 of them are used for dataset addition.

All the input data x are normalized to rescale the norm $\|x\|$ to be within the range of $[0, 1]$

4) Sampling Distribution

15000 subsets are sampled from each dataset following permutation sampling procedures in Algorithm 1 to construct the training sample set Φ.

5) Proposed Network Structure

We adopt a canonical model architecture DeepSets [24] for proposed DNN. DeepSets is designed to be permutation invariant of the input samples, which is fit for set function learning (*i.e.*, mapping a set of samples to a target output). A DeepSet model is a *set* function $f(S) = \rho(\sum_{x \in S} \phi(x))$ where both ρ and ϕ are neural networks. In the experiment, both ρ and ϕ networks have 3 fully-connected layers with 128 neurons in each layer.

6) Evaluation Metrics and Baseline

To evaluate the performance of the proposed **ModelPred**, baseline comparisons are conducted in two main scenarios: *dataset deletion* and *dataset addition*, where **ModelPred** is used to predict the model on deleted or added datasets. During the deletion, the size of training subsets is decreased from 100% of the full size to 50% (i.e, for SPAM, from 300 to 150). During the addition, the size of training subsets is increased from 50% of the full size to 100% (i.e, for SPAM, from 150 to 300). Notably, the added samples are unseen during the offline training phase. In both scenarios, 10 subsets are generated randomly with each size. To illustrate the generalizability of the proposed **ModelPred** to new data points from the same distribution, previously unseen data points are incorporated to form a larger subset in the dataset addition scenario. Table I details the size of the added or deleted subsets for each dataset for dataset deletion and dataset addition.

In these two scenarios, **Influence Function** is selected as one baseline. With the ability to approximate the effect of single data point deletion or addition on model parameters, Influence Function can also be extended to evaluate the subset change [10]. **Datamodel** [1] is selected as another baseline only for dataset deletion since it cannot make predictions for newly added training points. Besides, DeltaGrad [8] can also serve as a baseline for batch deletion or addition. However, it requires the application of SGD to optimize the base model, which cannot provide the optimal solution in a timely manner and cannot solve the primal problem of SVM accurately. Therefore, we do not include DeltaGrad in the experiments. Additionally, we conduct the ablation study by creating **ParaLearn** as another baseline. **ParaLearn** has the same neural network structure as **ModelPred** and also predicts the optimized parameter for a convex learning model. The difference is that it does not include local and global regularizers.

Three metrics are adopted to evaluate the effectiveness of the proposed method: 1) **the Euclidean distance** between the estimated model parameters and the exact parameters θ^* provided by the solver, *i.e.*, $\left\|\widehat{\theta} - \theta^*\right\|$; 2) **the Normalized-Root-Mean-Squared Error (NRMSE)** of the estimated utility calculated by the estimated model parameters; 3) **the Spearman rank-order correlation** ([25]) between the ground-truth utility and the estimated utility on the testing set. The first metric evaluates the accuracy of parameter estimation. Here, we use the parameters optimized by retraining the model with the solver as the ground-truth. We adopt the Spearman correlation in addition to the NRMSE because most of the applications of subsets utility [26] desire that the utility is ranked in the correct order.

7) Machine configuration

We run experiments with one Intel(R) Xeon(R) Gold 5218 CPU and use one GeForce RTX 2080 Ti for DNN training and inference.

B. Experimental results

*1) **ModelPred** can predict training outcomes*

Table II and III summarize results of proposed **ModelPred** and the baseline in scenarios of dataset deletion and addition, respectively. LR is used as the base model for the first four datasets while NN is used for the last Hymenoptera dataset. First, we focus on the results with LR as the base model. In general, **ModelPred** demonstrates a significant advantage in the accurate prediction of parameters if we compare $\left\|\widehat{\theta} - \theta^*\right\|$. **Comparing the results of different datasets, the Euclidean distance grows slightly with the dimension of the input feature dimension, but the estimated utility is unaffected.** It can also be shown the predicted parameters well maintain the utility by checking the NRMSE of utility and the Spearman correlation (*i.e.*, over 95% for most datasets) between the estimated utility and ground-truth. However, it is seen that both the parameter prediction accuracy and the utility estimation accuracy are relatively low on the HIGGS dataset. This might be caused by the low performance of the base model (*i.e.*, LR) on this dataset, which generates similar optimal parameters as training samples in the training phase.

By investigating the performance of three baselines, although **ParaLearn** outperforms the Influence Function on

TABLE I: Setting for Dataset Deletion and Dataset Addition.

Dataset	Total Training Data Points	Total Testing Data Points	Dataset Deletion			Dataset Addition		
			Size of Starting Subset	Size of Deleted Subsets	Size of Ending Subset	Size of Starting Subset	Size of Added Subsets	Size of Ending Subset
Iris	67	33	63	[5, 10, 15, …, 30]	33	67	[1, 2, 3, …, 17]	84
SPAM, HIGGS, MNIST	300	500	300	[5, 10, 15, …, 150]	150	150	[5, 10, 15, …, 150]	300
Hymenoptera	208	190	208	[5, 10, 15, …, 100]	108	108	[5, 10, 15, …, 100]	208

TABLE II: A summary of **ModelPred** results and baseline comparison results in the scenario of dataset deletion with LR (*i.e.*, for Iris, SPAM, HIGGS, and MNIST) and NN (*i.e.*, for Hymenoptera) as the base model. Mean and standard deviation reported over 10 experimental trials. The best result is highlighted in **bold**.

Dataset	Algorithm	Parameter		Utility			
		$\left\|\widehat{\theta}-\theta^*\right\|$	Std	NRMSE	Std	Spearman Corr	Std
Iris	**ModelPred**	**9.67E-02**	1.51E-02	**2.80%**	**0.42%**	**0.9964**	**0.0107**
	Influence function	8.97E-01	**1.39E-02**	74.62%	16.98%	0.4071	0.4262
	ParaLearn	1.95E-01	2.49E-02	9.47%	2.61%	0.9679	0.0297
	Datamodel	N/A	N/A	22.78%	5.72%	0.9571	0.0474
SPAM	**ModelPred**	**3.72E-01**	1.27E-02	**6.48%**	**1.19%**	**0.9856**	**0.0040**
	Influence function	1.07E+00	**3.40E-03**	50.60%	3.70%	0.1841	0.1797
	ParaLearn	2.36E+00	2.15E-02	125.55%	10.62%	N/A	N/A
	Datamodel	N/A	N/A	69.53%	9.98%	0.6174	0.0612
HIGGS	**ModelPred**	**2.41E-01**	1.77E-02	**12.58%**	**3.86%**	**0.7980**	**0.0662**
	Influence function	3.32E-01	**5.11E-03**	22.68%	6.11%	0.7504	0.1275
	ParaLearn	5.11E-01	2.89E-02	65.30%	17.03%	N/A	N/A
	Datamodel	N/A	N/A	167.83%	62.34%	0.3414	0.1412
MNIST	**ModelPred**	**9.92E-01**	9.60E-03	**2.28%**	**0.42%**	**0.9978**	**0.0010**
	Influence function	1.04E+01	6.65E-01	48.85%	2.81%	0.0278	0.2375
	ParaLearn	4.58E+00	**4.96E-03**	113.94%	7.11%	N/A	N/A
	Datamodel	N/A	N/A	46.30%	3.12%	0.9866	0.0058
Hymenoptera	**ModelPred**	**1.18E+00**	1.14E-02	**6.54%**	**0.53%**	**0.9920**	**0.0025**
	Influence function	1.24E+00	**6.24E-03**	56.43%	2.71%	0.03623	0.13195
	ParaLearn	2.04E+00	1.16E-02	73.97%	4.05%	N/A	N/A
	Datamodel	N/A	N/A	51.83%	4.86%	0.8042	0.0622

Iris dataset, the Spearman correlation for other datasets cannot be calculated. This is because it provides constant prediction regardless of the input subset change, resulting in constant utility for all subsets. **This result indicates the proposed KKT and utility loss effectively prevent overfitting.**

We visualize the result on MNIST in Figure 2. The x axis of Figure 2 a) and b) is the ground-truth utility (*i.e.*, testing loss) and the y axis shows the estimated utility. The red line represents the ground-truth utility and each circle represents the estimated utility of one subset, where the circle size is proportional to the subset size. It clearly shows that the error of loss predicted by Influence Function increases with the scale of the change on a dataset (either deletion or addition) since the utility estimated by Influence Function rapidly deviates from the red line, whereas **ModelPred** consistently provides accurate utility prediction.

Figure 2 c) and d) demonstrate the Euclidean distance $\left\|\widehat{\theta}-\theta^*\right\|$ with error bar where the x axis in c) is the size of the training subset after addition and the x axis in d) is the size of the remaining training subset after deletion, both of which are formatted as the percentage of the size of the full training set (*i.e.*, $n = 300$ for MNIST). From Figure 2 c) and d), it is shown that Influence Function can make an accurate approximation of the parameter change with a small scale of change (*i.e.*, at the beginning of data addition and deletion). However, the distance grows rapidly with the scale of change. The first-order Taylor approximation adopted in Influence Function cannot accurately estimate the effect of a large group of data points on the base model. The inferior performance of Datamodel is also caused by its nature of linear approximation by linear regression, which cannot accommodate large dataset changes.

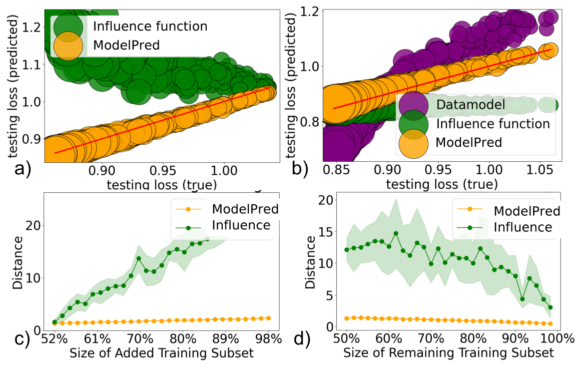

Fig. 2: Results on MNIST dataset with LR as the base model. a) and b): the estimated loss by predicted parameters of each subset in the scenario of dataset deletion and addition; c) and d): the Euclidean distance between predicted parameters and optimal parameters in the same scenario.

For better readability, we defer the results of two scenarios with SVM as the base model to the Appendix, which

TABLE III: A summary of **ModelPred** results and baseline comparison results in the scenario of dataset addition with LR as the base model. Mean and standard deviation reported over 10 experimental trials. The best results are highlighted in **bold**.

Dataset	Algorithm	Parameter		Utility			
		$\left\lVert\widehat{\theta}-\theta^*\right\rVert$	Std	NRMSE	Std	Spearman Corr	Std
Iris	**ModelPred**	**1.34E-01**	**8.81E-03**	**5.10%**	**0.52%**	**0.9882**	**0.0067**
	Influence function	3.56E-01	6.13E-04	63.07%	3.15%	0.0779	0.1521
	ParaLearn	2.85E-01	4.22E-03	43.91%	2.83%	0.9730	0.0161
SPAM	**ModelPred**	**5.96E-01**	**4.56E-03**	**10.22%**	**0.82%**	**0.9919**	**0.0026**
	Influence function	1.11E+00	2.34E-03	65.44%	2.26%	0.1886	0.1683
	ParaLearn	2.19E+00	2.10E-02	145.73%	4.36%	N/A	N/A
HIGGS	**ModelPred**	**3.26E-01**	**1.56E-02**	**14.11%**	**2.71%**	**0.8054**	**0.0684**
	Influence function	4.58E-01	3.23E-03	20.40%	4.27%	0.9008	0.0381
	ParaLearn	1.92E+00	1.86E-02	838.62%	128.59%	N/A	N/A
MNIST	**ModelPred**	**1.81E+00**	**1.22E-02**	**5.46%**	**0.25%**	**0.9969**	**0.0009**
	Influence function	1.31E+01	4.75E-01	125.26%	4.22%	-0.8431	0.0367
	ParaLearn	4.76E+00	6.01E-03	131.73%	2.17%	N/A	N/A
Hymenoptera	**ModelPred**	**1.94E+0**	1.57E-02	**17.45%**	0.58%	**0.70000**	**0.09458**
	Influence function	7.11E+00	2.81E-02	558.29%	8.77%	0.35000	0.14044
	ParaLearn	2.25E+00	**1.37E-02**	27.13%	**0.24%**	N/A	N/A

demonstrate the similar performance of **ModelPred** in terms of efficiency and effectiveness.

By checking the performance of applying **ModelPred** on transfer learning in NN. similar conclusions can be drawn compared to those of LR. Firstly, **ModelPred** can accurately predict the high-dimensional parameters for the transfer learning of NN. Secondly, the utility of the refitted NN can be accurately estimated by the proposed method, which indicates the effectiveness and generalizability brought by the proposed regularizers. Thirdly, the parameter prediction error is larger than the results of convex base models (*i.e.*, Table II and III), which might be due to the stochastic training process of NN.

C. Applications

We leverage **ModelPred** to a variety of ML applications to answer questions mentioned in Section I.

Which point is the most responsible for learning a given parameter in the model?

ModelPred Learns Learning Patterns To demonstrate the effectiveness of neural networks in learning the mappings between training data and the final parameters, we plot the saliency map of model parameter changes when a specific data point is excluded ("turned off") from the training set. In detail, LR is selected as the base model, and 300 data points from MNIST are used as the full training dataset. An LR model with L1 penalty [27] is firstly trained on the full training set to identify the significant parameters. Then, a subset of parameters is randomly selected for visualization. We flip the label of the first 20 samples in each training set to investigate the effect of noisy data on the fitted model, as well as to examine the generalizability of the proposed method. Note that during the training process of **ModelPred**, we do not include the flipped samples.

By checking the second row of four parameters in the left panel of Figure 3, it shows the color of the reordered samples gradually changes from blue to red, which indicates **ModelPred** can accurately predict relative parameter change caused by removing individual samples. By comparing the color of the boxed samples on the right panel of Figure 3, the parameter changes caused by noisy data (*i.e.*, first 20 samples) are more significant than others.

We further investigate the average value of the parameter changes caused by the exclusion of noisy and good samples and summarize them in Table IV. By checking the average parameter change estimated by Solver, we find a systematic difference between noisy samples and good samples. Removing noisy samples causes a decrease in the parameters while removing good samples leads to an increase. The change estimated by **ModelPred** is consistent with this pattern under most scenarios. This demonstrates the capability of **ModelPred** in accurately capturing the mapping from the training data to the learned model as well as its generalizability on noisy samples.

What are the training points with the most or least contribution to the model?

Data Valuation. Quantifying the value of each training data point to a learning task is a fundamental problem in ML. SV is a widely used data value notion nowadays to identify the contribution of each training point [3, 28, 2, 29, 30]. However, the exact SV calculation for a training dataset with n points involves computing the marginal utility of every point in all subsets, which requires 2^n times of utility evaluation by retraining the model. Since **ModelPred** is capable of efficient utility evaluation by predicting the model parameters on a new training subset, **ModelPred** can speed up the SV calculation.

To validate the performance of **ModelPred** on calculating SV, we use permutation sampling [31] to generate the subsets evaluated from the full training set D in sequence. We compare the SV results calculated by **ModelPred**, and **UtlLearn** (*i.e.*, predicted SV) with the SV calculated by the solver as ground truth, and we also record the total time consumed

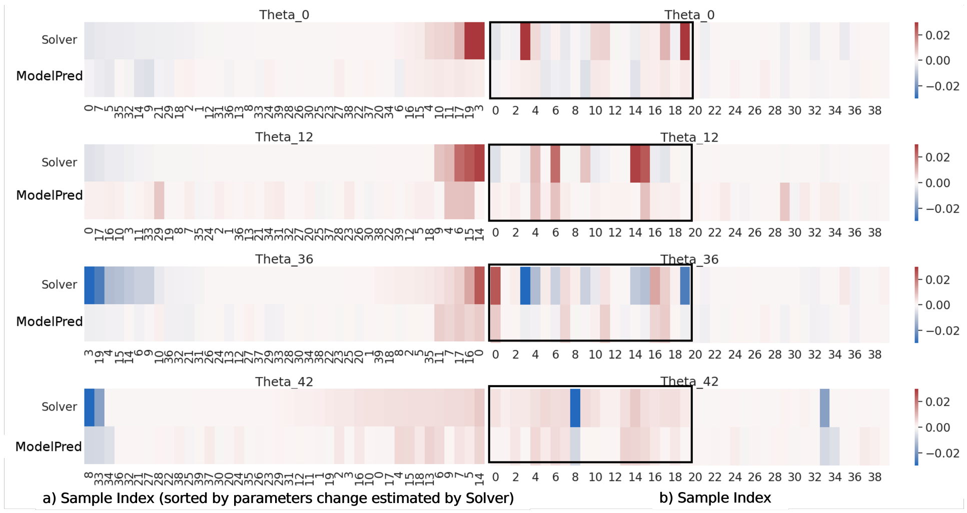

Fig. 3: Saliency map of model parameter changes estimated by the solver and **ModelPred** with LR as the base model on MNIST dataset. a) Samples are ordered by the corresponding value of parameter changes estimated by the Solver. b) Samples are ordered by the original index. Flipped sample indexes are boxed.

TABLE IV: Average parameter changes caused by the exclusion of noisy and good samples.

Method	Theta_0		Theta_12		Theta_36		Theta_42	
	noisy samples	good samples	noisy samples	good samples	noisy samples	good samples	noisy samples	good samples
Solver	-2.17E-03	5.27E-03	-6.86E-03	4.70E-03	-6.86E-03	3.32E-03	-2.81E-03	3.74E-03
ModelPred	-1.27E-03	9.68E-04	9.64E-04	1.66E-03	-1.28E-03	1.47E-03	-5.59E-04	2.28E-03

by each method for subset utility evaluation. Here, we create **UtlLearn** as a baseline which has the same neural network structure as **ModelPred** but directly predicts the utility of a subset, and thus, not including KKT loss. We set the number of permutations $T \in \{10, 50, 100, 500, 1000\}$ for Iris, SPAM, and HIGGS. For MNIST, we set $T \in \{10, 50, 100, 200\}$ due to the long computation time of the solver caused by high feature dimension. Similarly, we use NRMSE and Spearman correlation as performance metrics to evaluate the predicted SV.

Table V summarizes the results for Iris, SPAM, and HIGGS. It can be observed that **ModelPred** outperforms **UtlLearn** in terms of SV prediction under all scenarios. Compared to **UtlLearn**, **ModelPred** has more network parameters to train due to the higher output dimension d_{param}. However, **ModelPred** gains its advantage from the KKT regularization which enforces the optimality of predicted parameters, thus, achieving accurate parameter and utility prediction.

Additionally, comparing the results of a varied number of permutations, it is shown that the Spearman correlation tends to increase with the number of permutations T. When T is small, the ground-truth SV calculated by permutation sampling is very sensitive to the utility of small subsets. Therefore, the relatively inaccurate prediction on small subsets results in the high NRMSE and low Spearman correlation. With the increase of T, the SV predicted by **ModelPred** can better represent the true SV with the increasing correlation.

Figure 4 presents the trends of Spearman Correlation of predicted SV by **ModelPred**, and compares the total computation time of SV calculation with the solver on MNIST and HIGGS. It demonstrates the advantage of **ModelPred** over the solver in computation time, where time consumed by **ModelPred** has a slow growth with the increasing number of permutations.

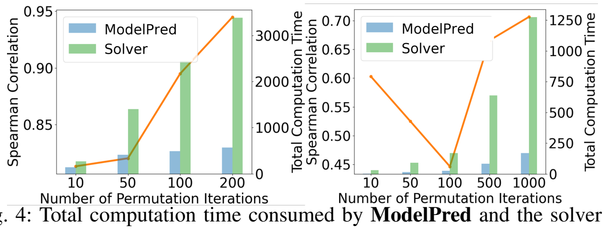

Fig. 4: Total computation time consumed by **ModelPred** and the solver on training a different number of subsets in SV calculation for MNIST (left) and HIGGS (right).

How to select data that benefit model performance?

Data Selection. Despite the rapid growth of big data, the performance of all learning algorithms is upper bounded by the quality of training data [32]. Low-quality training data could be attributed to the contamination of various harmful examples (*i.e.*, noisy samples, mislabeled samples) or the lack of representativeness of the data distribution. Furthermore, a small but high-quality training dataset can significantly reduce the computation workload of an ML model as well as maintain comparable utility.

In this application, we aim to efficiently identify the quality of training data by estimating the Shapley value of each data point. We estimate the SV of training samples in SPAM and MNIST by **ModelPred** with LR as the base model (*i.e.*, **ModelPred**-SV). The label of 10% training samples

TABLE V: A summary of **ModelPred** results and baseline comparison results in SV prediction with LR on IRIS, SPAM, and HIGGS with permutation number $T \in \{50, 100, 500, 1000\}$. Mean and standard deviation reported over 10 experimental trials. The best result is highlighted in **bold**.

Permutation		50			100			500			1000		
		Shapley value		Time (sec)	Shapley value		Time (sec)	Shapley value		Time (sec)	Shapley value		Time (sec)
Dataset	Algorithm	NRMSE	Spearman		NRMSE	Spearman		NRMSE	Spearman		NRMSE	Spearman	
Iris	**ModelPred**	**8.52%**	**0.9265**	**1.05E+01**	**7.69%**	**0.9463**	2.22E+01	**6.82%**	**0.9747**	**7.84E+01**	**6.22%**	**0.9585**	2.27E+02
	UtlLearn	25.97%	-0.2526	1.08E+01	28.46%	-0.1118	**2.21E+01**	36.14%	0.0388	7.87E+01	42.79%	-0.0480	**2.22E+02**
	solver	–	–	4.05E+01	–	–	8.64E+01	–	–	2.70E+02	–	–	8.57E+02
SPAM	**ModelPred**	**13.46%**	**0.3774**	**1.00E+02**	**18.16%**	**0.3686**	**1.23E+02**	**15.57%**	**0.7069**	1.43E+02	**18.51%**	**0.7772**	2.86E+02
	UtlLearn	15.21%	-0.2081	1.14E+02	20.76%	-0.2091	1.26E+02	18.72%	-0.1187	**1.42E+02**	22.96%	-0.0600	**2.75E+02**
	solver	–	–	3.07E+02	–	–	6.50E+02	–	–	8.96E+02	–	–	4.73E+03
HIGGS	**ModelPred**	**12.15%**	**0.5249**	1.37E+01	**15.60%**	**0.4463**	**2.42E+01**	**17.54%**	**0.6655**	**8.30E+01**	**17.45%**	**0.7066**	**1.69E+02**
	UtlLearn	43.12%	-0.2494	**1.27E+01**	61.82%	-0.3029	2.15E+01	86.85%	-0.5729	1.02E+02	90.52%	-0.6742	1.75E+02
	solver	–	–	9.17E+01	–	–	1.70E+02	–	–	6.39E+02	–	–	1.27E+03

is flipped in the experiment. Then, the samples are removed from the training set according to their SV from the smallest to the largest. The base model is retrained on the shrunk training set to obtain classification accuracy on the testing set. Shapley value estimated by Permutation Sampling with L-BFGS-B solver (*i.e.*, Perm-SV), and randomly selecting the sample to be removed (*i.e.*, Random) are compared with **ModelPred** as baselines. To maintain similar computation time by **ModelPred**-SV and Perm-SV, 1000 and 50 permutations are performed to calculate SV, respectively. Figure 5 shows that after removing the sample number of samples, **ModelPred** maintains a higher testing accuracy than the baselines while Perm-SV outperforms Random in most cases. This indicates that SV can effectively identify the quality of a data point, and **ModelPred**-SV can estimate the SV more accurately with a larger number of permutations. Figure 5 also shows that there is an increasing trend of the testing accuracy on both SPAM and MNIST in the initial phase by **ModelPred**-SV, which indicates that the mislabeled low-quality data are removed effectively to improve the model's learning performance.

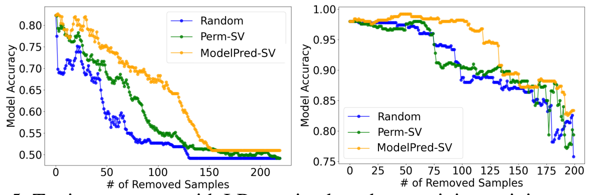

Fig. 5: Testing accuracy with LR retrained on the remaining training samples for SPAM (left) and MNIST dataset (right).

Is a training data point memorized by the model?

Memorization and Influence Score Estimation. One of the most important features of ML algorithms is their capability for generalization. Label memorization score has been proposed to identify and understand the utility of typical, atypical samples and outliers for a model in order to understand the generalization [33]. For a model $f(x;\theta)$ trained on a dataset D by a learning algorithm $\mathcal{A}$, the label memorization score on sample $(x_i, y_i) \in D$ is defined as:

$$\begin{aligned}\operatorname{mem}(\mathcal{A}, D, i) := \operatorname{Pr}_{f \leftarrow \mathcal{A}(D)}\left[f\left(x_i\right)=y_i\right]- \\ \operatorname{Pr}_{f \leftarrow \mathcal{A}(D \backslash i)}\left[f\left(x_i\right)=y_i\right],\end{aligned} \tag{9}$$

where $D\backslash i$ denotes the subset D with (x_i, y_i) removed.

Following [33], we adopt the *subsampling-based estimation algorithm* to estimate the memorization score. Specifically, we sample m subsets of equal size $|S_i| = 0.7|D|$ and the memorization score can be estimated as the following:

$$\begin{aligned}\widehat{\operatorname{mem}}(\mathcal{A}, D, i) = \\ \frac{1}{|\{j : x_i \in S_j\}|} \sum_{j: x_i \in S_j} I[f\left(\mathcal{A}\left(S_j\right), x_i\right) = y_i] \\ - \frac{1}{|\{j : x_i \notin S_j\}|} \sum_{j: x_i \notin S_j} I[f\left(\mathcal{A}\left(S_j\right), x_i\right) = y_i].\end{aligned}$$

However, with a large dataset D, the number of subsets m is substantially large to accurately estimate the label memorization of each sample [33]. This computation constraint can be partially overcome by the **ModelPred** since it can efficiently estimate the trained model on a subset, thus capable of accelerating the memorization score estimation.

In the experiment, two base models (NN and LR) with two datasets (Hymenoptera and MNIST) are employed for memorization estimation. As mentioned in Section IV-B, the NN is pre-trained on CIFAR-10 and the last layer is trained to adjust to Hymenoptera. In order to maintain comparable computational time for the comparison between **ModelPred** and the SGD solver, 1000 subsets are randomly generated for each sample with a size of 280 for **ModelPred**, and 50 subsets are generated for the SGD solver to estimate the memorization of 208 training samples in Hymenoptera. In order to demonstrate and compare the estimation accuracy of memorization scores, we depict the effect of removing memorized samples on test accuracy. Specifically, in the left panel of Figure 6, we show the model test accuracy when data points whose memorization scores are higher than a certain threshold are excluded from the training set. As we can see, memorization scores estimated by **ModelPred** result in higher test accuracy compared with the baseline estimator, which means that **ModelPred** can improve memorization score estimation and thus better identify outliers.

When adopting LR as the base model and MNIST as the dataset, memorization scores of the first 200 samples are estimated by L-BFGS-B (*i.e.*, the solver for LR) and **ModelPred** on 500 testing samples. Since the learning algorithm, in this case, is deterministic, we modify the definition of memorization score as the following:

$$\begin{aligned}&\mathrm{mem}(\mathcal{A}, D, i) \\ &:= \mathrm{conf}(f_{\text{include}}(x_i), y_i) - \mathrm{conf}(f_{\text{exclude}}(x_i), y_i),\end{aligned} \tag{10}$$

where $f_{\text{include}} = \mathcal{A}(D)$ and $f_{\text{exclude}} = \mathcal{A}(D \setminus i)$, and $\mathrm{conf}(f, y)$ denote the confidence score of f on class y.

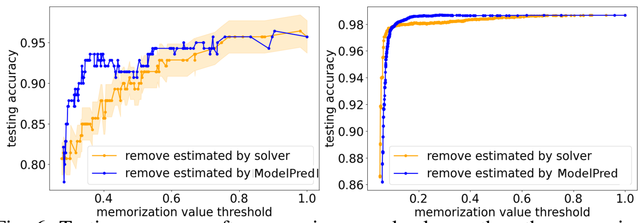

Fig. 6: Testing accuracy after removing samples larger than the memorization value threshold (*i.e.*, ground-truth and estimated by **ModelPred**) on Hymenoptera with pre-trained NN (left) and on MNIST with LR (right). Mean and standard deviation reported over 10 experimental trials.

The right panel of Figure 6 demonstrates the effect of removing samples with the memorization values estimated by **ModelPred** and the solver, which can be treated as the ground-truth memorization value, on the test accuracy of MNIST dataset. As shown by Figure 6, the trend of the removal effect estimated by **ModelPred** is very close to the ground-truth trend. Compared to the solver, it takes 1.84% of the time to estimate the value by **ModelPred**, which greatly reduces the computation workload.

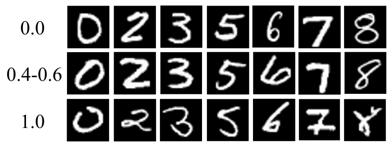

Fig. 7: Examples of memorization values estimated by **ModelPred** from the first 200 samples of MNIST class 0, 2, 3, 5, 6, 7, 8.

Figure 7 visualizes samples with memorization values estimated by **ModelPred** around 0, 0.5, and 1, respectively, which shows the effectiveness of **ModelPred** in identifying memorized examples. Intuitively, samples with an estimated memorization value of 0 are relatively normal, whereas those with a value of 1 are atypical (e.g., highly unclear or incorrectly categorized). Therefore, samples with high memorization values should be carefully validated to be included in the training dataset.

Furthermore, we present pairs of examples with influence estimated by **ModelPred** from high to low in Hymenoptera as shown in Figure 8. The reuslts on MNIST is included in the Appendix. The influence of a training sample (x_i, y_i) on a testing sample $z = (x, y)$ is measured as:

$$\begin{aligned}\mathrm{infl}(\mathcal{A}, D, i, z) := \; &\Pr_{f \leftarrow \mathcal{A}(D)}[f(x) = y] - \\ &\Pr_{f \leftarrow \mathcal{A}(D \setminus i)}[f(x) = y].\end{aligned} \tag{11}$$

As shown in Figure 8, high-influence pairs correspond to similar (or even near-duplicated) images.These samples in the test set benefit mostly from the high-memorized training samples due to the long-tail distribution. In contrast, examples in pairs with lower influences are more regular. This validates the effectiveness of the influence memorization value estimated by the proposed **ModelPred**, which helps to better interpret the influence of samples and the generalization of ML algorithms.

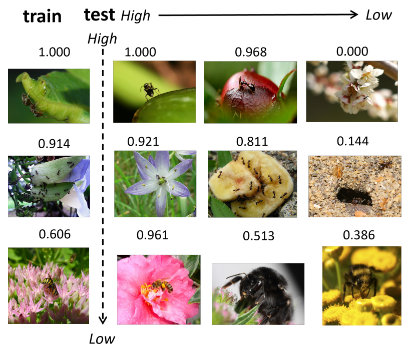

Fig. 8: Examples of selected influence pairs in Hymenoptera. The left column shows the examples with memorization from high to low in the training set. For each training example, examples with high and low estimates are presented in the testing set.

How to produce accurate estimates of uncertainty on model predictions?

Model Calibration. Classification models' confidence calibration performance is crucial in mission-critical tasks [34], and a well-calibrated model should have a confidence (*i.e.*, probability associated with the predicted class label) matching with its ground truth accuracy. Expected Calibration Error (ECE) [35] is the primary metric of model calibration performance, which is calculated by partitioning predictions (with range [0, 1]) into M equally-spaced bins and calculating the weighted average of differences between bins' accuracy and confidence:

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\mathrm{acc}\left(B_m\right) - \mathrm{conf}\left(B_m\right)\right|,$$

where n is the number of training samples, B_m is the set of indices of samples whose prediction confidence falls into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M})$ for m $\in$ {1,. . . , M}. Accuracy and confidence are calculated below:

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}[\widehat{y}_i = y_i], \tag{12}$$

$$\mathrm{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \widehat{p}_i, \tag{13}$$

where $\widehat{y}_i$ and y_i are the predicted and true class labels of sample i, respectively, and $\widehat{p}_i$ is the confidence for sample i on label y_i. A model with better calibration performance should have a lower ECE.

One way to improve a model's calibration performance is bagging [36], which generates an ensemble of models on subsampled datasets and aggregates over each prediction to obtain final prediction results. However, training a large

TABLE VI: Expected Calibration Error (ECE). 'Regular' refers to an ensemble of LR models obtained during the training of $\widehat{\mathcal{A}}$; '**ModelPred**' refers to an ensemble of a combination of these models and 5000 models generated by the trained $\widehat{\mathcal{A}}$.

Dataset	Regular	**ModelPred**
Iris	0.0562	**0.0508 ± 0.0002**
SPAM	0.2693	**0.1661 ± 0.0054**
MNIST	0.5492	**0.5227 ± 0.0014**

number of models can be time-consuming. We show here that **ModelPred** provides an efficient way to perform bagging, which effectively reduces the ECE. Specifically, we use LR base models as a baseline of the ensemble. Further, we combine these models with models predicted by the trained DNN, $\widehat{\mathcal{A}}$, to get a larger ensemble. To generate models using $\widehat{\mathcal{A}}$, each time we randomly sample a subset of instances with a ratio 0.6 from the training set with replacement and obtain estimated models predicted by $\widehat{\mathcal{A}}$. As shown in Table VI, combining more models generated by $\widehat{\mathcal{A}}$ effectively lower the ECE on Iris, SPAM, and MNIST dataset. Additionally, temperature scaling [34] might further improve the model's calibration performance and we consider investigating the combination of temperature scaling and our method with model ensemble as the future work.

V. Related Work

Rapid Model Parameter Approximation. Machine unlearning and incremental model maintenance provide post-hoc techniques to estimate model parameters without retraining from scratch. Ginart et al. [37], Nguyen et al. [38], Brophy and Lowd [39] have investigated unlearning strategies on specific types of learning algorithms, whereas Bourtoule et al. [40] provides generally applicable strategies. For simple linear models, Cauwenberghs and Poggio [41], Schelter [42], Wu et al. [43] introduce incremental model maintenance techniques for efficient updating a model for both data deletion and addition. In DNNs, Influence Function [6] is proposed to measure the effect of a manipulated data point by using Taylor expansion to approximate model parameters. DeltaGrad [8] saves optimizer's update steps in order to more accurately approximate the removal of multiple sample points. Golatkar et al. [44] used the approximation of the Fisher Information Matrix for the remaining data sample to measure the update step to modify the model parameters. Although all techniques are capable of efficient model parameter approximation for a change with a small number of samples [45, 46], they are not scalable for situations when a large number of training points are altered.

Learning to Optimize. L2O focuses on learning the optimization algorithm (*i.e.*, optimizer). An early approach was provided in Andrychowicz et al. [47] to use Recurrent Neural Networks (RNN) to learn the optimizer. For larger model training, it requires the RNN model to iterate through more time steps. Unfortunately, that will create a vanishing gradient or an exploding gradient of the RNN optimization, which in turn will render unstable training of L2O. Due to these barriers, more works [48, 49, 50, 51, 52, 53] focus on overcoming those problems by improving the LSTM structure. Even though, those works focus mainly on specific optimizers' family, such as SGD, Adam, or RMSProp. As another branch, Li and Malik [54] proposed a reinforcement learning (RL) technique, in which the RL policy is the update step of the optimizer and the reward is the loss for the optimizer. However, RL methods are not scalable. Therefore, Almeida et al. [55] proposed to update the optimizer's hyperparameters instead of learning to update parameters, which helps to improve the generalization ability of L2O models. Despite the advantages of fast optimization and potential generalizability, L2O learns to learn and optimize the optimizer, which is computationally intensive to be applied to repeated model training on a large number of subsets.

Learning Optimization by DNNs. A line of research studies using deep learning to solve convex optimization problems by encoding constraints and dependencies into network structure to find the optimal end-to-end mapping. Agrawal et al. [56] embedded disciplined convex optimization problems as differentiable layers within DNN architectures as a new solver. Amos and Kolter [57] introduced a network architecture to solve differentiable optimization by end-to-end deep learning training. Similar to L2o, repeated training is still computationally demanding with these approaches. Moreover, they also suffer from poor generalization performances, as they are limited to linear programs (LP), and quadratic programs, and are difficult to be applied to other settings. Chen et al. [58] proposed to use DNN to predict the optimal set of active constraints to improve the generalizability, but only for the form of the linear program. Carlini et al. [59] trained a set of models on the subset of the training dataset to perform the membership inference attack on a target model. However, they did not use such a technique to understand the input-output behavior of a learning algorithm, which is the focus of our work.

VI. Conclusion

We propose **ModelPred**, as a framework to analyze the dependence of the trained models on training data via supervised learning. We introduce two novel regularization techniques to prevent overfitting the context of predicting models from training data. We show that the expressiveness of DNNs makes them suitable for approximating the input-output behavior of a learning algorithm. We showcase the applications of **ModelPred** to build trust in ML.

For future work, it is intriguing to rigorously study the learnability of the mapping from data to the trained model using neural networks. Moreover, while providing promising results, our current design simply examines the ℓ_2 distance between the ground-truth and the predicted model parameters. It is interesting to explore more sophisticated ways to measure parameter distance that accounts for the difference in predictive behaviors of the two models.

References

[1] A. Ilyas, S. M. Park, L. Engstrom, G. Leclerc, and A. Madry, "Datamodels: Predicting predictions from training data," *arXiv preprint arXiv:2202.00622*, 2022.

[2] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. J. Spanos, "Towards efficient data valuation based on the shapley value," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1167–1176.

[3] A. Ghorbani and J. Zou, "Data shapley: Equitable valuation of data for machine learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2242–2251.

[4] C. Zhang, D. Ippolito, K. Lee, M. Jagielski, F. Tramèr, and N. Carlini, "Counterfactual memorization in neural language models," *arXiv preprint arXiv:2112.12938*, 2021.

[5] B. Efron, "Bootstrap methods: another look at the jackknife," in *Breakthroughs in statistics*. Springer, 1992, pp. 569–593.

[6] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1885–1894.

[7] A. Schioppa, P. Zablotskaia, D. Vilar, and A. Sokolov, "Scaling up influence functions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, 2022, pp. 8179–8186.

[8] Y. Wu, E. Dobriban, and S. Davidson, "Deltagrad: Rapid retraining of machine learning models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 355–10 366.

[9] G. Pruthi, F. Liu, S. Kale, and M. Sundararajan, "Estimating training data influence by tracing gradient descent," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 920–19 930, 2020.

[10] P. W. W. Koh, K.-S. Ang, H. Teo, and P. S. Liang, "On the accuracy of influence functions for measuring group effects," *Advances in neural information processing systems*, vol. 32, 2019.

[11] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[12] D. Yarotsky, "Error bounds for approximations with deep relu networks," *Neural Networks*, vol. 94, pp. 103–114, 2017.

[13] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, "Meta-transfer learning for few-shot learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 403–412.

[14] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li *et al.*, "Florence: A new foundation model for computer vision," *arXiv preprint arXiv:2111.11432*, 2021.

[15] G. Leclerc, A. Ilyas, L. Engstrom, S. M. Park, H. Salman, and A. Madry, "ffcv," https://github.com/libffcv/ffcv/, 2022.

[16] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on mathematical software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[17] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *the Journal of machine Learning research*, vol. 9, pp. 1871–1874, 2008.

[18] R. Fisher and T. Creator, "Iris," UCI Machine Learning Repository, 1988.

[19] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt, "Spambase," UCI Machine Learning Repository, 1999.

[20] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for exotic particles in high-energy physics with deep learning," *Nature communications*, vol. 5, no. 1, pp. 1–9, 2014.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[24] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, "Deep sets," *arXiv preprint arXiv:1703.06114*, 2017.

[25] L. Myers and M. J. Sirois, *Spearman Correlation Coefficients, Differences between*. American Cancer Society, 2006.

[26] A. Akhbardeh, H. Sagreiya, A. El Kaffas, J. K. Willmann, and D. L. Rubin, "A multi-model framework to estimate perfusion parameters using contrast-enhanced ultrasound imaging," *Medical physics*, vol. 46, no. 2, pp. 590–600, 2019.

[27] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[28] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gurel, B. Li, C. Zhang, C. J. Spanos, and D. Song, "Efficient task-specific data valuation for nearest neighbor algorithms," *arXiv preprint arXiv:1908.08619*, 2019.

[29] T. Wang, J. Rausch, C. Zhang, R. Jia, and D. Song, "A principled approach to data valuation for federated learning," in *Federated Learning*. Springer, 2020, pp. 153–167.

[30] R. Jia, F. Wu, X. Sun, J. Xu, D. Dao, B. Kailkhura, C. Zhang, B. Li, and D. Song, "Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8239–8247.

[31] S. Maleki, "Addressing the computational issues of the shapley value with applications in the smart grid," Ph.D. dissertation, University of Southampton, 2015.

[32] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala, "Overview and importance of data quality for machine learning tasks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3561–3562.

[33] V. Feldman and C. Zhang, "What neural networks memorize and why: Discovering the long tail via influence estimation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2881–2891, 2020.

[34] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1321–1330.

[35] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[36] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[37] A. Ginart, M. Y. Guan, G. Valiant, and J. Zou, "Making ai forget you: Data deletion in machine learning," *arXiv preprint arXiv:1907.05012*, 2019.

[38] Q. P. Nguyen, B. K. H. Low, and P. Jaillet, "Variational bayesian unlearning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[39] J. Brophy and D. Lowd, "Machine unlearning for random forests," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1092–1104.

[40] 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.

[41] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances in neural information processing systems*, pp. 409–415, 2001.

[42] S. Schelter, "amnesia–towards machine learning models that can forget user data very fast," in *1st International Workshop on Applied AI for Database Systems and Applications (AIDB19)*, 2019.

[43] Y. Wu, V. Tannen, and S. B. Davidson, "Priu: A provenance-based approach for incrementally updating regression models," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 447–462.

[44] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9304–9312.

[45] S. Basu, P. Pope, and S. Feizi, "Influence functions in deep learning are fragile," *arXiv preprint arXiv:2006.14651*, 2020.

[46] A. Mahadevan and M. Mathioudakis, "Certifiable machine unlearning for linear models," *arXiv preprint arXiv:2106.15093*, 2021.

[47] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in neural information processing systems*, 2016, pp. 3981–3989.

[48] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas, "Learning to learn without gradient descent by gradient descent," in *International Conference on Machine Learning*. PMLR, 2017, pp. 748–756.

[49] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," in *International Conference on Machine*

Learning. PMLR, 2017, pp. 2247–2255.

[50] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3751–3760.

[51] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4556–4565.

[52] Y. Cao, T. Chen, Z. Wang, and Y. Shen, "Learning to optimize in swarms," *Advances in neural information processing systems*, vol. 32, p. 15044, 2019.

[53] X. Chen, Y. Li, R. Umarov, X. Gao, and L. Song, "Rna secondary structure prediction by learning unrolled algorithms," *arXiv preprint arXiv:2002.05810*, 2020.

[54] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[55] D. Almeida, C. Winter, J. Tang, and W. Zaremba, "A generalizable approach to learning optimizers," *arXiv preprint arXiv:2106.00958*, 2021.

[56] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter, "Differentiable convex optimization layers," *arXiv preprint arXiv:1910.12430*, 2019.

[57] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[58] X. Chen, H. Dai, Y. Li, X. Gao, and L. Song, "Learning to stop while learning to predict," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1520–1530.

[59] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1897–1914.

[60] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

Appendix

A. Proof of Theorem 1

Theorem 1 (Restated)**.** *If for all $z \in [0,1]^d$, the loss function $\ell(\theta; z)$ is α-strongly convex in θ, and $\left\| \frac{\partial}{\partial\theta\partial z_i}\ell(\theta, z)\right\| \le B_1$ for $i \in [d]$, then for $\theta_k(D) = [\operatorname{argmin}_\theta L(\theta; D)]_k$, where $L(\theta; D) = \frac{1}{n}\sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$ and $[\cdot]_k$ means the kth element in the vector, then we have*

$$\left\|\frac{\partial\theta_k}{\partial D}\right\| \le B_1 \frac{\sqrt{d_{\text{param}}(d+1)}}{\sqrt{n}(\alpha+\lambda)}. \tag{14}$$

Proof. Given a loss function $L(\theta; D) = \frac{1}{n}\sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$, the unique local minimum $\widehat{\theta}(D)$ is characterized by the implicit function

$$\nabla L(\widehat{\theta}, D) = \frac{1}{n}\sum_{i=1}^n \nabla_\theta \ell(\widehat{\theta}, (x_i, y_i)) + \lambda\widehat{\theta} = 0. \tag{15}$$

Denote $\sigma(j) = \lceil j/(d+1)\rceil$ to be the index of the data point (and hence the loss function ℓ) jth dimension of D corresponds to. By the implicit function theorem and the chain rule, we have

$$\left(\sum_{i=1}^n \frac{\partial\ell_i}{\partial\theta^T\partial\theta} + n\lambda I\right)\frac{\partial\theta}{\partial D_j} + \frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j} = 0, \tag{16}$$

where $\ell_i = \ell(\cdot; (x_i, y_i))$ and, therefore, we obtain

$$\frac{\partial\theta}{\partial D_j} = -[H^{-1}]\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}, \tag{17}$$

where $H = \sum_{i=1}^n \frac{\partial\ell_i}{\partial\theta^T\partial\theta} + n\lambda I$ is the Hessian matrix. We use $[H^{-1}]_k$ to denote the kth row of the Hessian inverse, and, hence,

$$\frac{\partial\theta}{\partial D_j} = \left[\frac{\partial\theta_1}{\partial D_j}, \dots, \frac{\partial\theta_{d_{\text{param}}}}{\partial D_j}\right] \tag{18}$$

$$= \left[-[H^{-1}]_1\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}, \dots, -[H^{-1}]_{d_{\text{param}}}\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}\right]. \tag{19}$$

Therefore, we have

$$\left\|\frac{\partial\theta}{\partial D_j}\right\|_2^2 = \sum_{k=1}^{d_{\text{param}}}\left([H^{-1}]_k\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}\right)^2 \tag{20}$$

$$\le \sum_{k=1}^{d_{\text{param}}}\left\|[H^{-1}]_k\right\|^2\left\|\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}\right\|^2 \tag{21}$$

$$= \left\|\frac{\partial\ell_{\sigma(j)}}{\partial\theta\partial D_j}\right\|^2\left\|H^{-1}\right\|_F^2 \tag{22}$$

$$\le B_1^2\left\|H^{-1}\right\|_F^2, \tag{23}$$

where the inequality is due to the Cauchy–Schwarz inequality.

The remaining work is to bound $\left\|H^{-1}\right\|_F$. Denote $\lambda_1, \dots, \lambda_{d_{\text{param}}}$ as the eigenvalues of H. Observe that

$$\left\|H^{-1}\right\|_F = \sqrt{tr((H^{-1})^T H^{-1})} \tag{24}$$
$$= \sqrt{tr(H^{-1}H^{-1})} \tag{25}$$
$$= \sqrt{\sum_{i=1}^{d_{\text{param}}} \frac{1}{\lambda_i^2}}, \tag{26}$$

where the second equality holds due to H being symmetric, hence H^{-1} is also symmetric. Further, the last equality holds, because the eigenvalues for H^{-1} are $\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_{d_{\text{param}}}}$. Since each ℓ is α-strongly convex, we know that the minimum eigenvalue of the Hessian matrix H is at least $n\alpha + n\lambda$. Therefore, we obtain

$$\left\|H^{-1}\right\|_F \leq \frac{\sqrt{d_{\text{param}}}}{n\alpha + n\lambda}.$$

Then, it follows that

$$\left\|\frac{\partial \theta}{\partial D_j}\right\|_2 \leq B_1 \frac{\sqrt{d_{\text{param}}}}{n(\alpha+\lambda)}, \tag{27}$$

which implies

$$\left|\frac{\partial \theta_k}{\partial D_j}\right| \leq B_1 \frac{\sqrt{d_{\text{param}}}}{n(\alpha+\lambda)}, \tag{28}$$

and, finally, we have

$$\left|\frac{\partial \theta_k}{\partial D}\right| \leq B_1 \frac{\sqrt{d_{\text{param}}(d+1)}}{\sqrt{n}(\alpha+\lambda)}. \tag{29}$$

□

B. Proof of Theorem 2

Theorem 2. *If for all $z \in [0,1]^d$, the loss function $\ell(\theta; z)$ is β-smooth in θ, and $\left\|\frac{\partial}{\partial\theta\partial z_i}\ell(\theta, z)\right\| \leq B_1$ for $i \in [d]$, then for $\theta_k^{(t)}(D)$ defined by the kth entry of $\theta^{(t)}$, which is iteratively computed by $\theta^{(t)} = \theta^{(t-1)} - \eta\nabla L(\theta^{(t-1)}; D)$ and $\theta^{(0)} = 0$ with a learning rate $\eta > 0$, and $L(\theta; D) = \frac{1}{n}\sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2}\|\theta\|_2^2$, we have*

$$\left\|\frac{\partial \theta_k^{(t)}}{\partial D}\right\| \leq \left(1 - (1 - \eta\lambda - \eta d_{\text{param}}\beta)^t\right) \frac{B_1\sqrt{(d+1)}}{\sqrt{n}(\lambda + d_{\text{param}}\beta)}.$$

Proof. Consider an iteration $\theta_k^{(t)} = \theta_k^{(t-1)} - \eta\nabla L(\theta^{(t-1)}; D)$. Taking a derivative with respect to D_j for both sides, we have

$$\frac{\partial \theta_k^{(t)}}{\partial D_j} = \frac{\partial \theta_k^{(t-1)}}{\partial D_j} \tag{30}$$
$$- \eta \left[\lambda \frac{\partial \theta_k^{(t-1)}}{\partial D_j} + \frac{1}{n}\sum_{i=1}^n \frac{\partial^2 \ell_i}{\partial \theta_k^2}\frac{\partial \theta_k^{(t-1)}}{\partial D_j} + \frac{1}{n}\frac{\partial \ell_{\sigma(j)}}{\partial\theta_k \partial D_j}\right] \tag{31}$$
$$= \left(1 - \eta\lambda - \frac{\eta}{n}\sum_{i=1}^n \frac{\partial^2 \ell_i}{\partial \theta_k^2}\right)\frac{\partial \theta_k^{(t-1)}}{\partial D_j} - \frac{\eta}{n}\frac{\partial \ell_{\sigma(j)}}{\partial\theta_k \partial D_j} \tag{32}$$
$$\leq (1 - \eta\lambda - \eta d_{\text{param}}\beta)\frac{\partial \theta_k^{(t-1)}}{\partial D_j} - \frac{\eta}{n}B_1, \tag{33}$$

where the inequality follows due to $\left|\frac{\partial \ell_{\sigma(j)}}{\partial\theta_k \partial D_j}\right| \leq B_1$ and $\frac{\partial^2 \ell_i}{\partial \theta_k^2} \leq d_{\text{param}}\beta$.

Observe that

$$\frac{\partial \theta_k^{(t)}}{\partial D_j} + \frac{\eta B_1}{n\alpha} \leq (1-\alpha)\left(\frac{\partial \theta_k^{(t-1)}}{\partial D_j} + \frac{\eta B_1}{n\alpha}\right) \tag{34}$$
$$\leq (1-\alpha)^t \frac{\eta B_1}{n\alpha}. \tag{35}$$

Therefore, we obtain

$$\left|\frac{\partial \theta_k^{(t)}}{\partial D_j}\right| \leq \left(1 - (1 - \eta\lambda - \eta d_{\text{param}}\beta)^t\right) \frac{B_1}{n(\lambda + d_{\text{param}}\beta)}, \tag{36}$$

which results in

$$\left\|\frac{\partial \theta_k^{(t)}}{\partial D}\right\| \leq \left(1 - (1 - \eta\lambda - \eta d_{\text{param}}\beta)^t\right) \frac{B_1\sqrt{(d+1)}}{\sqrt{n}(\lambda + d_{\text{param}}\beta)}. \tag{37}$$

□

[12] shows that continuous differentiable functions with a smaller upper bound of the gradient norm can be more efficiently approximated by deep ReLU networks. For completeness, we present the result here.

Theorem 3. *For any Lipschitz continuous function $f : [0,1]^d \to \mathbb{R}$ with range B and $\left\|\frac{\partial f}{\partial x}\right\| \leq L$ for all $x \in [0,1]^d$, there is a ReLU network that is capable of expressing any such function within an arbitrary error ε and has no more than $c\ln(2^{d+1}Bd(d+1)/\varepsilon) + 1$ layers and $d\left(2^{d+1}dL/\varepsilon + 1\right)^d \left(c\ln(2^{d+1}Bd(d+1)/\varepsilon) + 1\right)$ computational units.*

As we can see, if L is smaller, the upper bound of the network size is smaller. The proof of this theorem is done by simply plugging in constants for Theorem 3.1 in [12].

C. Additional Results

1) SVM As the Base Model

We summarize the results for dataset deletion and dataset addition experiments with Support Vector Machine (SVM) as base models in VIII.

The results in Table VII and VIII illustrate that the proposed **ModelPred** achieves similar performance with different base models (*i.e.*, LR and SVM) in terms of the accuracy of parameter prediction and strong correlation between the predicted utility and the actual utility of subsets.

2) Transfer Learning in NN

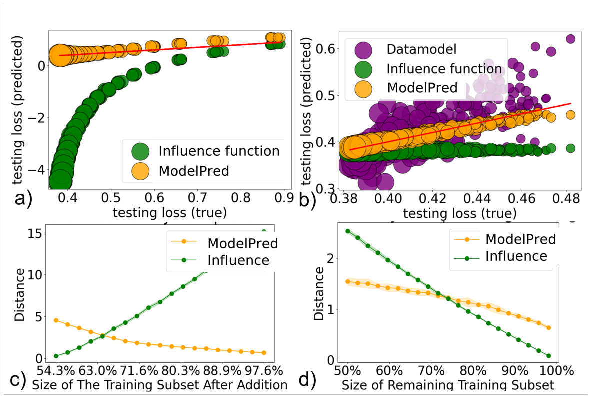

Fig. 9: Results on Hymenoptera images with NN as the base model. a) and b): the estimated loss by predicted parameters of each subset in the scenario of dataset deletion and addition; c) and d): the Euclidean distance between predicted parameters and optimal parameters in the same scenario.

Similar to Figure 2, Figure 9 visualizes the estimated loss and the Euclidean distance between predicted parameters and optimal parameters by Influence Function and **ModelPred** with NN as the base model. It shows the estimation error of the proposed method remains at a lower level after the intersection point. This demonstrates the advantage of the proposed method over linear approximation by Influence Function, and it also quantifies the size of the subset after addition or deletion where the proposed method dominates the performance. In summary, **ModelPred** demonstrates its superior performance in predicting the parameters for the training of ML models with subsets of different sizes.

3) Larger Training Set

We increase the number of training data points to conduct larger-scale experiments to validate our proposed method. The setting are summarized in Table IX and the results are summarized in Table X.

MNIST-2000 **MNIST-2000** We increase the number of training points to 2000 and 10000 for MNIST dataset and conduct the dataset addition and deletion experiments. We construct 15000 training samples to construct the training set Φ for MNIST-2000 and 100,000 training samples for MNIST-10000. Collecting 100,000 training samples takes roughly 3hrs. We find **ModelPred** achieves similar effective results with very high Spearman correlation compared to the results of a training set with 300 data points.

ADULT([60]) ADULT dataset (i.e., 'Census Income' dataset) is a collection of roughly 48000 records of personal income with 14 attributes (d=14). The predefined prediction task is to determine whether a person makes over 50K a year as a binary classification problem. We first sample 10000 data points with 9500 male records and 500 female records from the dataset as our full training dataset. Another 1000 points are selected for testing. Instead of studying the dependency of the trained model on a randomly generated subset, we target utilizing **ModelPred** to investigate the effect of data collected from a subgroup (i.e. female). Therefore, we construct the 15000 training sample set Φ by removing a subset of these 500 points from the training set. In the dataset deletion experiment, we also remove subsets randomly generated by these 500 points from the training set. The results in Table X show that **ModelPred** can accurately predict the trained model as well as the utility. Therefore, it can be applied to study the effect of a subgroup within a large dataset on the training performance of a base model.

4) Memorization and Influence Score Estimation

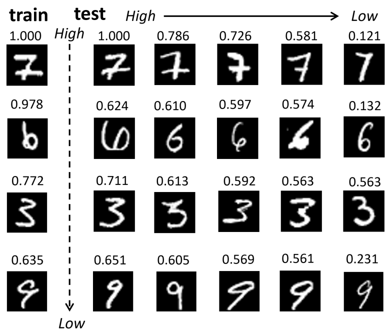

Fig. 10: Examples of selected influence pairs in MNIST. The left column shows examples of memorization score from high to low in the training set. For each training example, examples with high and low estimates are presented in the testing set.

5) Abaltion Study

In the phase of offline training, we approximate the learning algorithm by fine-tuning the last layer of a pre-trained model. We further perform an ablation study to validate the effectiveness of employing transfer learning in the large base model retraining. A Resnet-18 model is pre-trained on CIFAR-10 dataset and the weight of the last layer is fine-tuned to a small subset (*i.e.*, with 200 samples) randomly selected from CIFAR-10. To compare with fix embedding of feature extractors, we randomly initialize the weight of a Resnet-18 model and only adjust the weight of the last layer. These two approaches are used to retrain the subsets of 200 samples in the permutation sampling to estimate the SV of these samples. We select the samples with the lowest and highest SV estimated by these two approaches in Figure 11 and 12.

As shown in Figure 11, compared to samples in the same class (*i.e.*, deer, dog, and horse), those with lower SV (*i.e.*, in the blue box) are vaguer and less representative. By comparing Figure 11 and 12, we find that the same image (the third row and third column of Figure 11) is estimated to obtain a high SV by transfer learning, but a low SV by random initialization. However, this image is representative of the class of horse. Moreover, there is no significant difference in representativeness between the images with high

TABLE VII: A summary of **ModelPred** results and baseline comparison results in the scenario of dataset deletion with SVM as the base model. The best results are highlighted in **bold**.

Dataset	Algorithm	Parameter		Utility			
		$\left\|\widehat{\theta}-\theta^*\right\|$	Std	NRMSE	Std	Spearman Corr	Std
	ModelPred	**1.16E-01**	**1.53E-02**	**17.25%**	**7.70%**	**0.9214**	**0.0763**
Iris	Influence function	4.63E-01	8.31E-03	54.10%	8.99%	0.0143	0.4427
	ModelPred	**8.74E-01**	**1.58E-02**	**6.85%**	**0.55%**	**0.9826**	**0.0059**
SPAM	Influence Function	1.47E+00	7.78E-03	48.96%	3.23%	0.1395	0.2271
	ModelPred	**5.38E-01**	**2.72E-02**	**7.52%**	**1.16%**	**0.9048**	**0.0442**
HIGGS	Influence Function	6.12E-01	6.61E-03	16.97%	2.78%	0.7274	0.1377

TABLE VIII: A summary of **ModelPred** results and baseline comparison results in the scenario of dataset addition with SVM as the base model. The best results are highlighted in **bold**.

Dataset	Algorithm	Parameter		Utility			
		$\left\|\widehat{\theta}-\theta^*\right\|$	Std	NRMSE	Std	Spearman Corr	Std
	ModelPred	1.68E-01	1.15E-02	**40.82%**	**3.01%**	**0.9292**	**0.0379**
Iris	Influence Function	**1.50E-01**	**6.72E-04**	63.29%	3.88%	0.1694	0.2300
	ModelPred	**1.19E+00**	**1.77E-02**	**10.44%**	**1.04%**	**0.9848**	**0.0046**
SPAM	Influence Function	1.62E+00	3.27E-03	55.03%	2.01%	0.8080	0.0638
	ModelPred	**7.23E-01**	**2.07E-02**	20.66%	2.55%	**0.8467**	**0.0378**
HIGGS	Influence Function	7.30E-01	7.37E-03	**17.34%**	**2.41%**	0.7095	0.0828

TABLE IX: Setting for Experiments with Larger Training Set.

Experiment	Total Training Data Points	Total Testing Data Points	Size of Staring Subset	Size of Intermediate Subsets	Size of Ending Subset
Addition: MNIST-2000	2000	500	1000	[1000, 1050, 1100, …]	2000
Deletion: MNIST-2000	2000	500	2000	[1950, 1900, 1850, …]	1000
Deletion: MNIST-10000	10000	1000	5000	[5100, 5200, 5300, …]	10000
Deletion: ADULT	10000	1000	9250	[9250, 9255, 9230, ...]	10000

TABLE X: A summary of **ModelPred** results and baseline comparison results of experiments with larger training set. The best results are highlighted in **bold**.

Dataset	Algorithm	Parameter		Utility			
		$\left\|\widehat{\theta}-\theta^*\right\|$	Std	NRMSE	Std	Spearman Corr	Std
	ModelPred	1.44E+00	5.00E-03	**3.01%**	**0.12%**	**0.9993**	**0.0000**
Addition: MNIST-2000	Influence function	**1.41E-01**	5.00E-03	60.10%	0.31%	0.9439	0.0060
	ParaLearn	1.10E-01	**2.00E-03**	209.10%	1.13%	N/A	N/A
	ModelPred	2.53E+00	**2.38E-03**	**3.18%**	**0.22%**	**0.9912**	**0.0012**
Deletion: MNIST-2000	Influence function	3.41E+00	1.83E-01	57.56%	0.32%	0.9439	0.0060
	ParaLearn	**1.87E+00**	4.57E-03	125.32%	0.79%	N/A	N/A
	Datamodel	N/A	N/A	311.20%	7.37%	-0.5854	0.0271
	ModelPred	**2.30E+00**	**7.85E-03**	**1.19%**	**0.08%**	**0.9998**	**0.0002**
Deletion: MNIST-10000	Influence function	3.08E+00	1.18E-02	56.57%	0.55%	-0.0302	0.1309
	ParaLearn	2.92E+01	6.36E-03	119.73%	1.30%	N/A	N/A
	Datamodel	N/A	N/A	75.86%	7.11%	-0.0473	0.1530
	ModelPred	**3.27E-02**	**1.11E-03**	9.25%	1.78%	**0.9287**	0.0421
Deletion: ADULT	Influence function	9.70E-01	2.45E-02	**8.86%**	**1.68%**	0.9146	**0.0420**
	ParaLearn	1.06E-01	1.24E-03	39.80%	7.71%	N/A	N/A
	Datamodel	N/A	N/A	209.10%	1.13%	0.2454	0.1030

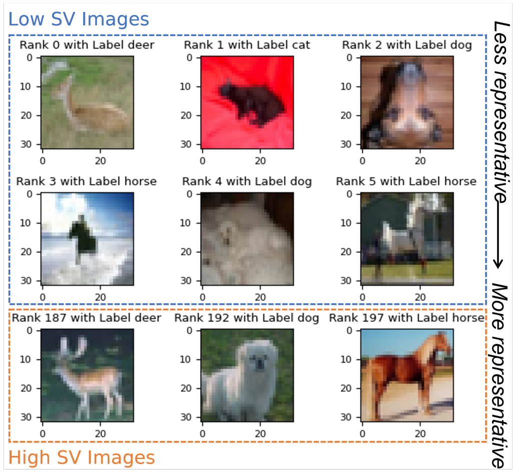

Fig. 11: Images with SV estimated by NN trained with transfer learning on a small subset of CIFAR-10.

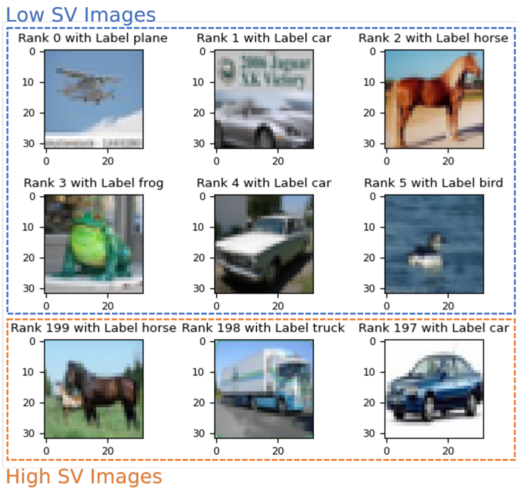

Fig. 12: Images with SV estimated by NN trained without transfer learning on a small subset of CIFAR-10.

and low SV estimated by NN with random initialization as shown in 12, which might be caused by the low learning performance of NN during the subset retraining. This indicates that retraining the large NN with transfer learning can effectively maintain the utility of training samples, which allows us to significantly reduce the model parameters to learn. Therefore, **ModelPred** can be further utilized to learn the parameters of the large DNN models by transfer learning.