Dynamic Shapley Value Computation

Jiayao Zhang*, Haocheng Xia*, Qiheng Sun*, Jinfei Liu†, Li Xiong‡, Jian Pei§, Kui Ren*
*Zhejiang University, {jiayaozhang,xiahc,qiheng_sun,kuiren}@zju.edu.cn
†Zhejiang University, ZJU-Hangzhou Global Scientific and Technological Innovation Center, {jinfeiliu}@zju.edu.cn
‡Emory University, lxiong@emory.edu
§Simon Fraser University, jpei@cs.sfu.ca

***Abstract*—With the prevalence of data-driven research, data valuation has attracted attention from the computer science field. How to appraise a single datum becomes an imperative problem, especially in the context of machine learning. *Shapley value* is widely used to fairly measure the contribution of data points in machine learning since it is the unique definition that satisfies all four desired properties: *balance*, *symmetry*, *additivity*, and *zero element*. However, computing Shapley value is known to be a #P-hard problem. As data is subject to changes, dynamic data exists pervasively in real-world scenarios. Pricing such dynamic data is more challenging due to the prohibitively expensive cost of recalculation from scratch. In this paper, we study the problem of *Dynamic Shapley Value Computation*, which updates Shapley value when dynamically adding/deleting data points. For adding data points, to prune unnecessary computation of overlapping model utilities, we propose the pivot-based algorithm that can reduce half computation time in general. We also propose the delta-based algorithm to capture Shapley value changes, which requires a smaller sample size to converge. For deleting data points, we present the YN-NN algorithm that derives the new Shapley value from the data structure of precomputed model utilities in an efficient way. Based on Shapley value changes, we give another version of the delta-based algorithm for deleting data points. Besides, we propose heuristic algorithms to draw on experimental observations for both adding and deleting data points. Extensive experimental results demonstrate the efficiency and effectiveness of our proposed algorithms.**

I. Introduction

Since data creates a steady stream of wealth, the economic value of data attracts great attention from both industry and academia. Data-driven applications, and more specifically machine learning, promote data valuation to become an increasingly important discipline in data science. How to quantify the value of a single datum equitably is a significant topic in the emerging data market field [5, 16, 22, 23, 28, 29, 30, 33].

As depicted in Figure 1, a model-based data market connects data owners, the broker, and model buyers [33, 37]. Data owners sell data to the broker in exchange for compensation; the broker collects data from multiple data owners, builds and sells various machine learning models to model buyers; model buyers pay for cost-effective models that satisfy their demands. We focus on the interaction between data owners and the broker in this paper. Data owners supply data to the broker for compensation which should be allocated from the model revenue and distributed fairly based on their contribution. To enforce this desideratum, data valuation assigns a value to each data owner based on her contribution to this model task. That is, the total compensation of each data owner is the sum of compensation on all model tasks. One prevalent approach for data valuation is Shapley value.

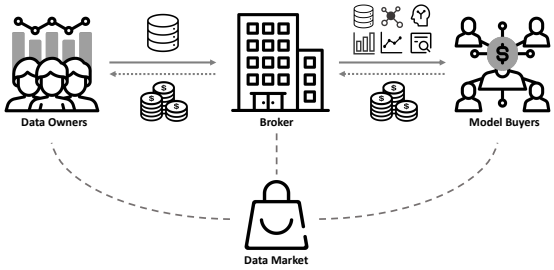


Fig. 1: Overview of the data market.

Shapley value is a concept used to measure the contribution of each participant in cooperative game theory, which was named in honor of Lloyd Shapley [38]. In decades, Shapley value has been widely adopted in many domains [7, 17, 32, 36] on the strength of its profound theoretical foundation. Shapley value is demonstrated to be the only definition that satisfies all four desired properties for payoff division: *balance*, *symmetry*, *additivity*, and *zero element* [38]. *Balance* indicates that the total payoff is fully distributed to all players; *Symmetry* specifies that two players have the same valuation if they have the same marginal contributions; *Additivity* indicates that value on individual tasks sums up to the value on a combined task; *Zero element* specifies that the value of players is null if their marginal contributions are null, i.e., no contribution, no payment.

In the context of machine learning, Shapley value has been extensively applied to appraise the contribution of data points. Shapley value of each data point is the average marginal contribution of the data point over all possible permutations of data points, where the marginal contribution refers to the difference of utilities or accuracy of models trained on (sub)coalitions of the dataset with and without the data point. This value represents the contribution or the relevance of the data point toward the model task. Empirical experiments show that the prediction accuracy of models trained on data points selected by Shapley value is substantially better than that of the other popular data valuation method called leave-one-out scores - the difference between model accuracy when trained on the entire dataset with and without the data point [9]. How-

The first three authors contributed equally. Jinfei Liu is the corresponding author.

ever, computing the exact Shapley value is known to be a #P-hard problem [12]. Complete enumeration consists of generating $O(2^n)$ (sub)coalitions of n data points and computing all marginal contributions of each point. This prohibitively expensive computational cost discourages Shapley value from being implemented in practical applications. Several approximation methods have been proposed to overcome such intractability. Monte Carlo algorithms are the most general approaches approximating Shapley value through permutation sampling or coalition sampling [3, 19, 35].

Motivation. The prior works assume that all data points are stable and immutable. They mainly attempt to approximate Shapley value efficiently on a fixed dataset. In practice, a dataset can be continuously changed with new or removed data points. Shapley value derived from an original dataset is no longer valid when the dataset is updated. Blindly reevaluating Shapley value for a large dataset from scratch is inefficient due to the exponential computation cost, which becomes more intolerable under the complexity of machine learning models particularly.

Different from the traditional Shapley value computation, our work focuses on Shapley value calculated on the dynamic dataset, referred to as *Dynamic Shapley Value Computation*, which aims at improving the efficiency and effectiveness of computing Shapley value with respect to dynamically adding or deleting data points.

TABLE I: The patient information (before addition).

Id	Age	Sex	Cp	Rbps	Chol	Fbs	Disease
$\boldsymbol{z}_1$	61	male	level 4	138	166	<120	severe
$\boldsymbol{z}_2$	46	female	level 2	105	204	<120	none

TABLE II: The patient information (after addition).

Id	Age	Sex	Cp	Rbps	Chol	Fbs	Disease
$\boldsymbol{z}_1$	61	male	level 4	138	166	<120	severe
$\boldsymbol{z}_2$	46	female	level 2	105	204	<120	none
$\boldsymbol{z}_3$	59	male	level 3	150	212	>120	none

TABLE III: Marginal contributions.

(a) Before addition.

Permutation	$\boldsymbol{z}_1$	$\boldsymbol{z}_2$
$[\boldsymbol{z}_1, \boldsymbol{z}_2]$	5	7
$[\boldsymbol{z}_2, \boldsymbol{z}_1]$	6	6

(b) After addition.

Permutation	$\boldsymbol{z}_1$	$\boldsymbol{z}_2$	$\boldsymbol{z}_3$
$[\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3]$	5	7	8
$[\boldsymbol{z}_1, \boldsymbol{z}_3, \boldsymbol{z}_2]$	5	10	5
$[\boldsymbol{z}_2, \boldsymbol{z}_1, \boldsymbol{z}_3]$	6	6	8
$[\boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_1]$	7	6	7
$[\boldsymbol{z}_3, \boldsymbol{z}_1, \boldsymbol{z}_2]$	7	10	3
$[\boldsymbol{z}_3, \boldsymbol{z}_2, \boldsymbol{z}_1]$	7	10	3

Motivating Example. There are many example applications that *dynamic Shapley value computation* may be desired. For instance, a medical institution known for treating heart disease patients may wish to construct a predictive classifier for heart disease in order to aid diagnosis. The patients contribute their data for the model construction and get compensated according to the corresponding contributions. It is common that new patients may join and original participants drop out.

Tables I and II demonstrate a dynamic patient dataset from the Cleveland Heart Disease Data Set [14] before and after addition of patient $\boldsymbol{z}_3$, respectively. Each data point represents the disease related information of a patient: the age, the resting blood pressure, the degree of heart disease, etc. Given the data update, a naive way is to recalculate Shapley value for all patients from scratch. With the marginal contributions in Table III where each value corresponds to column $\boldsymbol{z}_i$ indicates the marginal contribution of $\boldsymbol{z}_i$ with respect to the permutation on each row, the Shapley value of the two patients $\{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ is $\{\mathcal{SV}_1, \mathcal{SV}_2\} = \{\frac{11}{2}, \frac{13}{2}\}$. The new Shapley value of the patients' data in Table II is $\{\mathcal{SV}_1, \mathcal{SV}_2, \mathcal{SV}_3\} = \{\frac{37}{6}, \frac{49}{6}, \frac{17}{3}\}$. The detailed formula of Shapley value is shown in Section III, which requires all possible marginal contributions generated by data points. Recomputing the new Shapley value of patients in Table II naively incurs repetitive evaluation of model utilities $\mathcal{U}(\{\boldsymbol{z}_1\})$,$\mathcal{U}(\{\boldsymbol{z}_2\})$, and $\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\})$, which are already computed when evaluating Shapley value for patients in Table I. Hence, we aim to propose more efficient and effective approaches for incrementally computing Shapley values on dynamic datasets.

Contribution. In this paper, for the first time, we give the definition of *dynamic Shapley value computation* and propose efficient approaches to solve it. For adding a data point, we propose the pivot-based algorithm that utilizes precomputation to replace half of marginal contributions in permutation-based sampling. Different from traditional Monte Carlo algorithms, it costs almost half with the same sampled permutations. Also, we propose the delta-based algorithm based on the Shapley value changes that can achieve an (ϵ, δ)-approximation in $O(\frac{Tnd^2 \ln \frac{2}{\delta}}{\epsilon^2})$ time. We demonstrate that the pivot-based algorithm and the delta-based algorithm are suitable for adding multiple data points as well. For deleting a data point, we construct dynamic data structure of two three-dimension arrays to store utility functions efficiently. It is easy to extend the data structure to cope with deleting multiple data points by using two additional multiple-dimensional arrays. Based on the concept, we propose an algorithm with $O(n^k)$ space complexity for deleting k data points. Inspired by empirical observations, we propose heuristic algorithms when adding or deleting data points, which allows us to efficiently update Shapley value but comes with a small cost of accuracy.

Apart from data valuation, our proposed algorithms are practical for Shapley value computation among dynamic players in the general class of games with characteristic utility function forms. We briefly summarize our contributions.

- We identify the problem of dynamic Shapley value computation and propose several algorithms that are capable of deriving Shapley value on dynamic datasets.
- For optimization of dynamic Shapley value computation, we offer some practical methods including intermediate result reorganization, differential marginal contribution, and heuristics.
- Extensive experiments on Iris and Adult datasets are conducted, which demonstrate the effectiveness and efficiency of our proposed algorithms for updating Shapley value when data points are dynamically changed.

II. Related Work

Data Market. The growing interest in data trading has led to the emergence of data markets. Data markets with data-based pricing sell raw data directly or supply the personalized datasets for specific tasks, while data markets with query-based pricing sell queries [4, 24, 25]. Data markets with model-based pricing are proposed recently [5, 33], which allow machine learning model trading among stakeholders instead of raw data or queries. Kurtulmus et al. [26] developed a model exchange market via blockchain technology. Chen et al. [5] proposed a pricing framework for machine learning over relational data. It sells model instances with different accuracy options via a random noise injection approach, and the price of the purchased model depends on its accuracy. Liu et al. [33] proposed the first end-to-end model marketplace with differential privacy, which responds to the needs of data owners, the broker, and model buyers. Those efforts are made to establish complete data market platforms to bridge gaps in data exchange. In this paper, we focus on the interaction between data owners and the broker in model-based markets. Dynamic Shapley value computation is proposed to efficiently and fairly distribute compensation among data owners in response to the data owner dynamically joins or exits.

Shapley Value Computation. Shapley value [38] has an incredible impact on the cooperative game theory, which has been applied in tackling many problems, such as terrorist network [32], profit allocation [39], query answering [13], data/feature selection [15, 19], and data pricing [1, 5, 6, 21, 33, 31]. Computing the exact Shapley value can be a #P-hard problem [12]. To overcome the drawback, several techniques are developed to approximate Shapley value. Castro et al. [3] estimated Shapley value based on permutation sampling for the general class of games. Maleki et al. [35] provided a stratified sampling algorithm with non-asymptotic error bounds. Zhang et al. [42] proposed a novel stratification design based on complementary contributions.

In machine learning, Shapley value is used to quantify the contributions of data points toward training a model. The interpretation of the utility function is usually the model performance trained by subsets of the training dataset predicted on the test dataset. Ghorbani et al. [19] proposed Truncated Monte Carlo Shapley and Gradient Shapley, which leads to substantial computational savings of near-zero marginal contributions. Jia et al. [22] focused on k-NN classifier which is considered lazy and developed an algorithm based on Locality Sensitive Hashing with sublinear complexity. Ghorbani et al. [18] proposed distributional Shapley to measure the value of data points where the dataset is drawn i.i.d from the underlying distribution. On the basis of this work, Kwon et al. [27] derived the analytic expressions for distributional Shapley for the canonical problems of linear regression, binary classification, and non-parametric density estimation. Distributional Shapley can derive the Shapley value distribution of a subset from the Shapley value distribution of the whole dataset, but cannot estimate Shapley value for a specific data point.

Dynamic Problem. Dynamic problem is a classic area not only in computational geometry but also in networking, data mining, etc. Those computational problems arise in contexts where the input is changing and we call this setting *dynamic* as opposed to *static*. Algorithms in dynamic setting have been studied for decades. For instance, a self-balancing binary search tree attempts to keep its height under random insertions or deletions [20]. Tong et al. [40] formally defined the global dynamic pricing problem in spatial crowdsourcing, presented a base pricing strategy and developed MAPS to optimize supply and pricing. Similarly, Liu et al. [34] proposed a skyline diagram which can be used to facilitate dynamic skyline queries.

The most related literature to our work is [10, 11, 41] for dynamic query-based pricing in data markets, which refers to price temporal views on data stream properly. Upadhyaya et al. [41] designed a notion of refunds on data APIs to achieve optimal history-aware pricing so that buyers do not have to be charged twice for the same data. As Deep and Koutris [10] pointed out, the refund mechanism gives no guarantee to arbitrage-freeness. A pricing platform, called Qirana [10, 11], provided a query-based real-time pricing mechanism with an arbitrage-free guarantee. These works offer dynamic prices for buyers, while our work focuses on the dynamic contribution evaluation of data owners based on Shapley value. Furthermore, those existing techniques on dynamic problems cannot be adopted to our problem directly.

III. Preliminaries and Problem Statement

In this section, we first revisit the definition of Shapley value in data markets as well as the classical approximation method. Then, we present the problem statement.

Algorithm 1: Monte Carlo Shapley value computation.

input : data points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ and $\tau > 0$
output: Shapley value $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le n)$

$\mathcal{SV}_i \leftarrow 0$ $(1 \le i \le n)$;
for *k=1 to* τ **do**
  let π^k be a random permutation of $\{1, \ldots, n\}$;
  for *i=1 to n* **do**
    $\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}\})$;
    $\mathcal{SV}_{\pi^k(i)} += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
for *i=1 to n* **do**
  $\mathcal{SV}_i /= \tau$;
return $\mathcal{SV}_1, \ldots, \mathcal{SV}_n$;

Consider n data owners $D_i, \ldots, D_n$ such that data owner D_i owns a data point $\boldsymbol{z}_i$ $(1 \le i \le n)$. A *coalition* S is a subset of $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$. Denote by $\mathcal{U}(\mathcal{S})$ a *utility function* that represents the performance of a model trained on coalition S towards a task, e.g., model accuracy. Shapley [38] gave a function that evaluates the contribution from each owner D_i to the whole coalition $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$, which is the unique function satisfying four desirable properties, namely *balance*, *symmetry*, *additivity*, and *having a zero element*.

$$\mathcal{SV}_i = \frac{1}{n} \sum_{\mathcal{S} \subseteq \{\boldsymbol{z}_1 \ldots \boldsymbol{z}_n\} \setminus \boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})}{\binom{n-1}{|\mathcal{S}|}} \tag{1}$$

Computing the exact Shapley value is exponential in time. In practice, the Monte Carlo simulation method [3] is a well-known technique to approximate Shapley value, as shown in Algorithm 1. Concretely, let τ be the number of sampled permutations and $\boldsymbol{z}_{\pi^k(i)}$ be the data point with position index i in permutation π^k. In line 3, we randomly sample permutations of all data points. In lines 4-5, we traverse each permutation from the head to the tail and then calculate the marginal contribution of each data point. Finally, we compute the average of the marginal contributions as the approximate Shapley value in line 6. This Monte Carlo simulation gives an unbiased estimation of the exact Shapley value. The number of permutations controls the trade-off between approximation error and time cost. A larger number of samples brings more accurate Shapley value at the expense of more running time.

In this paper, we consider computing Shapley value in a dynamic dataset. A dataset $\mathcal{D}$ is said to be *dynamic* if the data points in the dataset may be added or deleted over time.

Addition A set of data points $\mathcal{D}_{add}$ are added to $\mathcal{D}$ to form a new dataset $\mathcal{N}^+ = \mathcal{D} \cup \mathcal{D}_{add}$.

Deletion A subset of data points $\mathcal{D}_{del} \subseteq \mathcal{D}$ are removed from $\mathcal{D}$ to form a new dataset $\mathcal{N}^- = \mathcal{D} - \mathcal{D}_{del}$.

Given a dynamic dataset with n data points $\mathcal{D} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$, denote by $\mathcal{SV}_i$ the Shapley value of $\boldsymbol{z}_i$ in $\mathcal{D}$. For a set of added data points $\mathcal{D}_{add} = \{\boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_m\}$ $(m \geq n)$ and the updated data set $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\} \cup \{\boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_m\}$, denote by $\mathcal{SV}_i^+$ the Shapley value of $\boldsymbol{z}_i$ in $\mathcal{N}^+$ $(1 \leq i \leq m)$. That is,

$$\mathcal{SV}_i^+ = \frac{1}{m} \sum_{\mathcal{S} \subseteq \mathcal{N}^+ \setminus \boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})}{\binom{m-1}{|\mathcal{S}|}}. \tag{2}$$

Similarly, for a set of data points $\mathcal{D}_{del} = \{\boldsymbol{z}_p, \ldots, \boldsymbol{z}_q\} \subseteq \mathcal{D}$ and the updated data set $\mathcal{N}^- = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\} - \{\boldsymbol{z}_p, \ldots, \boldsymbol{z}_q\}$, denote by $\mathcal{SV}_i^-$ the Shapley value of $\boldsymbol{z}_i$ in $\mathcal{N}^-$ $(1 \leq i \leq n)$. That is,

$$\mathcal{SV}_i^- = \frac{1}{n+p-q-1} \sum_{\mathcal{S} \subseteq \mathcal{N}^- \setminus \boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})}{\binom{n+p-q-2}{|\mathcal{S}|}}. \tag{3}$$

Trivially, for any i $(1 \leq i \leq n)$ such that $\boldsymbol{z}_i \in \mathcal{D}_{del}$, since $\boldsymbol{z}_i$ is removed from $\mathcal{D}$ and thus does not appear in $\mathcal{N}^-$ at all, $\mathcal{SV}_i^- = 0$. In this paper, we only focus on the Shapley values of those data points $\boldsymbol{z}_i$ that belong to $\mathcal{N}^-$ in the deletion case.

In dynamic schema, the dataset can be sequentially added or delete a data point. Thus, the *problem of dynamic Shapley value computation* is to compute $\mathcal{SV}_i^+/\mathcal{SV}_i^-$ for all the data points in $\mathcal{N}^+$ and $\mathcal{N}^-$ efficiently in real time.

It is far from trivial to compute dynamic Shapley values. A straightforward approach is to compute the new Shapley value using the Monte Carlo simulation method on the new dataset once changes occur. Because the measurement of utility functions involves model training, the time cost is dramatic. We notice that not all utilities of subsets are affected by changes, an efficient dynamic Shapley value computation method should try to reduce or avoid recomputing utility functions.

IV. Adding Data Points

We start from the basic scenario where only one data point is added. In Section IV-A we develop the pivot-based algorithm, which can reuse half of the utility results. While the pivot-based method focuses on reducing redundant computation, a method which aims to reduce the number of permutations is needed. Thus, in Section IV-B, we develop the delta-based algorithm, which evaluates differential marginal contributions. In Section IV-C, we extend the delta-based algorithm to handle the general situation where multiple data points are added sequentially.

A. The Pivot-based Algorithm

As discussed in Algorithm 1, the Shapley value can be interpreted as the average marginal contribution over all possible permutations. Consider dataset $\mathcal{D} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and updated dataset $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$. For the permutations in the updated dataset $\mathcal{N}^+$, each original data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$ appears either before or after the new data point $\boldsymbol{z}_{n+1}$ with equal frequency. It is easy to see that, for the half of the permutations where $\boldsymbol{z}_i$ appears before the new point $\boldsymbol{z}_{n+1}$, the marginal contributions of $\boldsymbol{z}_i$ remain the same as in $\mathcal{D}$. For example, in Table II, data point $\boldsymbol{z}_1$ appears before new point $\boldsymbol{z}_3$ in three permutations $[\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3]$, $[\boldsymbol{z}_2, \boldsymbol{z}_1, \boldsymbol{z}_3]$, and $[\boldsymbol{z}_1, \boldsymbol{z}_3, \boldsymbol{z}_2]$. The marginal contributions of $\boldsymbol{z}_1$, i.e., $\mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\emptyset)$, $\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_2\})$, and $\mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\emptyset)$, respectively, are the same as in the original dataset and can be reused.

Motivated by this observation, we propose the *pivot-based algorithm* that reuses the unchanged marginal contributions computed in the original dataset for the new dataset. For each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$ in the original dataset, taking the new data point $\boldsymbol{z}_{n+1}$ as the pivot, we can divide all permutations in the new dataset into two groups: $G_{\mathcal{L}}^i$ consists of the permutations where $\boldsymbol{z}_i$ is located in front of the pivot and $G_{\mathcal{R}}^i$ consists of the permutations where $\boldsymbol{z}_i$ is located behind the pivot. For example, consider $\boldsymbol{z}_3$ in Table II as a pivot. For $\boldsymbol{z}_1$, $G_{\mathcal{L}}^1$ contains $[\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3]$, $[\boldsymbol{z}_2, \boldsymbol{z}_1, \boldsymbol{z}_3]$, and $[\boldsymbol{z}_1, \boldsymbol{z}_3, \boldsymbol{z}_2]$, and $G_{\mathcal{R}}^1$ contains $[\boldsymbol{z}_3, \boldsymbol{z}_2, \boldsymbol{z}_1]$, $[\boldsymbol{z}_2, \boldsymbol{z}_3, \boldsymbol{z}_1]$, and $[\boldsymbol{z}_3, \boldsymbol{z}_1, \boldsymbol{z}_2]$.

It is easy to see that the Shapley value of $\boldsymbol{z}_i$ is the average of marginal contributions over the two groups. The marginal contributions in $G_{\mathcal{L}}^i$ can be inherited from the original dataset. We propose a new representation of Shapley value as shown in Lemma 1.

Lemma 1: In datasets $\mathcal{D} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$, for $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, denote by

$$\begin{aligned}\mathcal{LSV}_i^+ = &\frac{1}{(n+1)!} \sum_{\pi^k \in G_L^i} [\mathcal{U}(\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(j)}\}) \\ &- \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(j-1)}\})]\end{aligned}$$

the average marginal contribution in group G_L^i and by

$$\begin{aligned}\mathcal{RSV}_i^+ = &\frac{1}{(n+1)!} \sum_{\pi^k \in G_R^i} [\mathcal{U}(\{\{\boldsymbol{z}_{n+1}, \boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(j)}\}) \\ &- \mathcal{U}(\{\{\boldsymbol{z}_{n+1}, \boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(j-1)}\})]\end{aligned}$$

the average marginal contribution in group G_R^i, $\boldsymbol{z}_{\pi^k(j)}$ is $\boldsymbol{z}_i$. Then, the Shapley value of $\boldsymbol{z}_i$ in $\mathcal{N}^+$ is $\mathcal{SV}_i^+ = \mathcal{LSV}_i^+ + \mathcal{RSV}_i^+$.

We first compute Shapley value on $\mathcal{D}$ adopting the Monte Carlo algorithm and store $\mathcal{LSV}_i^+$ $(1 \leq i \leq n)$ at the same time. This computation is performed only once, which is described in Algorithm 2. Denote by τ_1 the number of sampled permutations. We follow the steps of Algorithm 1 to compute $\mathcal{SV}$ on $\mathcal{D}$. Then we compute $\mathcal{LSV}^+$. In line 1, we define $\mathcal{LSV}_i^+$ to store marginal contributions in $G_{\mathcal{L}}^i$. In lines 8-9, we design a uniform sampling distribution for the position index of the new data point and accumulate $\mathcal{LSV}^+$.

Algorithm 2: Initialization (computing $\mathcal{SV}$ in $\mathcal{D}$).

input : datasets $\mathcal{D} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and sample size $\tau > 0$
output: $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, $\mathcal{LSV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, the set of permutations $\pi^1, \ldots, \pi^{\tau_1}$, and the set of indexes $t^1, \ldots, t^{\tau_1}$

1 $\mathcal{SV}_i, \mathcal{LSV}_i^+ \leftarrow 0$ $(1 \leq i \leq n)$;
2 **for** *k=1 to* τ_1 **do**
3 let π^k be a random permutation of $\{1, \ldots, n\}$;
4 let t^k be an integer uniformly randomly drawn from $\{0, \ldots, n\}$;
5 **for** *i=1 to n* **do**
6 $\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}\})$;
7 $\mathcal{SV}_{\pi^k(i)} += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
8 **if** $i \leq t^k$ **then**
9 $\mathcal{LSV}^+_{\pi^k(i)} += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
10 **for** *i=1 to n* **do**
11 $\mathcal{LSV}_i^+ /= \tau_1$;
12 $\mathcal{SV}_i /= \tau_1$;
13 **return** $\mathcal{SV}_1, \ldots, \mathcal{SV}_n, \mathcal{LSV}_1^+, \ldots, \mathcal{LSV}_n^+, \pi^1, \ldots, \pi^{\tau_1}, t^1, \ldots, t^{\tau_1}$;

We then develop two incremental algorithms which aim to compute Shapley value on $\mathcal{N}^+$. $\mathcal{SV}_i^+$ can be obtained by summing $\mathcal{LSV}_i^+$ and $\mathcal{RSV}_i^+$ which can be approximated by sampling the same or different permutations. The algorithm with the same permutations is shown in Algorithm 3. We store the set of permutations $\pi^1, \ldots, \pi^{\tau_1}$ and the set of indexes $t^1, \ldots, t^{\tau_1}$ used in computing $\mathcal{SV}$ on $\mathcal{D}$, and take them as input. In line 4, we update π^k by inserting $\boldsymbol{z}_{n+1}$ at t^k. In lines 7-8, we calculate marginal contributions containing $\boldsymbol{z}_{n+1}$ for $\mathcal{RSV}_i^+$. In lines 11-12, we combine $\mathcal{LSV}_i^+$ and $\mathcal{RSV}_i^+$ to get $\mathcal{SV}_i^+$. In lines 15-16, we update t to p because p is the set of indexes corresponding to new permutations.

Given the same number of sampled permutations, sampling the same permutations can provide more accurate Shapley value than sampling different permutations. However, if more sampled permutations for $\mathcal{LSV}$ are allowed, sampling different permutations outperforms sampling the same permutations in terms of space cost and accuracy. First, no storage of permutations is required for sampling different permutations, which can save space. Second, as the size of coalitions increases, the change in model utility by adding a data point becomes smaller. Therefore, $\mathcal{LSV}_i^+$ plays a more important role in $\mathcal{SV}_i$ than $\mathcal{RSV}_i^+$. To improve the accuracy, we can sample more permutations when computing $\mathcal{LSV}_i^+$ offline, which does not affect the time cost of the online dynamic processing. Moreover, we can sample fewer permutations when computing $\mathcal{RSV}_i^+$ online due to the limitation of time

Algorithm 3: The pivot-based algorithm with the same sampled permutations.

input : datasets $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$, the set of permutations $\pi^1, \ldots, \pi^{\tau_1}$, the set of indexes $t^1, \ldots, t^{\tau_1}$, and sample size $\tau_1 > 0$
output: Shapley value $\mathcal{SV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n+1)$, $\mathcal{LSV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n+1)$, a set of permutations $\pi^1, \ldots, \pi^{\tau_1}$, and the set of indexes $t^1, \ldots, t^{\tau_1}$

1 $\mathcal{SV}_i^+, \mathcal{RSV}_i^+, \Delta\mathcal{LSV}_i^+ \leftarrow 0$ $(1 \leq i \leq n+1)$;
2 $\mathcal{LSV}_{n+1}^+ \leftarrow 0$;
3 **for** $k = 1$ *to* τ_1 **do**
4 $\pi^k \leftarrow \{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(t^k-1)}, \boldsymbol{z}_{n+1}, \boldsymbol{z}_{\pi^k(t^k)}, \ldots, \boldsymbol{z}_{\pi^k(n)}\}$;
5 let p^k be an integer uniformly randomly drawn from $\{0, \ldots, n+1\}$;
6 **for** $i = t^k$ *to* $n+1$ **do**
7 $\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}\})$;
8 $\mathcal{RSV}_i^+ += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
9 **if** $i \leq p^k$ **then**
10 $\Delta\mathcal{LSV}^+_{\pi^k(i)} += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
11 **for** *i=1 to n+1* **do**
12 $\mathcal{SV}^+_{\pi^k(i)} = \mathcal{LSV}_i^+ + \mathcal{RSV}_i^+ / \tau_1$;
13 **for** *i=1 to n+1* **do**
14 $\mathcal{LSV}_i^+ = \frac{2}{3}\mathcal{LSV}_i^+ + \Delta\mathcal{LSV}^+_{\pi^k(i)} / \tau_1$;
15 **for** *i=1 to* τ_1 **do**
16 $t^i = p^i$;
17 **return** $\mathcal{SV}_1^+, \ldots, \mathcal{SV}_{n+1}^+, \mathcal{LSV}_1^+, \ldots, \mathcal{LSV}_{n+1}^+, \pi^1, \ldots, \pi^{\tau_1}, t^1, \ldots, t^{\tau_1}$;

cost. These important advantages of sampling different permutations lead to computing $\mathcal{LSV}_i^+$ and $\mathcal{RSV}_i^+$ with different sampled permutations.

Algorithm 4 shows the processing of computing $\mathcal{RSV}_i^+$ and $\mathcal{SV}_i^+$ $(1 \leq i \leq n+1)$ via sampling different permutations. Denote by τ_2 the number of sampled permutations. In line 1, we define $\mathcal{RSV}_i^+$ to store marginal contributions in $G_{\mathcal{R}}^i$. We begin with drawing permutations sampled from a uniform distribution. In line 5, we find the position index t of the added data point. In lines 8-9, we scan each permutation from t to $n+1$, and calculate marginal contributions to approximate $\mathcal{RSV}_i^+$. Finally, we derive the new Shapley value via summing $\mathcal{LSV}_i^+$ and $\mathcal{RSV}_i^+$ in lines 12-13.

$\mathcal{LSV}_i^+$ should be constantly updated according to the dynamic changes of the dataset. In Algorithm 3 and Algorithm 4, we update $\mathcal{LSV}^+$ in the same way. Taking Algorithm 4 as an example, we record the changes of $\mathcal{LSV}_i^+$ as $\Delta\mathcal{LSV}_i^+$ in line 1. Suppose that we add data point $\boldsymbol{z}_{n+2}$ after adding data point $\boldsymbol{z}_{n+1}$. For data point $\boldsymbol{z}_i$ $(0 < i < n)$, we can construct permutations $[\boldsymbol{z}_i, \boldsymbol{z}_{n+1}, \boldsymbol{z}_{n+2}]$, $[\boldsymbol{z}_i, \boldsymbol{z}_{n+2}, \boldsymbol{z}_{n+1}]$, and $[\boldsymbol{z}_{n+2}, \boldsymbol{z}_i, \boldsymbol{z}_{n+1}]$, which are computed in $\mathcal{LSV}_i^+$. We take $\frac{2}{3}\mathcal{LSV}_i^+$ because only $[\boldsymbol{z}_i, \boldsymbol{z}_{n+1}, \boldsymbol{z}_{n+2}]$ and $[\boldsymbol{z}_i, \boldsymbol{z}_{n+2}, \boldsymbol{z}_{n+1}]$ are still valid for computing $\mathcal{LSV}_i^+$ after adding $\boldsymbol{z}_{n+2}$. We then compute marginal contributions in permutations like $[\boldsymbol{z}_{n+1}, \boldsymbol{z}_i, \boldsymbol{z}_{n+2}]$. In line 6, we uniformly randomly draw the position index p of $\boldsymbol{z}_{n+2}$. In lines 10-11, we obtain $\Delta\mathcal{LSV}_i^+$. In lines 14-15, we update $\mathcal{LSV}_i^+$ by summing up $\frac{2}{3}\mathcal{LSV}_i^+$ and $\Delta\mathcal{LSV}_i^+$.

Example 1: Given $\mathcal{D} = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ and $\mathcal{N}^+ = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}$, we compute $\mathcal{SV}_i^+$ for $\boldsymbol{z}_i \in \mathcal{N}^+$. In Algorithm 2 with $\tau_1 = 2$, suppose that we sample $\pi^1 = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ and $t^1 = 1$ at $k = 1$. We can get

$\mathcal{LSV}_1^+ = \mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\emptyset)$ and $\mathcal{LSV}_2^+ = 0$. Suppose that we sample $\pi^2 = \{\boldsymbol{z}_2, \boldsymbol{z}_1\}$ and $t^2 = 2$ at $k = 2$. We can get $\mathcal{LSV}_1^+ = \mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\emptyset) + \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_2\})$ and $\mathcal{LSV}_2^+ = \mathcal{U}(\{\boldsymbol{z}_2\}) - \mathcal{U}(\emptyset)$. In Algorithm 4 with $\tau_2 = 2$, suppose that we sample $\pi^1 = \{\boldsymbol{z}_1, \boldsymbol{z}_3, \boldsymbol{z}_2\}$ at $k = 1$. We can get $\mathcal{RSV}_1^+ = 0$, $\mathcal{RSV}_2^+ = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\})$, and $\mathcal{RSV}_3^+ = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1\})$. We then sample $\pi^2 = \{\boldsymbol{z}_3, \boldsymbol{z}_2, \boldsymbol{z}_1\}$ at $k = 2$. We can get $\mathcal{RSV}_1^+ = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_2, \boldsymbol{z}_3\})$, $\mathcal{RSV}_2^+ = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) + \mathcal{U}(\{\boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_3\})$, and $\mathcal{RSV}_3^+ = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1\}) + \mathcal{U}(\{\boldsymbol{z}_3\}) - \mathcal{U}(\emptyset)$. Finally, we get $\mathcal{SV}_1^+ = \mathcal{LSV}_1^+ + \mathcal{RSV}_1^+$, $\mathcal{SV}_2^+ = \mathcal{LSV}_2^+ + \mathcal{RSV}_2^+$, and $\mathcal{SV}_3^+ = \mathcal{RSV}_3^+$.

Algorithm 4: The pivot-based algorithm with different sampled permutations.

input : datasets $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$, $\mathcal{LSV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le n)$, and sample size $\tau_2 > 0$
output: Shapley value $\mathcal{SV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le n+1)$ and $\mathcal{LSV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le n+1)$

1 $\mathcal{SV}_i^+, \mathcal{RSV}_i^+, \Delta\mathcal{LSV}_i^+ \leftarrow 0$ $(1 \le i \le n+1)$;
2 $\mathcal{LSV}_{n+1}^+ \leftarrow 0$;
3 **for** *k=1 to* τ_2 **do**
4 let π^k be a random permutation of $\{0, \ldots, n+1\}$;
5 let t be the index of $\boldsymbol{z}_{n+1}$ in π^k;
6 let p be an integer uniformly randomly drawn from $\{0, \ldots, n+1\}$;
7 **for** *i=t to n+1* **do**
8 $\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}\})$;
9 $\mathcal{RSV}_{\pi^k(i)}^+ += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
10 **if** $i \le p$ **then**
11 $\Delta\mathcal{LSV}_{\pi^k(i)}^+ += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
12 **for** *i=1 to n+1* **do**
13 $\mathcal{SV}_i^+ = \mathcal{LSV}_i^+ + \mathcal{RSV}_i^+ / \tau_2$;
14 **for** *i=1 to n+1* **do**
15 $\mathcal{LSV}_i^+ = \frac{2}{3}\mathcal{LSV}_i^+ + \Delta\mathcal{LSV}_{\pi^k(i)}^+ / \tau_2$;
16 **return** $\mathcal{SV}_1^+, \ldots, \mathcal{SV}_{n+1}^+, \mathcal{LSV}_1^+, \ldots, \mathcal{LSV}_{n+1}^+$;

By drawing a sufficient number of samples of permutations in line 4 in Algorithm 4, the pivot-based algorithm with different sampled permutations can provide an (ϵ, δ)-approximation for $\mathcal{RSV}_i^+$ of all data points in the updated dataset.

Theorem 1: Algorithm 4 returns an (ϵ, δ)-approximation to $\mathcal{RSV}_i^+$ if the number of sampled permutations τ satisfies $\tau \ge \frac{2r^2 \ln \frac{2}{\delta}}{\epsilon^2}$ with time complexity $O(\frac{Tnr^2 \ln \frac{2}{\delta}}{\epsilon^2})$, where T is the time of training the model once, r is the range of marginal contributions, and n is the size of the new dataset.

Proof 1: Let $\mathcal{U}(\{z_{\pi(1)}, \ldots, z_{\pi(k)}\}) - \mathcal{U}(\{z_{\pi(1)}, \ldots, z_{\pi(k-1)}\}$ be R. Given $R \in [-r, r]$ $(r \ge 0)$, an error bound ϵ, and a confidence $1 - \delta$, according to Hoeffding's inequality, the number of sampled permutations required such that $P(|\overline{R} - E(\overline{\mathcal{RSV}_i^+})| \ge \epsilon) \le \delta$ is $P(|\frac{1}{\tau}\sum_{k=1}^{\tau} R - E(\overline{\mathcal{RSV}_i^+})| \ge \epsilon) \le 2\exp(-\frac{2\tau^2\epsilon^2}{\sum_{k=1}^{\tau}(2r)^2})$.

Since we want the right hand side to be at most δ, we have $\tau \ge \frac{2r^2 \ln \frac{2}{\delta}}{\epsilon^2}$. Therefore, the time complexity is $O(\frac{Tnr^2 \ln \frac{2}{\delta}}{\epsilon^2})$.

B. The Delta-based Algorithm

As sampling-based methods in approximating Shapley value raise the question of finding a compromise between accuracy and computation time, the approach that gives better results with a sample of the same size is always desired. Generally speaking, the sample size needed for achieving the stability of a variable with a small range is smaller than that of a larger one according to Hoeffding's inequality. In this part, we propose an approach that requires a smaller sample to achieve the same accuracy by representing the difference of Shapley value with the differential marginal contribution, whose absolute value is smaller than the marginal contribution. Given the original data points with previous Shapley values, the key idea is to compute the relative changes of their Shapley values instead of their absolute values. Utilizing the definition of Shapley value, the difference between the precomputed Shapley value and the new Shapley value for each original data point can be represented formally as Lemma 2.

Lemma 2: In datasets $\mathcal{D} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$ and $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$, for any $\boldsymbol{z}_i \in \mathcal{D}$, the difference between the new Shapley value and the precomputed Shapley value of $\boldsymbol{z}_i$ is $\Delta\mathcal{SV}_i = \frac{1}{n+1}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{|\mathcal{S}|+1}{n\binom{n-1}{|\mathcal{S}|}}\{[\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})]\}$, where $[\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})]$ denotes the differential marginal contribution.

Proof 2:

$$\begin{aligned}
\Delta\mathcal{SV}_i &= \frac{1}{n+1}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\})}{\binom{n}{|\mathcal{S}|+1}} \\
&+ \frac{1}{n+1}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})}{\binom{n}{|\mathcal{S}|}} - \frac{1}{n}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})}{\binom{n-1}{|\mathcal{S}|}} \\
&= \frac{1}{n+1}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\})}{\binom{n}{|\mathcal{S}|+1}} \\
&- \sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{(|\mathcal{S}|+1)!(n-1-|\mathcal{S}|)!}{(n+1)!}[\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S})] \\
&= \frac{1}{n+1}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\boldsymbol{z}_i} \frac{|\mathcal{S}|+1}{n\binom{n-1}{|\mathcal{S}|}}\{[\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\})] \\
&- [\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})]\}.
\end{aligned}$$

Based on Lemma 2, we present the delta-based algorithm based on the precomputed Shapley value and the difference of Shapley value. It is shown in Algorithm 5 in detail. The computation starts with the given precomputed Shapley value. In line 4, we randomly sample permutations of all original data points. In lines 5-7, we scan the permutation from the first data point to the last data point, and then construct and calculate the differential marginal contribution of each data point. In line 8, we estimate the Shapley value of the new data point via marginal contributions. Repeating the same procedure over multiple permutations, the difference of Shapley value can be approximated by the average of all the calculated differential marginal contributions. Finally, in lines 9-10, it combines the precomputed Shapley value and the Shapley value difference to infer the new Shapley value of original data points. As Shapley value computation is reduced to the estimation of the average differential marginal contributions, fewer sampled permutations are required to achieve the same accuracy.

Example 2: Given $\mathcal{D} = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ and $\mathcal{N}^+ = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}$, we compute $\Delta\mathcal{SV}_i$ $(1 \le i \le 2)$. In Algorithm 5 with $\tau = 2$, suppose that we sample $\pi^1 = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ at $k = 1$. We can get $\Delta\mathcal{SV}_1 = \frac{1}{6}[\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\{\boldsymbol{z}_3\}) + \mathcal{U}(\emptyset)]$ and $\Delta\mathcal{SV}_2 = \frac{1}{3}[\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) + \mathcal{U}(\{\boldsymbol{z}_1\})]$. Suppose that we then sample $\pi^2 = \{\boldsymbol{z}_2, \boldsymbol{z}_1\}$ at $k = 2$. We can get $\Delta\mathcal{SV}_1 = \frac{1}{6}[\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1\}) - \mathcal{U}(\{\boldsymbol{z}_3\}) + \mathcal{U}(\emptyset)] + \frac{1}{3}[\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_2, \boldsymbol{z}_3\}) + \mathcal{U}(\{\boldsymbol{z}_2\})]$ and $\Delta\mathcal{SV}_2 = \frac{1}{3}[\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_3\}) + \mathcal{U}(\{\boldsymbol{z}_1\})] + \frac{1}{6}[\mathcal{U}(\{\boldsymbol{z}_2, \boldsymbol{z}_3\}) - \mathcal{U}(\{\boldsymbol{z}_2\}) - \mathcal{U}(\{\boldsymbol{z}_3\}) + \mathcal{U}(\emptyset)]$.

Algorithm 5: The delta-based algorithm (adding a data point).

input : datasets $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}\}$, $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, and sample size $\tau > 0$
output: Shapley value $\mathcal{SV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n+1)$

$\Delta\mathcal{SV}_i \leftarrow 0$ $(1 \leq i \leq n)$;
$\mathcal{SV}_i^+ \leftarrow 0$ $(1 \leq i \leq n+1)$;
for *k=1 to* τ **do**
  let π^k be a random permutation of $\{1, \ldots, n\}$;
  for *i=1 to n* **do**
    $\Delta\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = [\mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}, \boldsymbol{z}_{n+1}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\})] - [\mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}, \boldsymbol{z}_{n+1}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i-1)}\})]$;
    $\Delta\mathcal{SV}_{\pi^k(i)} += \Delta\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})/\tau/(n+1) \cdot i$;
    $\mathcal{SV}_{n+1}^+ += [\mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}, \boldsymbol{z}_{n+1}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \ldots, \boldsymbol{z}_{\pi^k(i)}\})]/n\tau$;
for *k=1 to n* **do**
  $\mathcal{SV}_k^+ = \mathcal{SV}_k + \Delta\mathcal{SV}_k$;
return $\mathcal{SV}_1^+, \ldots, \mathcal{SV}_{n+1}^+$;

The following theorem provides an error bound on the number of sampled permutations τ needed to achieve an (ϵ, δ)-approximation.

Theorem 2: Algorithm 5 returns an (ϵ, δ)-approximation to $\Delta\mathcal{SV}_i$ if the number of sampled permutations τ satisfies $\tau \geq \frac{2n^2d^2\ln\frac{2}{\delta}}{(n+1)^2\epsilon^2}$ with time complexity $O(\frac{Tnd^2\ln\frac{2}{\delta}}{\epsilon^2})$, where n is the size of the original dataset, and d is the range of differential marginal contributions.

Proof 3: Let ϕ_i^k denote $\frac{k+1}{n+1}\{[\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\} \cup \{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})]\}$, where $|S| = k$. Suppose that ϕ_i^k is drawn with the following sampler. First, uniformly sample the length k from $0, 1, \ldots, n-1$. Second, uniformly sample a coalition $\mathcal{S}$ from $\binom{n-1}{k}$ all possible length-k coalitions.

$$
\begin{aligned}
E(\phi_i^k) &= \sum_{k=0}^{n-1} \frac{1}{n} \frac{1}{\binom{n-1}{k}} \sum_{\boldsymbol{z}_i \notin S, |S|=k} \phi_i^k \\
&= \frac{1}{n+1} \sum_{\mathcal{S} \subseteq \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\} \setminus \boldsymbol{z}_i} \frac{|S|+1}{n\binom{n-1}{|\mathcal{S}|}} \{[\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\} \cup \{\boldsymbol{z}_i\}) \\
&\quad - \mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})]\} = \Delta\mathcal{SV}_i
\end{aligned}
$$

Now, $E(\phi_i^k) = \Delta\mathcal{SV}_i$. Given $[\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\} \cup \{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S} \cup \{\boldsymbol{z}_{n+1}\}) - \mathcal{U}(\mathcal{S})] \in [-d, d]$ $(d \geq 0)$, an error bound ϵ, and a confidence $1 - \delta$, according to Hoeffding's inequality, the sample size required such that $P(\overline{\phi_i^k} - E(\overline{\Delta\mathcal{SV}_i}) \geq \epsilon) \leq \delta$ is:

$$
\begin{aligned}
P(|\frac{1}{\tau}\sum_{k=1}^{\tau} \phi_i^k - E(\overline{\Delta\mathcal{SV}_i})| \geq \epsilon) &\leq 2\exp(-\frac{2\tau^2\epsilon^2}{\sum_{k=1}^{\tau} \frac{(k+1)^2}{(n+1)^2}(2d)^2}) \\
&\leq 2\exp(-\frac{\tau^2\epsilon^2}{2\tau d^2 \frac{n^2}{(n+1)^2}}).
\end{aligned}
$$

Since we want the right hand side to be at most δ, we have $\tau \geq \frac{2n^2d^2\ln\frac{2}{\delta}}{(n+1)^2\epsilon^2}$. For τ random sampled permutations, the number of model training required is $2n\tau$. Therefore, the time complexity is $O(\frac{Tnd^2\ln\frac{2}{\delta}}{\epsilon^2})$.

The following theorem provides that Algorithm 5 can achieve a convergence rate of $O(1/\sqrt{\tau})$.

Theorem 3: Algorithm 5 returns an unbiased estimator of Shapley value difference with a convergence rate of $O(1/\sqrt{\tau})$, which is the same as the convergence rate of the Monte Carlo algorithm and the pivot-based algorithm with different sampled permutations, where τ is the number of sampled permutations.

Proof 4: As the calculation of $\Delta\mathcal{SV}_i$ is a discretization method, the convergence rate of $\Delta\mathcal{SV}_i$ should be $E[\frac{1}{\tau}\sum_{t=1}^{\tau} X_t - \overline{\Delta\mathcal{SV}_i}|]$, where X_t represents the sampled value of ϕ_i^k. According to Central Limit Theorem, $E[(\frac{1}{\tau}\sum_{t=1}^{\tau} X_t - \overline{\Delta\mathcal{SV}_i})^2] = \frac{\delta_d^2}{\tau}$, where δ_d^2 represents the variance of ϕ_i^k. Meanwhile, for any random variable Z, $E[|Z|]^2 \leq E[Z^2]$. So $E[|\frac{1}{\tau}\sum_{t=1}^{\tau} X_t - \overline{\Delta\mathcal{SV}_i}|]^2 \leq E[(\frac{1}{\tau}\sum_{t=1}^{\tau} X_t - \overline{\Delta\mathcal{SV}_i})^2] = \frac{\delta_d^2}{\tau}$. Obviously, $E[|\frac{1}{\tau}\sum_{t=1}^{\tau} X_t - \overline{\Delta\mathcal{SV}_i}|] \leq \frac{\delta_d}{\sqrt{\tau}} = O(1/\sqrt{\tau})$. The same conclusion can be obtained that the convergence rate of Shapley value calculated by the Monte Carlo algorithm or the pivot-based algorithm with different sampled permutations is $E[\frac{1}{\tau}\sum_{t=1}^{\tau} Y_t - \overline{\mathcal{SV}_i}|] \leq \frac{\delta_m^2}{\tau}$, where Y_t represents the sampled value of R and δ_m^2 represents the variance of R. It is worth noting that because δ_d^2 is smaller than δ_m^2, the actual convergence rate of the delta-based algorithm is better than both the pivot-based algorithm and the Monte Carlo algorithm.

C. Adding Multiple Data Points

Update operation of adding multiple data points can be thought of as gradually adding a single data point at a time. Thereby, the problem of updating Shapley value when adding multiple data points can be solved by applying Algorithm 3, Algorithm 4, or Algorithm 5 progressively. However, applying the pivot-based algorithm or the delta-based algorithm while adding a large amount of data points could be more time-consuming than using the Monte Carlo algorithm once. Thereby, we introduce heuristic algorithms with stable time cost in Section VI.

V. Deleting data points

In this section, we tackle the case of deleting data points. The marginal contributions in the original dataset cover all marginal contributions in the new dataset. Motivated by this, we design the YN-NN algorithm with a polynomial-time bound based on dynamic data structure which maintains marginal contributions of unchanged data points in an efficient way in Section V-A. In Section V-B, we present the delta-based algorithm based on differential marginal contributions. In Section V-C, we show how to extend our algorithms towards deleting multiple data points. The two algorithms are proposed from two angles. The YN-NN algorithm aims to reduce unnecessary computation, while the delta-based algorithm aims to reduce the number of sampled permutations.

A. The YN-NN Algorithm

To avoid unnecessary marginal contribution computation, we capture the storage of utilities which are performed before

data points are changed. Our proposed solution contains two phases: *Phase Preprocessing* - computing Shapley value over the original dataset and *Phase Merging* - updating Shapley value over the new dataset.

Phase Preprocessing is performed only once. We design and utilize two three-dimensional arrays: $\mathcal{YN}$ and $\mathcal{NN}$, which store the utilities for all data points. $\mathcal{Y}$ refers to "$\mathcal{Y}$es" and $\mathcal{N}$ refers to "$\mathcal{N}$o". We give the definition of $\mathcal{YN}$ and $\mathcal{NN}$ as follows.

Definition 1: Given a dataset of n data points $\mathcal{D} = \{\boldsymbol{z}_1 \dots \boldsymbol{z}_n\}$,

$$\mathcal{YN}[\boldsymbol{z}_i][\boldsymbol{z}_j][k] = \sum_{\mathcal{S}\subseteq\mathcal{D},|\mathcal{S}|=k,\boldsymbol{z}_i\in\mathcal{S},\boldsymbol{z}_j\notin\mathcal{S}} \mathcal{U}(\mathcal{S})$$

$$\mathcal{NN}[\boldsymbol{z}_i][\boldsymbol{z}_j][k] = \sum_{\mathcal{S}\subseteq\mathcal{D},|\mathcal{S}|=k,\boldsymbol{z}_i\notin\mathcal{S},\boldsymbol{z}_j\notin\mathcal{S}} \mathcal{U}(\mathcal{S})$$

In *Phase Merging*, the aforementioned two arrays $\mathcal{YN}$ and $\mathcal{NN}$ assist in averting superfluous computation. Based on the definition of Shapley value, one can derive the following formula of the new Shapley value after deleting a data point.

Lemma 3: Suppose that $\boldsymbol{z}_{del}$ is the deleted data point. $\mathcal{N}^- = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\} \setminus \boldsymbol{z}_{del}$. For any remaining data point $\boldsymbol{z}_i \in \mathcal{N}^-$, the new Shapley value is

$$\mathcal{SV}_i^- = \frac{1}{n-1}\sum_{k=1}^{n-1}\frac{(\mathcal{YN}[\boldsymbol{z}_i][\boldsymbol{z}_{del}][k] - \mathcal{NN}[\boldsymbol{z}_i][\boldsymbol{z}_{del}][k-1])}{\binom{n-2}{k-1}}$$

Example 3: Given three patients' data points $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{z}_3\}$ shown in Table II and the deleted data point $\boldsymbol{z}_3$, as for $\boldsymbol{z}_1$, arrays are shown as follows.

$$\begin{aligned}
&\mathcal{YN}[\boldsymbol{z}_1][\boldsymbol{z}_3][0] = \mathcal{U}(\emptyset) && \mathcal{NN}[\boldsymbol{z}_1][\boldsymbol{z}_3][0] = \mathcal{U}(\emptyset)\\
&\mathcal{YN}[\boldsymbol{z}_1][\boldsymbol{z}_3][1] = \mathcal{U}(\{\boldsymbol{z}_1\}) && \mathcal{NN}[\boldsymbol{z}_1][\boldsymbol{z}_3][1] = \mathcal{U}(\{\boldsymbol{z}_2\})\\
&\mathcal{YN}[\boldsymbol{z}_1][\boldsymbol{z}_3][2] = \mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) && \mathcal{NN}[\boldsymbol{z}_1][\boldsymbol{z}_3][2] = \mathcal{U}(\emptyset)
\end{aligned}$$

The new Shapley value of $\boldsymbol{z}_1$ can be calculated as follows.

$$\begin{aligned}
\mathcal{SV}_1^- &= \frac{1}{2}\sum_{k=1}^{2}\frac{\mathcal{YN}[\boldsymbol{z}_1][\boldsymbol{z}_3][k] - \mathcal{NN}[\boldsymbol{z}_1][\boldsymbol{z}_3][k-1]}{\binom{1}{k-1}}\\
&= \frac{1}{2}\mathcal{U}(\{\boldsymbol{z}_1, \boldsymbol{z}_2\}) + \frac{1}{2}\mathcal{U}(\{\boldsymbol{z}_1\}) - \frac{1}{2}\mathcal{U}(\{\boldsymbol{z}_2\}) - \frac{1}{2}\mathcal{U}(\emptyset)
\end{aligned}$$

Based on Lemma 3, the key idea of updating Shapley value is generating the utility arrays. We show the pseudo-code for filling $\mathcal{YN}$ and $\mathcal{NN}$ during computing original Shapley value in Algorithm 6 and show how to derive the new Shapley value from $\mathcal{YN}$ and $\mathcal{NN}$ succinctly in Algorithm 7. Concretely, in Algorithm 6, we store utility functions in two arrays when calculating Shapley value on the original dataset in lines 8-10, which does not lead to extra computational overhead. Then we can recover Shapley value of the remaining data points in $O(n^2)$ whichever a data point is removed in Algorithm 7.

B. The Delta-based Algorithm

Based on similar rationale in Section IV-B, we introduce another version of Lemma 2. Suppose that $\boldsymbol{z}_{del}$ is the deleted data point in the dataset. For any $\boldsymbol{z}_i \in \mathcal{D} \setminus \boldsymbol{z}_{del}$, the difference between the new Shapley value and the original Shapley value of $\boldsymbol{z}_i$ is $\Delta\mathcal{SV}_i = -\frac{1}{n}\sum_{\mathcal{S}\subseteq\mathcal{D}\setminus\{\boldsymbol{z}_i,\boldsymbol{z}_{del}\}}\frac{|\mathcal{S}|+1}{(n-1)\binom{n-2}{|\mathcal{S}|}}\{[\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{del}\}\cup\{\boldsymbol{z}_i\}) - \mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_i\})] - [\mathcal{U}(\mathcal{S}\cup\{\boldsymbol{z}_{del}\}) - \mathcal{U}(\mathcal{S})]\}$. The pseudo-code can be seen in Algorithm 8. We aim at deriving new Shapley value from Shapley value changes. In line 3, we uniformly sample a permutation. In lines 5-6, we calculate the difference of marginal contributions to model Shapley value changes based on differential marginal contributions.

The following theorem provides an error bound on the number of sampled permutations τ needed to achieve an (ϵ, δ)-approximation.

Theorem 4: Algorithm 8 returns an (ϵ, δ)-approximation to $\Delta\mathcal{SV}_i$ if the number of sampled permutations τ satisfies $\tau \geq \frac{2(n-1)^2 d^2 \ln\frac{2}{\delta}}{n^2\epsilon^2}$ with time complexity $O(\frac{Tnd^2\ln\frac{2}{\delta}}{\epsilon^2})$, where n is the size of the original dataset, and d is the range of differential marginal contributions.

Proof 5: The proof is similar to the proof of Theorem 2.

Algorithm 6: *Preprocessing* (deleting a data point).

input : datasets $\mathcal{D} = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\}$, and sample size $\tau > 0$
output: $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, $\mathcal{YN}$, and $\mathcal{NN}$
1 let $\mathcal{YN}$ and $\mathcal{NN}$ be two n^3 arrays;
2 $\mathcal{SV}_i, \mathcal{YN}[\boldsymbol{z}_i][\boldsymbol{z}_j][k], \mathcal{NN}[\boldsymbol{z}_i][\boldsymbol{z}_j][k] \leftarrow 0$ $(1 \leq i, j, k \leq n)$;
3 **for** *k=1 to* τ **do**
4 let π^k be a random permutation of $\{1, \dots, n\}$;
5 **for** *i=1 to n* **do**
6 $\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i)}\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i-1)}\})$;
7 $\mathcal{SV}_{\pi^k(i)} += \mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})$;
8 **for** *j=i to n* **do**
9 $\mathcal{YN}[\boldsymbol{z}_{\pi^k(i)}][\boldsymbol{z}_{\pi^k(j)}][i] += \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i)}\})/\tau$;
10 $\mathcal{NN}[\boldsymbol{z}_{\pi^k(i)}][\boldsymbol{z}_{\pi^k(j)}][i-1] += \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i-1)}\})/\tau$;
11 **return** $\mathcal{SV}_1, \dots, \mathcal{SV}_n, \mathcal{YN}, \mathcal{NN}$;

Algorithm 7: *Merging* (deleting a data point).

input : datasets $\mathcal{N}^- = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\} \setminus \boldsymbol{z}_p$, $\mathcal{YN}$, and $\mathcal{NN}$
output: Shapley value $\mathcal{SV}_i^-$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n, i \neq p)$
1 **for** *i=1 to n* **do**
2 **if** $i \neq p$ **then**
3 **for** *j=1 to n* **do**
4 $\mathcal{SV}_i^- += (\mathcal{YN}[\boldsymbol{z}_i][\boldsymbol{z}_p][j] - \mathcal{NN}[\boldsymbol{z}_i][\boldsymbol{z}_p][j-1]) \cdot (n-1)/(n-j)$;
5 **return** $\mathcal{SV}_1^-, \dots, \mathcal{SV}_n^-$;

Algorithm 8: The delta-based algorithm (deleting a data point).

input : datasets $\mathcal{N}^- = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\} \setminus \boldsymbol{z}_p$, $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$, and sample size $\tau > 0$.
output: Shapley value $\mathcal{SV}_i^-$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n, i \neq p)$
1 $\mathcal{SV}_i^-, \Delta\mathcal{SV}_i \leftarrow 0$ $(1 \leq i \leq n)$;
2 **for** *k=1 to* τ **do**
3 let π^k be a random permutation of $\{1, \dots, n\} \setminus p$;
4 **for** *i=1 to n-1* **do**
5 $\Delta\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)}) = -[\mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i)}, \boldsymbol{z}_p\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i)}\})] + [\mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i-1)}, \boldsymbol{z}_p\}) - \mathcal{U}(\{\boldsymbol{z}_{\pi^k(1)}, \dots, \boldsymbol{z}_{\pi^k(i-1)}\})]$;
6 $\Delta\mathcal{SV}_{\pi^k(i)} += \Delta\mathcal{SV}(\boldsymbol{z}_{\pi^k(i)})/\tau/n \cdot i$;
7 **for** *i=1 to n* **do**
8 $\mathcal{SV}_i^- = \mathcal{SV}_i + \Delta\mathcal{SV}_i$
9 **return** $\mathcal{SV}_1^-, \dots, \mathcal{SV}_n^-$;

C. Deleting Multiple Data Points

In this section, we show how to retrieve Shapley value in a scenario where multiple data points are deleted.

For the YN-NN algorithm, we introduce multiple-dimension utility arrays as follows.

Definition 2: Given a dataset of n data points $\mathcal{D} = \{\boldsymbol{z}_1 \dots \boldsymbol{z}_n\}$,

$$\mathcal{Y}\overbrace{\mathcal{N},\dots,\mathcal{N}}^{d}[\boldsymbol{z}_t]\overbrace{[\boldsymbol{z}_i],\dots,[\boldsymbol{z}_j]}^{d}[k] = \sum_{\substack{\mathcal{S}\subseteq\mathcal{D},|\mathcal{S}|=k,\boldsymbol{z}_t\in\mathcal{S}\\ \boldsymbol{z}_i,\dots,\boldsymbol{z}_j\notin\mathcal{S}}} \mathcal{U}(\mathcal{S})$$

$$\mathcal{N}\overbrace{\mathcal{N},\dots,\mathcal{N}}^{d}[\boldsymbol{z}_t]\overbrace{[\boldsymbol{z}_i],\dots,[\boldsymbol{z}_j]}^{d}[k] = \sum_{\substack{\mathcal{S}\subseteq\mathcal{D},|\mathcal{S}|=k,\boldsymbol{z}_t\notin\mathcal{S}\\ \boldsymbol{z}_i,\dots,\boldsymbol{z}_j\notin\mathcal{S}}} \mathcal{U}(\mathcal{S})$$

Analogously, one can derive the following formula naturally.

Lemma 4: Suppose that $\boldsymbol{z}_p, \dots, \boldsymbol{z}_q$ are d deleted data points. $\mathcal{N}^- = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n\} \setminus \{\boldsymbol{z}_p, \dots, \boldsymbol{z}_q\}$. For any remaining data point $\boldsymbol{z}_i \in \mathcal{N}^-$, the new Shapley value is

$$\mathcal{SV}_i^- = \frac{1}{n-d}\sum_{k=1}^{n-d}\frac{1}{\binom{n-d-1}{k-1}}\Big(\mathcal{Y}\overbrace{\mathcal{N},\dots,\mathcal{N}}^{d}[\boldsymbol{z}_i]\overbrace{[\boldsymbol{z}_p],\dots,[\boldsymbol{z}_q]}^{d}[k] - \mathcal{N}\overbrace{\mathcal{N},\dots,\mathcal{N}}^{d}[\boldsymbol{z}_i]\overbrace{[\boldsymbol{z}_p],\dots,[\boldsymbol{z}_q]}^{d}[k-1]\Big)$$

For deleting d data points, we obtain the two (d+2)-dimension utility arrays in the process of Shapley value computation on the original dataset. Based on Lemma 4, we then derive the new Shapley value via conducting pairwise subtraction operations between the two utility arrays similar to deleting a data point.

For the delta-based algorithm, the problem of updating Shapley value when deleting multiple data points can be solved by applying the delta-based algorithm progressively. Moreover, due to the fact that the delta-based algorithm only requires the original Shapley value and the added/deleted data points, it is natural to apply the delta-based algorithm on dynamic datasets of interspersed addition and deletion.

VI. Heuristic Algorithm

In this section, two different heuristic algorithms are studied to efficiently update Shapley value, which is inspired by our empirical observations of changes in Shapley value caused by dynamic data points on the labeled datasets.

Extrapolating from property *Symmetry*, data points with similar features tend to have a similar performance on machine learning models, which results in similar utility functions and similar Shapley value. Thus, we develop a technique which finds adjacent data points of the added data points powered by k-nearest neighbors algorithm (k-NN) and averages over the Shapley values. The pseudo-code is presented in Algorithm 9. In lines 2-4, we assign the average Shapley value of adjacent data points to the added data points.

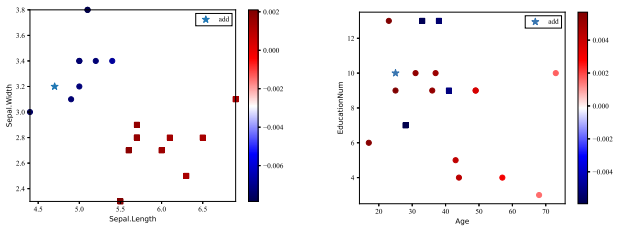
The above algorithm assumes the Shapley values of the original data points do not change when a new data point is added.

Algorithm 9: Heuristic $\mathcal{SV}$ computation (KNN).

input : datasets $\mathcal{N}^+ = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}, \dots, \boldsymbol{z}_m\}$ $(m > n)$, $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le n)$
output: Shapley value $\mathcal{SV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \le i \le m)$

1 $\mathcal{SV}_i^+ \leftarrow 0$ $(1 \le i \le m)$;
2 **for** *i = n+1 to m* **do**
3 let $\boldsymbol{nz}_1, \dots, \boldsymbol{nz}_k$ be k neighbors of $\boldsymbol{z}_i$;
4 $\mathcal{SV}_i^+ = \frac{1}{k}(\mathcal{SV}_{\boldsymbol{nz}_1} + \dots + \mathcal{SV}_{\boldsymbol{nz}_k})$;
5 **for** *i = 1 to n* **do**
6 $\mathcal{SV}_i^+ = \mathcal{SV}_i$;
7 **return** $\mathcal{SV}_1^+, \dots, \mathcal{SV}_m^+$;

However, this is not the case. Figure 2 illustrates the changes of Shapley values of the original data points after adding a new point (star). We observed that the changes depend on their similarity with the new data point. Concretely, when adding a data point, Shapley values of data points with the same label decrease, and those with different labels increase. Moreover, the degree of changes decreases as the similarity distance increases, which can be fitted into a function to represent the corresponding relationship of 1) the similarity between original data points and new data points and 2) the changes of Shapley value.



(a) Iris. (b) Adult.

Fig. 2: Changes of Shapley value.

With the discovery, we develop another heuristic technique, as shown in Algorithm 10. The main idea is to learn a regression function for the changes of Shapley values based on their similarity to the new data point and use this function to derive the updated Shapley values of original data points. In lines 5-7, we sample a set of data points from the whole original dataset and calculate the changes of Shapley value through the Monte Carlo algorithm. In line 8, we fit the changes into a function to represent the relationship between changes and similarity. In lines 9-13, we update Shapley value of original data points based on the changes derived from similarity and assign the average Shapley value of several data points selected by k-NN to new data points.

Both KNN and KNN+ are adoptable for deleting data points. For KNN, we assign Shapley value of deleted data points averagely to their neighbors, which is similar to Algorithm 9. For KNN+, we update Shapley value based on the relationship between the changes of Shapley value and the similarity between original data points and deleted data points, which is similar to Algorithm 10.

Algorithm 10: Heuristic $\mathcal{SV}$ computation (KNN+).

input : datasets $\mathcal{N}^+ = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n, \boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_m\}$ $(m > n)$, $\mathcal{SV}_i$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq n)$

output: Shapley value $\mathcal{SV}_i^+$ for each data point $\boldsymbol{z}_i$ $(1 \leq i \leq m)$

$\mathcal{SV}_i^+ \leftarrow 0$ $(1 \leq i \leq m)$;

$\Delta\mathcal{SV}_i \leftarrow 0$ $(1 \leq i \leq n)$;

for *i = 1 to n* **do**

  $\mathcal{SV}_i^+ = \mathcal{SV}_i$;

sample train data points $\{\boldsymbol{z}_{t_1}, \ldots, \boldsymbol{z}_{t_d}\}$ from train data points $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}$;

for *i = 1 to d* **do**

  $\Delta\mathcal{SV} = MonteCarloShapley(\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}) - MonteCarloShapley(\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\} \setminus \boldsymbol{z}_{t_i})$;

  $CurveFuncs[the\ label\ of\ \boldsymbol{z}_{t_i}] = CurveFitting(simi(\boldsymbol{z}_{t_i}, \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n\}), \Delta\mathcal{SV})$;

for *i=n+1 to m* **do**

  for *j = 1 to n* **do**

    $\mathcal{SV}_j^+$ += $CurveFuncs[the\ label\ of\ \boldsymbol{z}_i](simi(\boldsymbol{z}_i, \boldsymbol{z}_j))$;

  let $\boldsymbol{nz}_1, \ldots, \boldsymbol{nz}_k$ be k neighbors of $\boldsymbol{z}_i$;

  $\mathcal{SV}_i^+ = \frac{1}{k}(\mathcal{SV}_{\boldsymbol{nz}_1} + \ldots + \mathcal{SV}_{\boldsymbol{nz}_k})$;

return $\mathcal{SV}_1^+, \ldots, \mathcal{SV}_m^+$;

VII. Experiments

A. Experiment Setup

We ran experiments on a machine with 2 Montage(R) Jintide(R) C6226R @ 2.90GHz and 4 Geforce RTX 3090 running Ubuntu 18.04 LTS 64-bit with 256GB memory. Support Vector Machine (SVM) is employed as the machine learning model, and the utility function is the accuracy score of the trained SVM model on the test dataset. We used both Iris dataset and Adult dataset from the UCI machine learning repository [14] in our experiments. We adopt the Monte Carlo algorithm as our benchmark, which is the universal baseline method for Shapley value computation [19]. Even the state-of-the-art Monte Carlo algorithm cannot support large datasets. It is hard to obtain a sufficiently accurate Shapley value in tolerable time for comparison on a large dataset. To make it feasible, we performed analysis on datasets by sampling 10000 data points at most. Our algorithms can be applied to larger datasets in practical applications, which are comparable to the state-of-the-art Monte Carlo algorithm because our algorithms are much faster than the Monte Carlo algorithm in general when they are used to compute fairly accurate dynamic Shapley value.

For adding a data point, we compare the baseline algorithms including MC (Alg. 1), Base which adopts original Shapley value and assigns the average Shapley value of all data points to the added data point, and TMC developed by [19] (the performance tolerance is set to 1e-12 and the truncated point must be in $[n/2, n]$) with the proposed algorithms including Pivot-s (Alg. 3), Pivot-d (Alg. 4), Delta (Alg. 5), KNN (Alg. 9), and KNN+ (Alg. 10).

For adding multiple data points, we compare the baseline algorithms including MC, Base, and TMC with the proposed algorithms including Pivot-s (Alg. 3), Pivot-d (Alg. 4), Delta (Alg. 5), KNN (Alg. 9), and KNN+ (Alg. 10).

For deleting a data point, we compare the baseline algorithms including MC and TMC with the proposed algorithms including YN-NN (Alg. 7), Delta (Alg. 8), a variant of KNN, and a variant of KNN+. The last two algorithms are briefly discussed in Section VI.

For deleting multiple data points, we compare the baseline algorithms including MC and TMC with the proposed algorithms including YNN-NNN demonstrated in Section V-C which is the extension of Alg. 4, Delta (Alg. 8), a variant of KNN, and a variant of KNN+.

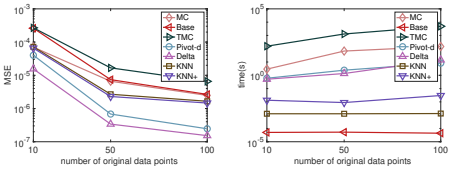
We compute the p-value [2, 8] of the differences between the MSEs of our algorithms and MC for the following experiments. All p-values are much smaller than 0.05, confirming the statistical significance of the difference.

B. Evaluation Metrics

Effectiveness. We adopt the average of the mean squared errors (MSEs) to verify the effectiveness of the proposed algorithms. Given benchmark Shapley value $\mathcal{SV}_i$ and estimated Shapley value $\overline{\mathcal{SV}_i}$ $(1 \leq i \leq n)$ computed by the proposed algorithms, the mean squared error for the estimated Shapley value compared to the benchmark Shapley value is $MSE(\mathcal{SV}, \overline{\mathcal{SV}}) = \frac{1}{n}\sum_{i=1}^{n}(\mathcal{SV}_i - \overline{\mathcal{SV}_i})^2$. Computing the exact Shapley value $\mathcal{SV}_i$ for evaluation purposes is prohibitively expensive because it grows exponentially with the number of data points. Therefore, we use the estimated Shapley value computed by Algorithm 1 with $\tau = 1000n$ as the benchmark Shapley value in the experiments.

Efficiency. We propose a time-based metric to verify the efficiency of the proposed algorithms. Given benchmark Shapley value $\mathcal{MCSV}+$ and $\mathcal{MCSV}$ computed by Algorithm 1 with $\tau = 1000n$ and $\tau = 20n$ respectively, and estimated Shapley value $\overline{\mathcal{SV}}$ computed by the proposed algorithms, we simulate the number of sampled permutations and its corresponding runtime to achieve $|MSE(\mathcal{MCSV}+, \mathcal{MCSV}) - MSE(\mathcal{MCSV}+, \overline{\mathcal{SV}})| < 0.2MSE(\mathcal{MCSV}+, \mathcal{MCSV})$.

C. Adding A Data Point



(a) MSEs. (b) Time cost.

Fig. 3: Adding a data point.

TABLE IV: MSEs for adding a data point.

MC	Base	TMC	Pivot-d	Delta	KNN	KNN+
2.48e-6	2.66e-6	5.47e-5	2.45e-7	1.53e-7	1.65e-6	1.46e-6

TABLE V: MSEs for adding a data point.

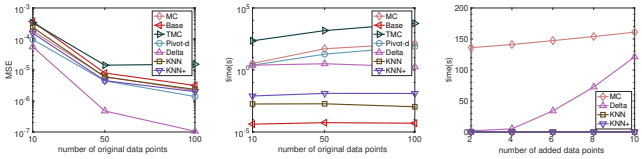
	$\tau_{\mathcal{LSV}} = 20n$ $\tau_{\mathcal{RSV}} = 20n$	$\tau_{\mathcal{LSV}} = 100n$ $\tau_{\mathcal{RSV}} = 20n$	$\tau_{\mathcal{LSV}} = 500n$ $\tau_{\mathcal{RSV}} = 20n$
Pivot-s	2.18e-6	N/A	N/A
Pivot-d	2.36e-6	5.86e-7	3.01e-7

Effectiveness. We first study the effectiveness of the proposed algorithms for adding a data point on Iris dataset. We take a sample of size 100 and add one data point further. In

order to make a fair comparison, we set $\tau_{MC} = \tau_{TMC} = \tau_{Pivot-d} = \tau_{Delta} = 20n$. Table IV presents the MSEs of the proposed algorithms. We can observe that Pivot-d and Delta outperform KNN and KNN+, and the MSE of Delta is smaller than that of Pivot-d. KNN and KNN+ significantly perform better than baseline algorithms, although they cannot recover the new Shapley value as well as Pivot-d and Delta. Figure 3(a) shows the MSEs on varying numbers of original data points. We can find that the MSEs of Pivot-d and Delta are always lower than baseline algorithms. We compare Pivot-s and Pivot-d when sampling different numbers of permutations to estimate $\mathcal{LSV}$. Experimental results are shown in Table V. When $\tau_{\mathcal{LSV}} = \tau_{\mathcal{RSV}}$, Pivot-s outperforms Pivot-d. When $\tau_{\mathcal{LSV}} \neq \tau_{\mathcal{RSV}}$, Pivot-s is not applicable since it requires the same number of sampled permutations for $\mathcal{LSV}$ and $\mathcal{RSV}$. As $\tau_{\mathcal{LSV}}$ increases, the MSE of Pivot-d gets smaller than Pivot-s.

Efficiency. We experimentally study the efficiency of the proposed algorithms for adding a data point on Iris dataset. Figure 3(b) shows the time cost of algorithms on varying numbers of original data points. Base only needs to calculate the average Shapley value of original data points. The MSEs of results given by KNN, and KNN+ are not involved with the number of sampled permutations. Thus, the time cost of Base, KNN, and KNN+ is very low and their output cannot guarantee the output quality. The time cost of Pivot-d and Delta is consistently less than that of MC and TMC with the number of data points, which verifies the efficiency of the proposed algorithms.

D. Adding Multiple Data Points



(a) MSEs. (b) Time cost. (c) Time cost.

Fig. 4: Adding multiple data points.

TABLE VI: MSEs for adding two data points.

MC	Base	TMC	Pivot-d	Delta	KNN	KNN+
2.17e-6	3.17e-6	1.54e-5	1.39e-6	1.06e-7	2.36e-6	2.00e-6

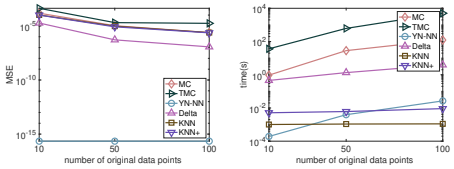
TABLE VII: MSEs for adding two data points.

	$\tau_{\mathcal{LSV}} = 20n$ $\tau_{\mathcal{RSV}} = 20n$	$\tau_{\mathcal{LSV}} = 100n$ $\tau_{\mathcal{RSV}} = 20n$	$\tau_{\mathcal{LSV}} = 500n$ $\tau_{\mathcal{RSV}} = 20n$
Pivot-s	5.98e-6	N/A	N/A
Pivot-d	6.53e-6	6.07e-6	4.88e-6

Effectiveness. We study the effectiveness of the proposed algorithms for adding multiple data points on Iris dataset. We take a sample of size 100 and add two data points further. In order to make a fair comparison, we set $\tau_{MC} = \tau_{TMC} = \tau_{Pivot-d} = \tau_{Delta} = 20n$. Table VI presents the MSEs of the proposed algorithms. We can observe Delta owns the smallest MSE, which means that it is closest to the benchmark Shapley value. Pivot-d owns the second smallest MSE and is the second best algorithm. Since the accuracy of the new round of $\mathcal{LSV}$ is lower than that of the original Shapley value, Delta provides the Shapley value with MSE lower than Pivot-d. Heuristic algorithms, KNN and KNN+, have better performance than Base and TMC. Figure 4(a) shows the MSEs on varying numbers of original data points. We can find that the MSEs of Pivot-d and Delta are always lower than baseline algorithms. Table VII shows findings similar to adding a data point.

Efficiency. We experimentally study the efficiency of the proposed algorithms for adding two data points on Iris dataset. Figure 4(b) shows the time cost of algorithms on varying numbers of original data points. The time cost of MC and TMC is larger than Pivot-d and Delta as anticipated, which further confirms the superiority of Pivot-d and Delta. For adding more data points, we add 2-10 data points to the 100-size dataset sampled from Iris dataset, respectively. As the number of added data points increases, the loss of precision in $\mathcal{LSV}$ makes Pivot-d less and less effective. The time cost is shown in Figure 4(c). KNN and KNN+ can quickly compute the new Shapley value, but not necessarily achieve the required MSE. The time cost for MC and Delta to reach the same level of MSE increases with the number of added data points and Delta achieves greater efficiency.

E. Deleting A Data Point



(a) MSEs. (b) Time cost.

Fig. 5: Deleting a data point.

TABLE VIII: MSEs for deleting a data point.

MC	TMC	YN-NN	Delta	KNN	KNN+
1.34e-6	1.71e-5	0	1.51e-7	1.52e-6	1.30e-6

TABLE IX: Memory consumption.

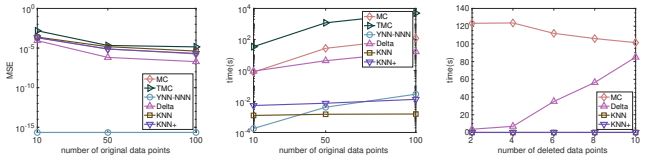
n	10	50	100
cost (MB)	0.014668	1.927675	15.25421

Effectiveness. We experimentally study the effectiveness of the proposed algorithms for deleting a data point on Iris dataset. We take a sample of size 100 and delete one data point further. In order to make a fair comparison, we set $\tau_{MC} = \tau_{TMC} = \tau_{Delta} = 20n$. Table VIII presents the MSEs of the proposed algorithms. YN-NN can recover the exact new benchmark Shapley value. Delta gives the second best result, while KNN and KNN+ perform worse than MC and TMC. Figure 5(a) shows the MSEs on varying number of original data points. We can find that the MSEs of YN-NN and Delta are always lower than baseline algorithms.

Efficiency. We experimentally study the efficiency of the proposed algorithms for deleting a data point on Iris dataset. Figure 5(b) shows the time cost of algorithms on varying numbers of original data points. KNN and KNN+ have a low time cost and cannot guarantee the output quality. Because YN-NN only needs to scan YN and NN once, the time cost is very

low. The time cost of MC and TMC is limited by the number of sampled permutations but grows prohibitively high with the number of data points. In contrast, Delta takes less time than MC and TMC to achieve the satisfying Shapley value approximation. Table IX shows the memory consumption of YN-NN with the same experimental settings as Figure 5(b).

F. Deleting Multiple Data Points



(a) MSEs. (b) Time cost. (c) Time cost.

Fig. 6: Deleting multiple data points.

TABLE X: MSEs for deleting two data points.

MC	TMC	YNN-NNN	Delta	KNN	KNN+
1.83e-6	1.41e-5	0	1.93e-7	4.16e-6	2.10e-6

Effectiveness. We experimentally study the effectiveness of the proposed algorithms for deleting multiple data points on Iris dataset. We take a sample of size 100 and delete two data points further. In order to make a fair comparison, we set $\tau_{MC} = \tau_{TMC} = \tau_{Delta} = 20n$. Table X presents the MSEs of the proposed algorithms. YNN-NNN has the lowest MSE and Delta performs better than MC and TMC, which agrees with what is observed in the previous experiment. Moreover, we can see that KNN and KNN+ have higher MSEs concerning sampling-based algorithms. Figure 6(a) shows the MSEs on varying numbers of original data points. We can find that the MSEs of YNN-NNN and Delta are always lower than baseline algorithms.

Efficiency. We experimentally study the efficiency of the proposed algorithms for deleting two data points on Iris dataset. Figure 6(b) shows the time cost of algorithms on varying numbers of original data points. Because YNN-NNN only needs to scan YNN and NNN once, the time cost is very low. The time cost of MC, TMC, and Delta grows with the number of data points, but Delta still performs better than MC and TMC. For deleting more data points, we delete 2-10 data points from the 100-size dataset sampled from Iris dataset, respectively. The time cost is shown in Figure 6(c). KNN and KNN+ can compute new Shapley value quickly but at the expense of accuracy. Since Delta processes deleted data points sequentially, the time cost of Delta increases with the number of deleted data points, while MC does the opposite. Nevertheless, the time cost of Delta is always less than that of MC when deleting a couple of data points, which implies that Delta can be used for dynamic Shapley value computation.

G. Large Dataset

We compare the time cost of the proposed algorithms on a dataset of size 10000 and 3 features constructed from the Adult dataset. We set $\tau_{MC} = \tau_{TMC} = \tau_{Pivot-d} = \tau_{Delta} = 100$ and $\tau_{MC+} = 1000$. It should be mentioned that MC, TMC, Pivot-d, and Delta are parallelizable and the time cost can become less using more machines in parallel with k threads (here $k = 48$). In Tables XI, XII, XIII, and XIV, we observe the time cost of algorithms for adding or deleting data points and omit the MSE comparison as MC is not converging under this small number of permutations. KNN and KNN+ significantly outperform other algorithms due to their simplicity. Pivot-d has the intermediate time cost. Delta costs a high time because it needs to evaluate more utility functions than MC with the same number of permutations. However, as we have shown in previous results, even given a much smaller number of permutations compared with other algorithms, Delta can produce fairly accurate Shapley value.

TABLE XI: Time cost for adding one data point (s).

MC+	MC	TMC	Pivot-d	Delta	KNN	KNN+
2.08e5	2.10e4	4.89e3	1.83e4	6.32e4	1.43e-3	15.25

TABLE XII: Time cost for adding two data points (s).

MC+	MC	TMC	Pivot-d	Delta	KNN	KNN+
2.08e5	2.10e4	4.89e3	3.24e4	8.88e4	2.03e-3	14.92

TABLE XIII: Time cost for deleting one data point (s).

MC+	MC	TMC	YN-NN	Delta	KNN	KNN+
2.08e5	2.10e4	4.89e3	3.27e2	4.27e4	1.44e-2	16.08

TABLE XIV: Time cost for deleting two data points (s).

MC+	MC	TMC	YNN-NNN	Delta	KNN	KNN+
2.08e5	2.10e4	4.89e3	4.09e2	8.37e4	1.63e-2	30.11

VIII. Conclusion And Future Work

In this paper, for the first time, we proposed the problem of dynamic Shapley value computation and presented approaches that are capable of deriving Shapley value on dynamic datasets. In the case of adding data points, we proposed the pivot-based algorithm and the delta-based algorithm. The pivot-based algorithm focuses on reusing computation, while the delta-based algorithm focuses on reducing the number of sampled permutations. In the case of deleting data points, we defined and proposed the YN-NN algorithm of polynomial complexity achieving full accuracy. We then presented the delta-based algorithm, based on Shapley value difference similar to adding data points. Inspired by empirical observations, we proposed similarity-based heuristic algorithms. Experiments on Iris and Adult datasets verified the efficiency and effectiveness of our proposed algorithms. When the datasets are from different distribution, we expect that the KNN algorithm may not be very effective given its assumption of the original data points not change when new data points are added. However, we expect other algorithms to work similarly. We will explore this case in future work.

IX. Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported in part by NSFC grants (62102352), the National Key R&D Program of China (2021YFB3101100, 2021YFB3101102), NSF grants (CNS-2124104, CNS-2125530), and NIH grants (R01ES033241, R01LM013712, UL1TR002378).

References

[1] A. Agarwal, M. A. Dahleh, and T. Sarkar. A marketplace for data: An algorithmic solution. In A. Karlin, N. Immorlica, and R. Johari, editors, *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 701–726. ACM, 2019.

[2] S. Agarwal, S. Dutta, and A. Bhattacharya. Chisel: Graph similarity search using chi-squared statistics in large probabilistic graphs. *Proc. VLDB Endow.*, 13(10):1654–1668, 2020.

[3] J. Castro, D. Gómez, and J. Tejada. Polynomial calculation of the shapley value based on sampling. *Computers & OR*, 36(5):1726–1730, 2009.

[4] S. Chawla, S. Deep, P. Koutrisw, and Y. Teng. Revenue maximization for query pricing. *Proceedings of the VLDB Endowment*, 13(1):1–14, 2019.

[5] L. Chen, P. Koutris, and A. Kumar. Towards model-based pricing for machine learning in a data marketplace. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *SIGMOD*, pages 1535–1552. ACM, 2019.

[6] L. Chen, H. Wang, L. Chen, P. Koutris, and A. Kumar. Demonstration of nimbus: Model-based pricing for machine learning in a data marketplace. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1885–1888. ACM, 2019.

[7] S. B. Cohen, E. Ruppin, and G. Dror. Feature selection based on the shapley value. In L. P. Kaelbling and A. Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 665–670. Professional Book Center, 2005.

[8] W. J. Conover. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

[9] R. D. Cook. Detection of influential observation in linear regression. *Technometrics*, 42(1):65–68, 2000.

[10] S. Deep and P. Koutris. QIRANA: A framework for scalable query pricing. In S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 699–713. ACM, 2017.

[11] S. Deep, P. Koutris, and Y. Bidasaria. QIRANA demonstration: Real time scalable query pricing. *Proc. VLDB Endow.*, 10(12):1949–1952, 2017.

[12] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19(2):257–266, 1994.

[13] D. Deutch, N. Frost, B. Kimelfeld, and M. Monet. Computing the shapley value of facts in query answering. In *SIGMOD Conference 2022*, 2022.

[14] D. Dua and C. Graff. UCI machine learning repository, 2017.

[15] E. Farchi, R. Narayanam, and L. Nagalapatti. Ranking data slices for ML model validation: A shapley value approach. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1937–1942. IEEE, 2021.

[16] R. C. Fernandez, P. Subramaniam, and M. J. Franklin. Data market platforms: Trading data assets to solve data problems. *Proc. VLDB Endow.*, 13(11):1933–1947, 2020.

[17] V. Fragnelli, I. García-Jurado, H. Norde, F. Patrone, and S. Tijs. How to share railways infrastructure costs? In *Game practice: contributions from applied game theory*, pages 91–101. Springer, 2000.

[18] A. Ghorbani, M. P. Kim, and J. Zou. A distributional framework for data valuation. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3535–3544. PMLR, 2020.

[19] A. Ghorbani and J. Y. Zou. Data shapley: Equitable valuation of data for machine learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2242–2251. PMLR, 2019.

[20] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 8–21. IEEE Computer Society, 1978.

[21] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gurel, B. Li, C. Zhang, C. Spanos, and D. Song. Efficient task-specific data valuation for nearest neighbor algorithms. *Proceedings of the VLDB Endowment*, 12(11):1610–1623, 2019.

[22] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gürel, B. Li, C. Zhang, C. J. Spanos, and D. Song. Efficient task-specific data valuation for nearest neighbor algorithms. *Proc. VLDB Endow.*, 12(11):1610–1623, 2019.

[23] 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*, pages 1167–1176. PMLR, 2019.

[24] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *PODS*, pages 167–178. ACM, 2012.

[25] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Toward practical query pricing with querymarket. In *SIGMOD*, pages 613–624. ACM, 2013.

[26] A. B. Kurtulmus and K. Daniel. Trustless machine learning contracts; evaluating and exchanging machine learning models on the ethereum blockchain. *CoRR*,

abs/1802.10185, 2018.

[27] Y. Kwon, M. A. Rivas, and J. Zou. Efficient computation and analysis of distributional shapley values. *CoRR*, abs/2007.01357, 2020.

[28] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. In W. Tan, G. Guerrini, B. Catania, and A. Gounaris, editors, *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 33–44. ACM, 2013.

[29] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. *ACM Trans. Database Syst.*, 39(4):34:1–34:28, 2014.

[30] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. *Commun. ACM*, 60(12):79–86, 2017.

[31] Q. Lin, J. Zhang, J. Liu, K. Ren, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun. Demonstration of dealer: An end-to-end model marketplace with differential privacy. *Proc. VLDB Endow.*, 14(12):2747–2750, 2021.

[32] R. Lindelauf, H. Hamers, and B. Husslage. Cooperative game theoretic centrality analysis of terrorist networks: The cases of jemaah islamiyah and al qaeda. *Eur. J. Oper. Res.*, 229(1):230–238, 2013.

[33] J. Liu, J. Lou, J. Liu, L. Xiong, J. Pei, and J. Sun. Dealer: An end-to-end model marketplace with differential privacy. *Proc. VLDB Endow.*, 14(6):957–969, 2021.

[34] J. Liu, J. Yang, L. Xiong, J. Pei, and J. Luo. Skyline diagram: Finding the voronoi counterpart for skyline queries. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 653–664. IEEE Computer Society, 2018.

[35] S. Maleki, L. Tran-Thanh, G. Hines, T. Rahwan, and A. Rogers. Bounding the estimation error of sampling-based shapley value approximation with/without stratifying. *CoRR*, abs/1306.4265, 2013.

[36] T. P. Michalak, T. Rahwan, P. L. Szczepanski, O. Skibski, R. Narayanam, N. R. Jennings, and M. J. Wooldridge. Computational analysis of connectivity games with applications to the investigation of terrorist networks. In F. Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 293–301. IJCAI/AAAI, 2013.

[37] J. Pei. A survey on data pricing: from economics to data science. *IEEE Trans. Knowl. Data Eng.*, 2021.

[38] L. S. Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[39] T. Song, Y. Tong, and S. Wei. Profit allocation for federated learning. In C. Baru, J. Huan, L. Khan, X. Hu, R. Ak, Y. Tian, R. S. Barga, C. Zaniolo, K. Lee, and Y. F. Ye, editors, *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*, pages 2577–2586. IEEE, 2019.

[40] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye. Dynamic pricing in spatial crowdsourcing: A matching-based approach. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 773–788. ACM, 2018.

[41] P. Upadhyaya, M. Balazinska, and D. Suciu. Price-optimal querying with data apis. *Proc. VLDB Endow.*, 9(14):1695–1706, Oct. 2016.

[42] J. Zhang, Q. Sun, J. Liu, L. Xiong, J. Pei, and K. Ren. Efficient sampling approaches to shapley value approximation. In *SIGMOD*. ACM, 2023.