This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

Ultra Data-Oriented Parallel Fractional Hot-Deck Imputation with Efficient Linearized Variance Estimation

Yicheng Yang, Yonghyun Kwon, Jae Kwang Kim, and In Ho Cho*

Abstract—Parallel fractional hot-deck imputation (P-FHDI [1]) is a general-purpose, assumption-free tool for handling item nonresponse in big incomplete data by combining the theory of FHDI and parallel computing. FHDI cures multivariate missing data by filling each missing unit with multiple observed values (thus, hot-deck) without resorting to distributional assumptions. P-FHDI can tackle big incomplete data with millions of instances (big-n) or $10,000$ variables (big-p). However, handling ultra incomplete data (i.e., concurrently big-n and big-p) with tremendous instances and high dimensionality has posed problems to P-FHDI due to excessive memory requirement and execution time. To tackle the aforementioned challenges, we propose the ultra data-oriented P-FHDI (named UP-FHDI) capable of curing ultra incomplete data. In addition to the parallel Jackknife method, this paper enables a computationally efficient ultra data-oriented variance estimation using parallel linearization techniques. Results confirm that UP-FHDI can tackle an ultra dataset with one million instances and $10,000$ variables. This paper illustrates the special parallel algorithms of UP-FHDI and confirms its positive impact on the subsequent deep learning performance.

Index Terms—Ultra data-oriented parallel fractional hot-deck imputation, ultra incomplete data, ultrahigh dimensional missing data curing, parallel linearized variance estimation, two-staged feature selection, deep learning.

1 Introduction

Incomplete data commonly occurs in nearly all scientific and engineering domains, which may result in biased estimation of parameters and exacerbate subsequent statistical analyses [2]. Inadequate handling of missing data may lead to incorrect statistical inference and subsequent machine learning (ML) [3]. A popular approach, known as listwise deletion [4], is to omit instances with missing values from analysis, but it may seriously bias sample statistics if the data does not follow the assumption of missing completely at random (MCAR). Pairwise deletion proposed by [5] is found deficient for severe missingness. Naive imputation (i.e., replace missing values with the mean of observed values) is popular [6], but it may lead to inconsistent bias when there is considerable missingness inequality across variables.

There are two branches of imputation theory that replace a missing value with statistically plausible values [7]: single imputation and repeated imputation. The single imputation uses expectation-maximization (EM) to replace each missing item with a predicted value estimated by the maximum likelihood methods [8]. However, it may require a considerable convergence time when data missingness is severe. Single imputation methods tend to underestimate the standard errors.

- *Y. Yang and I. Cho are with Department of Civil Engineering, Iowa State University, Ames, IA, 50011. {yicheng, icho}@iastate.edu. *: corresponding author, M. IEEE*
- *J.K. Kim and Y. Kwon are with Department of Statistics, Iowa State University, Ames, IA, 50011. E-mail: {jkim, yhkwon}@iastate.edu*

Manuscript received October 30, 2019; revised October 30, 2019.

Repeated imputation includes two popular methods: multiple imputation and fractional imputation. Multiple imputation proposed by Rubin [9] overcomes the downside of single imputation by replacing each missing value with several plausible values representing the distribution of all possible values. The R package $mice$ implements multiple imputation using chained equations to handle multivariate missing data such that each variable has its own imputation model [10]. However, the so-called "congeniality" and "self-efficiency" conditions are required for the validity of multiple imputation which may pose challenges in practice [11], [12]. Fractional imputation, proposed by [13] and extensively discussed in [14], [15], [16], creates a complete dataset with fractional weights after imputation. Departing from the fractional imputation, [17] proposed fractional hot-deck imputation, a non-parametric imputation method using two-phase sampling for stratification. Some of the authors of this paper developed the R package $FHDI$ [18] to perform fractional hot-deck imputation or fully-efficient fractional imputation (FEFI) to cure general multivariate incomplete datasets without prior distributional assumptions.

Recently, ML-based imputation methods have been gradually emerging: e.g., generative adversarial networks (GAN) for missing data [19], [20]. K-Harmonic mean imputation [21], sequential regression multivariate imputation [22], Fuzzy C-Means imputation [23], and predictive mean matching [24] are also noteworthy. These existing imputation methods often require expert-level distributional assumptions that are difficult for general researchers. Also, computational limitations pose fundamental challenges to curing big incomplete data.

This paper follows the conventions in [1] to divide

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

different types of big incomplete data according to their configurations – "big-n" data with many instances, "big-p" data with high dimensionality, and "ultra" data with concurrently many instances and high-dimensionality. None of the aforementioned existing imputation methods are adequate to cure these three categories of big incomplete data. As the first attempt to handle big-n data with millions of instances or big-p data with $10,000$ variables, some authors of this paper developed an initial version of parallel-FHDI (P-FHDI) by leveraging high-performance computing (HPC) technologies and adding big data-oriented algorithms to FHDI [1]. Yet, "ultra" data was beyond the scope of the previous P-FHDI.

This paper develops ultra (concurrently big-n and big-p) data-oriented parallel FHDI (denoted as UP-FHDI) with specialized theoretical and computational advancements. UP-FHDI inherits the essence of FHDI theory, i.e., imputation cells are created to match donors and recipients, and the observed values of each donor are jointly imputed to fill missing items, generating a single completed dataset at the end.

The major contributions of this paper compared to the previous works are as follows: (1) UP-FHDI supports inter-processor communication and IO communication with the hard drive necessary for ultra data storage, while the previous P-FHDI allows only memory usage and no external data storage; (2) UP-FHDI adopts the parallel k-nearest neighbors (KNN) method for handling deficient donors (i.e., when a missing cell has less than two possible donors), while P-FHDI uses the cell collapsing scheme that causes substantial computing time; (3) UP-FHDI enables a fast and efficient variance estimation by developing parallel linearization techniques, while P-FHDI uses the parallel Jackknife method that is not feasible for ultra data; (4) This paper evaluates the performance of UP-FHDI against baseline imputation methods using large real-world and ultra synthetic datasets and affirms that UP-FHDI is adequate to tackle a wide spectrum of missing data; (5) This paper proposes a two-staged feature selection method using parallel mutual information and the graphical lasso, which facilitates the investigation of the impact of UP-FHDI on the subsequent deep learning performance with ultrahigh dimensional data.

The rest of this paper is organized as follows: we review the related work in Section 2. Section 3 recaps the backbone theories of the four stages of UP-FHDI. Section 4 presents the adopted parallel file system and expounds upon the primary parallel algorithms of UP-FHDI. Section 5 validates UP-FHDI by Monte Carlo simulations and comparative studies against baseline imputation methods. Section 6 evaluates scaling performance and provides cost models. Section 7 introduces a two-staged feature selection method and investigates the impact of UP-FHDI on subsequent deep learning performance. Finally, we present the future works and concluding remarks in Sections 8 and 9. Comprehensive examples in APPENDIX I illustrate how to use UP-FHDI.

2 Related work

This section reviews state-of-the-art parallel computing-based imputation methods. [25] developed a parallel imputation algorithm with cloud-based tensor decomposition to impute missing epigenomics experiments. Package $MaCH$ developed the genotype imputation to infer the missing genotypes in genetic studies. However, it considers all observed genotypes when imputing each missing genotype, leading to a quadratic execution time increase, for which [26] proposed a parallel $MaCH$-based imputation using GPU implementations. Yet, the above parallel imputation methods are restrictive in bioinformatics, and the favorable scaling performance is not broadly confirmed. Incompleteness often occurs in big enterprise data, [27] proposed a parallel imputation framework to cure enterprise registration data, but it is tailored for big geo-referenced text data processing, exhibiting limited scaling (3.5 times at maximum). Other related works include R package $missForst$ [28] for mixed-type data and parallel Bayesian Markov chain Monte Carlo (MCMC) imputation [29] for missing data in epidemiology. $missForst$ adopted the random-forest imputation method and could run in parallel when the computation of random forests is time-consuming. [29] first adopted parallel MCMC for Bayesian imputation on the disk-based shared memory, which parallelized MCMC iterations over available nodes by a simple divide-and-conquer strategy, resulting in promising scalability. However, the parallel MCMC requires highly customized, setting-specific, and parallelized software. All of these parallel imputation methods are restricted to specific disciplines and not suitable for general ultra incomplete data in broad science and engineering domains, which is to be tackled by the present UP-FHDI.

3 Key equations for UP-FHDI

For a concise description of UP-FHDI, we introduce the following basic setup. Assume a finite population $\mathbf{U}$ with p-dimensional continuous variables $\mathbf{y} = \{y_1, y_2, \ldots, y_p\}$, and y_{il} represents the ith realization of the lth variable where $i \in \{1, 2, \ldots, N\}$ and $l \in \{1, 2, \ldots, p\}$. A response indicator δ_{il} takes the value of 1 if y_{il} is observed and $\delta_{il} = 0$ otherwise. Given the number of categories K, one can discretize continuous variables $\mathbf{y}$ to discrete variables $\mathbf{z}$, the so-called "imputation cells," such that $\mathbf{z}$ take values within categories $\{1, 2, \cdots, K\}$ for each variable. The y-values within the same value of z are relatively homogeneous. One can decompose an instance $\boldsymbol{y}_i = \{\boldsymbol{y}_{i,obs}, \boldsymbol{y}_{i,mis}\}$ as the observed and missing parts, respectively. Similarly, the corresponding z-vector is decomposed as $\boldsymbol{z}_i = \{\boldsymbol{z}_{i,obs}, \boldsymbol{z}_{i,mis}\}$.

Let $\boldsymbol{A}$ be the index set of the sample of size n selected by a probability sampling mechanism. Consider $\boldsymbol{A}_M$ as an index set of missing units such that $\boldsymbol{A}_M = \{i \in \boldsymbol{A}; \prod_{l=1}^{p} \delta_{il} = 0\}$; alternatively an index set with observed units is $\boldsymbol{A}_R = \{i \in \boldsymbol{A}; \prod_{l=1}^{p} \delta_{il} = 1\}$ such that $\boldsymbol{A}_M \cup \boldsymbol{A}_R = \boldsymbol{A}$. Imputation cells $\mathbf{z}$ consist of missing patterns $\mathbf{z}_M = \{\boldsymbol{z}_i \mid i \in \boldsymbol{A}_M\}$ and observed patterns $\mathbf{z}_R = \{\boldsymbol{z}_i \mid i \in \boldsymbol{A}_R\}$. The cross-classification of all variables forms imputation cells $\mathbf{z}$, and we assume a cell mean model on the cells determined by $\mathbf{z}$. Considering a finite mixture model under missing at random (MAR), the conditional distribution of $f(\boldsymbol{y}_{i,mis} \mid \boldsymbol{y}_{i,obs})$ can be approximated (See detailed theory

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

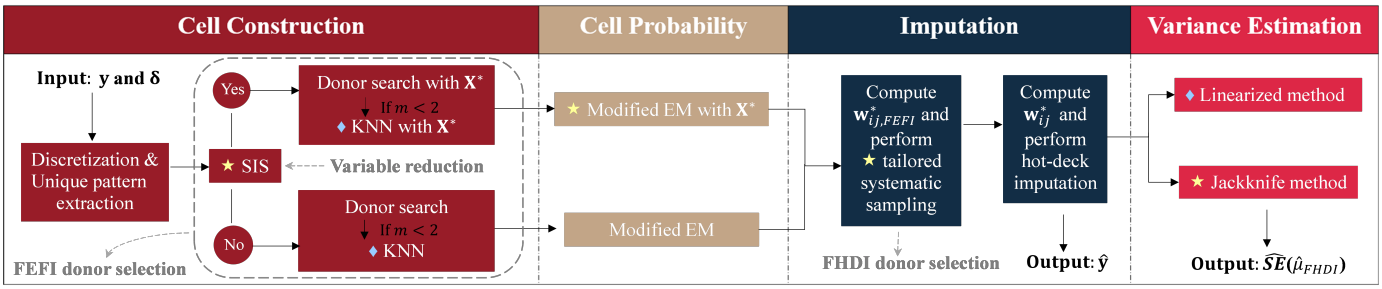


Fig. 1: Key four stages of UP-FHDI. The new theoretical advancements for UP-FHDI are marked with diamonds. The incremental updates of UP-FHDI based on P-FHDI are marked with stars. Note that $\mathbf{X}^*$ is the set of selected variables for all recipients and m is the minimum number of matched donors for all recipients.

in [1]). Fig. 1 gives audiences a comprehensive guide to UP-FHDI theory. We will follow this diagram to present key equations of UP-FHDI stages as follows.

3.1 Cell construction and variable reduction

The mathematical symbol '$\bigtriangledown$' proposed in [1] denotes a loop that repeats a sequence of the same operation $S(x)$ with discrete input augments. Consequently, the operations $\bigtriangledown_{i=a}^{b} S(x_i)$ enumerate a sequence of $S(x_i)$ in a vector format. The operations $\bigtriangledown_{i=a}^{b} \bigtriangledown_{j=c}^{d} S(x_{ij})$ enumerate a sequence of $S(x_{ij})$ in a matrix format. This paper continues to use these notations to facilitate the description of UP-FHDI. [17] proposed the discretization method to pre-determine imputation cells $\mathbf{z}$ using the estimated sample quantiles. Considering $\mathbf{z}_M$ and $\mathbf{z}_R$, we can obtain index sets of unique missing patterns $\tilde{\boldsymbol{A}}_M = \{i, j \in \boldsymbol{A}_M \mid \forall i \neq j,\ \boldsymbol{z}_i \neq \boldsymbol{z}_j\}$ of size $\tilde{n}_M$ and unique observed patterns $\tilde{\boldsymbol{A}}_R = \{i, j \in \boldsymbol{A}_R \mid \forall i \neq j,\ \boldsymbol{z}_i \neq \boldsymbol{z}_j\}$ of size $\tilde{n}_R$, respectively. Sequentially, we can further extract the unique missing patterns $\tilde{\mathbf{z}}_M = \{\boldsymbol{z}_i \mid i \in \tilde{\boldsymbol{A}}_M\}$ and unique observed patterns $\tilde{\mathbf{z}}_R = \{\boldsymbol{z}_i \mid i \in \tilde{\boldsymbol{A}}_R\}$. Let $\boldsymbol{D}_i \in \mathbb{N}^{M_i}$ be the donor indices for the ith recipient $\boldsymbol{z}_i = \{\boldsymbol{z}_{i,obs}, \boldsymbol{z}_{i,mis}\}$ where $\boldsymbol{D}_i = \{j \in \tilde{\boldsymbol{A}}_R \mid \boldsymbol{z}_{j,obs} = \boldsymbol{z}_{i,obs}\}$ with a size of M_i. The entity M_i is the number of donors for the ith recipient. We can perform an exhausting search over $\tilde{\mathbf{z}}_R$ to iteratively obtain index sets of donors for all recipients $\mathbf{L} = \{\boldsymbol{D}_i \mid i \in \tilde{\boldsymbol{A}}_M\}$. Meanwhile, we derive a set of the number of total donors for all recipients $\boldsymbol{M} = \{M_i \mid i \in \tilde{\boldsymbol{A}}_M\}$. [1] gave full details on unique pattern extraction and donor search, this section needs no further elaboration.

UP-FHDI must overcome the curse of dimensionality. A huge literature was accumulated on the problem of variable reduction (e.g., [30], [31], [32], [33]). Noteworthy, [34] proposed the sure independence screening (SIS) for ultrahigh dimensional variable selection. Inspired by the screening step of SIS, [1] came up with a correlation-based SIS to filter out noise variables with weak correlations to responses (i.e., missing variables). Suppose we select v variables for a recipient such that $v \ll p$. Let $\mathbf{X} = \{X_1, \ldots, X_q\}$ be always observed and $\mathbf{Y} = \{Y_1, \ldots, Y_w\}$ be subject to missingness such that $p = q + w$. Consider $\boldsymbol{r}_k = (r_k^1, r_k^2, \ldots, r_k^q)$ as a vector of sample correlations of $\mathbf{X}$ given Y_k. This paper proposes a complement to the existing correlation-based SIS as follows:

(1) Compute correlation vectors $\boldsymbol{r}_k$ where $k \in \{1, \ldots, w\}$.
(2) Extract $\boldsymbol{r}^* = (r_1^*, r_2^*, \ldots, r_q^*)$, where $r_i^* = \max(\mathrm{r}_1^i, r_2^i, \ldots, r_k^i)$.
(3) Define sub-covariate set $\mathcal{M}$ for imputing $\mathbf{Y}$ such that

$$\mathcal{M} = \{1 \leqslant i \leqslant q; \quad \mid r_i^* \mid \text{ is among top of the larget } v, \text{ where } r_i^* \in \boldsymbol{r}^*\}. \tag{1}$$

By contrast, the existing SIS in [1] derived $\mathcal{M} = \cap_{k=1}^{w} \mathcal{M}_k$ or $\mathcal{M} = \cup_{k=1}^{w} \mathcal{M}_k$, where $\mathcal{M}_k$ is the sub-covariate set with top correlations for imputing Y_k.

Imputation cells $\mathbf{z}$ after discretization can not always guarantee at least two donors for every recipient to capture the variability from imputation. Denote the minimum entity of $\boldsymbol{M}$ be m. [1] adopted the cell collapsing to produce more donors and stop only if $m \geqslant 2$. The non-parallelizable cell collapsing method is computationally intractable in practice. As a remedy, this paper applies the KNN method [35] to efficiently determine deficient donors. Let $\boldsymbol{e} \in \mathbb{R}^{\tilde{n}_R}$ be the Euclidean distance (ED) between a recipient $\boldsymbol{z}_i$ and $\tilde{\mathbf{z}}_R$. Let $\boldsymbol{w}$ be the observed covariate set for $\boldsymbol{z}_i$ if SIS is not used. Otherwise, $\boldsymbol{w}$ is set to be the sub-covariate set $\mathcal{M}$ derived by SIS for $\boldsymbol{z}_i$. Suppose $\boldsymbol{z}_i$ has less than two donors such that $M_i < 2$, the KNN algorithm consists of the following steps:

(1) Compute $\boldsymbol{e}$ by

$$\boldsymbol{e} = \bigtriangledown_{j=1}^{\tilde{n}_R} \sqrt{\sum_{l \in \boldsymbol{w}} \left(\frac{z_{il}}{k_l} - \frac{\tilde{z}_{R(j,l)}}{k_l} \right)^2}. \tag{2}$$

where $\tilde{z}_{R(j,l)} \in \tilde{\mathbf{z}}_R$ and k_l is the number of categories of $\boldsymbol{z}_l$.
(2) Suppose e_{t_1} is the minimum entity of $\boldsymbol{e}$. Add t_1 to $\boldsymbol{D}_i$ and update M_i. If $M_i \geqslant 2$, stop.
(3) Suppose e_{t_2} is the second minimum entity of $\boldsymbol{e}$. Add t_2 to $\boldsymbol{D}_i$ and update M_i.

We will iteratively apply the KNN method to every unqualified recipient until $m \geqslant 2$.

3.2 Estimation of cell probability with selected variables

A modified EM algorithm by weighting was proposed by [36] and extensively used in [1], [17] to estimate the conditional probability. This section extends this method with selected variables. Let $\mathbf{X}^*$ be the selected covariates

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

in the sub-covariate set $\mathcal{M}$ derived by SIS such that $\mathbf{X}^* \subset \mathbf{X} = \{X_1, \ldots, X_q\}$. Consider $\mathbf{Y} = \{Y_1, \ldots, Y_w\}$ as a w-dimensional categorical random vector with support $\boldsymbol{C}$. We develop an EM algorithm for estimating the joint probability for $P(\mathbf{Y} \mid \mathbf{X}^*)$. Let $\pi_c(\boldsymbol{x}^*) = P(\mathbf{Y} = \boldsymbol{c} \mid \boldsymbol{x}^*)$ be the conditional probability of $\mathbf{Y} = \boldsymbol{c}$ given $\mathbf{X}^* = \boldsymbol{x}^*$. Note that

$$\sum_{c \in \boldsymbol{C}} \pi_c(\boldsymbol{x}^*) = 1. \tag{3}$$

The E-step is to compute the conditional probability

$$\begin{aligned} & p_i^{(t)}(\boldsymbol{c}) \equiv P^{(t)}(\boldsymbol{y}_i = \boldsymbol{c} \mid \boldsymbol{x}_i^*, \boldsymbol{y}_{i,obs}) \\ & = I(\boldsymbol{y}_{i,obs} = \boldsymbol{c}_{obs(i)}) \cdot P^{(t)}(\boldsymbol{y}_{i,mis} = \boldsymbol{c}_{mis(i)} \mid \boldsymbol{x}_i^*, \boldsymbol{y}_{i,obs}), \end{aligned} \tag{4}$$

where $\{\boldsymbol{c}_{obs(i)}, \boldsymbol{c}_{mis(i)}\}$ is a partition of $\boldsymbol{c}$ based on the missing pattern in unit i. And $P^{(t)}(\boldsymbol{y}_{i,mis} = \boldsymbol{c}_{mis(i)} \mid \boldsymbol{x}_i^*, \boldsymbol{y}_{i,obs})$ is

$$P^{(t)} = \begin{cases} \frac{\pi_c^{(t)}(\boldsymbol{x}_i^*)}{\sum_{c \in \boldsymbol{C}_i} \boldsymbol{\pi}_c^{(t)}(\boldsymbol{x}_i^*)} & \text{if } \boldsymbol{y}_{i,obs} = \boldsymbol{c}_{obs(i)} \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

and $\boldsymbol{C}_i = \{\boldsymbol{c} \in \boldsymbol{C}; \boldsymbol{y}_{i,obs} = \boldsymbol{c}_{obs(i)}\}$ is the subset of $\boldsymbol{C}$ whose $\boldsymbol{c}_{obs(i)}$ components are equal to $y_{i,obs}$. The M-step is to update the conditional probability as

$$\pi_c^{(t+1)}(\boldsymbol{x}^*) = \frac{\sum_{i=1}^n I(\boldsymbol{x}_i^* = \boldsymbol{x}^*, y_{i,obs} = c_{obs(i)}) p_i^{(t)}(\boldsymbol{c})}{\sum_{c \in \boldsymbol{C}} \sum_{i=1}^n I(\boldsymbol{x}_i^* = \boldsymbol{x}^*, y_{i,obs} = c_{obs(i)}) p_i^{(t)}(\boldsymbol{c})}. \tag{6}$$

3.3 Imputation

For simplicity in description, consider two discrete random variables (z_1, z_2) with the same support of (z_{obs}, z_{mis}). The key equation for developing fractional hot deck imputation is to approximate the conditional distribution of $f(y_{mis} \mid y_{obs})$ by

$$\begin{aligned} & f(y_{mis} \mid y_{obs}) \\ & = \sum_{z_2} \sum_{z_1} p(z_1, z_2 \mid y_{obs}) f(y_{mis} \mid z_1, z_2) \\ & = \sum_{z_2} \sum_{z_1} p(z_1 \mid y_{obs}) p(z_2 \mid z_1) f(y_{mis} \mid z_1, z_2) \\ & = p(z_1 = z_{obs} \mid y_{obs}) p(z_{mis} \mid z_{obs}) f(y_{mis} \mid z_1 = z_{obs}, z_2 = z_{mis}) \\ & = p(z_1 = z_{obs} \mid y_{obs}) p(z_{mis} \mid z_{obs}) f(y_{mis} \mid z_2 = z_{mis}). \end{aligned} \tag{7}$$

where the last equality is derived under the conditional independence between y_{mis} and z_{obs} given z_{mis}. The first component is equal to $I(z_1 = z_{obs})$ if z_{obs} is a sufficient summary of y_{obs}. The second component is computed by the above EM algorithm. The third component is computed from the hot-deck imputation within $\mathbf{z}$.

For concise explanations, let $\boldsymbol{A}$ be partitioned into G groups such that $\boldsymbol{A} = \boldsymbol{A}_1 \cup \ldots, \boldsymbol{A}_G$. The group $\boldsymbol{A}_g$ can be sub-partitioned into $\boldsymbol{A}_{R_g} = \{j \in \boldsymbol{A}_g; \delta_j = 1\}$ and $\boldsymbol{A}_{M_g} = \{j \in \boldsymbol{A}_g; \delta_j = 0\}$. Let n_{R_g} and n_{M_g} be size of $\boldsymbol{A}_{R_g}$ and $\boldsymbol{A}_{M_g}$, respectively. We can decompose a donor for the ith recipient $(\boldsymbol{z}_{i,obs}, \boldsymbol{z}_{i,mis})$ as $(\boldsymbol{z}_{j,obs}, \boldsymbol{z}_{j,mis}^*)$ where $\boldsymbol{z}_{j,obs} = \boldsymbol{z}_{i,obs}$ and $\boldsymbol{z}_{j,mis}^*$ is a possible imputed value for $\boldsymbol{z}_{i,mis}$. Let $w_{ij,FEFI}^*$ be the fractional weights of the jth FEFI donor for the ith recipient. The FEFI donors refer to all possible donors and $w_{ij,FEFI}^*$ is expressed by

$$w_{ij,FEFI}^* = \hat{\pi}_{\boldsymbol{z}_{j,mis}^* | \boldsymbol{z}_{j,obs}} \frac{w_j}{\sum_{l \in A} w_l a_l}. \tag{8}$$

The term $a_l = 1$ if $(\boldsymbol{z}_{l,obs}, \boldsymbol{z}_{l,mis}^*) = (\boldsymbol{z}_{j,obs}, \boldsymbol{z}_{j,mis}^*)$. Otherwise, $a_l = 0$. Note that $\sum_{j=1}^{n_{R_g}} w_{ij,FEFI}^* = 1$. The term $\hat{\pi}_{\boldsymbol{z}_{j,mis}^* | \boldsymbol{z}_{j,obs}}$ is the conditional probability of $\boldsymbol{z}_{j,mis}^*$ given $\boldsymbol{z}_{j,obs}$ computed by

$$\hat{\pi}_{\boldsymbol{z}_{j,mis}^* | \boldsymbol{z}_{obs}} = \frac{P(\boldsymbol{z}_{j,mis}^*, \boldsymbol{z}_{obs})}{P(\boldsymbol{z}_{obs})} = \frac{P(\boldsymbol{z}_{j,mis}^*, \boldsymbol{z}_{obs})}{\sum_{j=1}^{n_{R_g}} P(\boldsymbol{z}_{j,mis}^*, \boldsymbol{z}_{obs})}. \tag{9}$$

where $P(\boldsymbol{z}_{j,mis}^*, \boldsymbol{z}_{obs})$ is the joint probability derived after the convergence of the EM algorithm in the preceding subsection. Note that $\sum_{j=1}^{n_{R_g}} \hat{\pi}_{\boldsymbol{z}_{j,mis}^* | \boldsymbol{z}_{j,obs}} = 1$. The tailored systematic sampling [18] was designated to efficiently select M FHDI donors among FEFI donors. UP-FHDI inherits this method with modifications. Originally, the first step of the tailored systematic sampling scheme sorts all FEFI donors in Mahalanobis distance by the half-ascending half-descending order. However, computation of Mahalanobis distance is intractable when the dimension space is ultra-high. As a remedy, we substitute this step with a random shuffle and confirm that this modification affects the mean estimates in a negligible manner. Let w_{ij}^* be the fractional weights of the jth FHDI donor for the ith recipient. The w_{ij}^* is given by

$$w_{ij}^* = \begin{cases} \frac{1}{M} & \text{if } n_{R_g} > M \\ w_{ij,FEFI}^* & \text{if } n_{R_g} \leqslant M, \end{cases} \tag{10}$$

The FHDI estimator of y_l can be expressed by

$$\hat{Y}_{l,FHDI} = \sum_{i \in A} w_i \{\delta_{il} y_{il} + (1 - \delta_{il}) \sum_{j=1}^{M} w_{ij}^* y_{il}^{*(j)}\}. \tag{11}$$

where $y_{il}^{*(j)}$ is the jth imputed value of y_{il}.

3.4 Fast and efficient variance estimation for ultra data

Popular variance estimation methods include balanced repeated replication [37], bootstrap [38], and Jackknife methods [39]. The Jackknife variance estimation had been successfully implemented into P-FHDI and proved to be effective [1]. We modified the parallel Jackknife method in this study for better efficiency. Yet, it is not applicable for ultra data owing to the computational bottleneck. To overcome this challenge, we develop a linearized variance estimation technique based on one of the author's dedicated prior works [40]. Consider variance estimation of a deterministic imputation. Assume the original sample is decomposed into G disjoint groups. The sample observations follow the cell mean model:

$$y_i \mid i \in A_g \overset{i.i.d}{\sim} (\mu_g, \sigma_g^2). \tag{12}$$

where A_g is the index set of group g. Let n_g and r_g be the number of elements and the number of observed elements in group g, respectively. Assume the response mechanism is MAR and the parameter of interest is $\theta = E(Y)$. We will use a deterministic imputation with $\hat{\eta} = (\hat{\mu}_1, \ldots, \hat{\mu}_G)$, where $\hat{\mu}_g = r_g^{-1} \sum_{i \in A_g} \delta_i y_i$ is the gth cell mean of y among respondents. The imputed estimator of θ will be

$$\hat{\theta}_I = \frac{1}{n} \sum_{g=1}^{G} \sum_{i \in A_g} \{\delta_i y_i + (1 - \delta_i) \hat{\mu}_g\} = \frac{1}{n} \sum_{g=1}^{G} n_g \hat{\mu}_g, \tag{13}$$

Writing $e_{ig} = y_i - \mu_g$ for $i \in A_g$, we can express

$$\hat{\theta}_I = \frac{1}{n} \sum_{g=1}^{G} \left\{ n_g \mu_g + \frac{n_g}{r_g} \sum_{i \in A_g} \delta_i e_{ig} \right\},$$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

Using $E(e_{ig}) = 0$, the uncertainty associated with n_g/r_g is asymptotically negligible. Thus, the imputed estimator $\hat{\theta}_I$ can be expressed by

$$\hat{\theta}_I \cong \frac{1}{n}\sum_{g=1}^{G}\sum_{i\in A_g}\left\{\mu_g + \frac{1}{p_g}\delta_i(y_i - \mu_g)\right\}. \tag{14}$$

where p_g is the probability limit of r_g/n_g. The resulting variance estimator is

$$\hat{V}(\hat{\theta}_I) = \frac{1}{n(n-1)}\sum_{g=1}^{G}\sum_{i\in A_g}(\hat{d}_{gi} - \bar{d}_n)^2, \tag{15}$$

where

$$\hat{d}_{gi} = \hat{\mu}_g + \frac{n_g}{r_g}\delta_i(y_i - \hat{\mu}_g), \tag{16}$$

$$\bar{d}_n = \frac{1}{n}\sum_{g=1}^{G}\sum_{i\in A_g}\hat{d}_{gi}. \tag{17}$$

We can extend the linearization formula for the variance of the mean estimator of FHDI with multivariate missing data, where M imputed values are randomly selected from respondents in the same imputation cell. The FHDI estimator of the mean of $\boldsymbol{y}_l$ is given by

$$\hat{\mu}_{l,FHDI} = \frac{1}{n}\sum_{i=1}^{n}\left\{\delta_{il}y_{il} + (1-\delta_{il})\bar{y}^*_{il}\right\}, \tag{18}$$

where

$$\bar{y}^*_{il} = M^{-1}\sum_{j=1}^{M} y^{*(j)}_{il}. \tag{19}$$

Assume each unit i in the sample belong to one and only one imputation cell. Let $\bar{y}_{gl}$ be the sample mean of $\boldsymbol{y}_l$ among the respondents in cell g such that

$$\bar{y}_{gl} = \frac{1}{r_{gl}}\sum_{i\in\boldsymbol{S}_{gl}}\delta_{il}y_{il}, \tag{20}$$

where $\boldsymbol{S}_{gl} \in \mathbb{N}^{n_{gl}}$ is the index set of the units in imputation cell g. And r_{gl} is the number of observed units (i.e., $\delta_i = 1$) in cell g of $\boldsymbol{y}_l$. The variance estimator of $\hat{\mu}_{l,FHDI}$ can be expressed by

$$\hat{V}(\hat{\mu}_{l,FHDI}) = \frac{1}{n(n-1)}\sum_{i=1}^{n}(\hat{\eta}_{il} - \bar{\eta}_l)^2, \tag{21}$$

where

$$\hat{\eta}_{il} = \delta_{il}\bar{y}_{gl} + (1-\delta_{il})\bar{y}^*_{il} + \delta_{il}\frac{n_{gl}}{r_{gl}}(y_{il} - \bar{y}_{gl}), \tag{22}$$

and $\bar{\eta}_l = n^{-1}\sum_{i=1}^{n}\hat{\eta}_{il}$. The standard error of $\hat{\mu}_{l,FHDI}$ is computed by

$$\widehat{SE}(\hat{\mu}_{l,FHDI}) = \sqrt{\hat{V}(\hat{\mu}_{l,FHDI})}. \tag{23}$$

4 Parallel Algorithms of UP-FHDI

Fig. 2 visualizes the parallel file system tailored for UP-FHDI. This system targets the HPC environment that allows simultaneous access from multiple compute servers. Suppose we have in total Q processors indexed by $0, \ldots, Q-1$. The slave processors indexed by $1, \ldots, Q-1$ perform distributed computations. The master processor (indexed by 0) is responsible for aggregating results from slave processors. The MPI library [41] provides basic routines MPI_Send and MPI_Recv (denoted as MPI_SR for brevity), and MPI_Bcast to support inter-processor communication. The master processor integrates pieces of results together by an operation MPI_Gather (denoted as Ω). The operation Ω_i^j $(i \neq 0)$ integrates results generated by slave processors (indexed by i to j) and stacks them on the master. Notably, the parallel file system supports IO communication between slave processors and the hard drive by cooperative parallel IO in MPI, thereby overcoming out-of-memory problems. We adopt IO routines MPI_File_Read (denoted as MPI_RD) and MPI_File_Write (denoted as MPI_WR) to simultaneously read or write from multiple processes to a single binary file on the hard drive. The MPI parallel IO is superior to non-parallel IO since its operations are collective such that IO is scalable. A single user's intensive and simultaneous IO running on a small number of nodes can overload the metadata server and degrade global file system performance. As a remedy, [42] developed the Optimal Overload IO Protection System (OOOPS) to automatically detect and throttle the IO workload. We apply this tool with UP-FHDI to dynamically adjust IO during the job without interruption. Hereafter, any vector or matrix will be marked with a star symbol '*' on the left-up corner when it is temporarily placed on the hard drive. The development of UP-FHDI is conducted and tested on the local HPC of ISU (Condo [43]) and NSF Cyberinfrastructure's HPC (Stampede2 [44]). The benchmark tests across different HPC environments guarantee the compatibility of UP-FHDI.

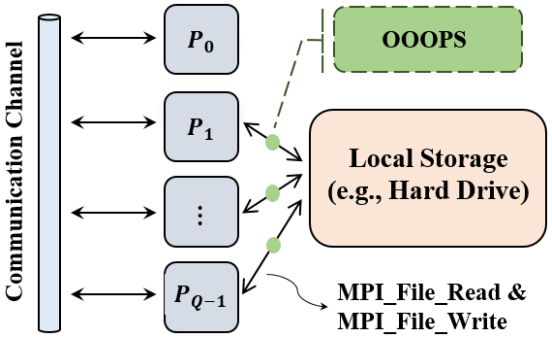


Fig. 2: Scheme of the parallel file system on multiple writers and readers. The IO communication between slave processors and the hard drive is conveyed by MPI_File_Read and MPI_File_Write routines. The OOOPS optimally throttles the IO workload (solid green circles).

4.1 Parallel cell construction

Let v be the number of reduced variables after SIS per each missing pattern. Algorithm 1 recaps the key sub-steps of parallel cell construction with SIS. The function CAT predetermines imputation cells $\mathbf{z}$ and the function ZMAT decomposes $\mathbf{z}$ into unique missing patterns $\tilde{\mathbf{z}}_M$ and unique observed patterns $\tilde{\mathbf{z}}_R$. Sequentially, we pair donors for each recipient by function nDAU. Note that we search donors for each recipient based on v selected variables when SIS is activated by setting $v \neq 0$. Otherwise, donors are determined based on all observed variables of a recipient. If there exists $\boldsymbol{z}_i \in \tilde{\mathbf{z}}_M$ such that $M_i < 2$ (i.e., deficient donor situation), we adopt the KNN function until every recipient has at least two donors. Unless otherwise stated, UniformDistr(.) means a simple uniform task/data distribution scheme. The indices (s and e) represent the

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

beginning and ending of distributed work on processor q, respectively. In lines 1 and 2 of Algorithm 2, we read distributed raw data on hard drive $\mathbf{y}^{(q)}$ to each slave processor. Considering the estimated distribution function of y_k:

$$\hat{F}_k(t) = \frac{\sum_{i\in A}\delta_{ik}w_i I(y_{ik}\leq t)}{\sum_{i\in A}\delta_{ik}w_i}. \tag{24}$$

For every y_k where $k \in \{s,\ldots,e\}$, we construct proportions $\{\hat{q}_{a_1},\ldots,\hat{q}_{a_G}\}$ satisfying $0 < a_1 < \cdots < a_G$ in line 4, the estimated sampling quantile of y_k for a_g is defined as:

$$\hat{q}_{a_g} = min\{t, \hat{F}(t) > a_g\}. \tag{25}$$

From lines 5 to 9, we can discrete y_k to z_k with respect to $\{\hat{q}_{a_1},\ldots,\hat{q}_{a_G}\}$. In lines 11 and 12, write distributed imputation cells $\mathbf{z}^{(q)}$ from slave processors to the hard drive and integrate as ${}^*\mathbf{z}$. In lines 1 and 2 of Algorithm 3, read distributed imputation cells $\mathbf{z}^{(q)}$ to each slave processor. Let $F \in \mathbb{N}$ such that $\mathbf{z} = \mathbf{z}_1 \cup \cdots \cup \mathbf{z}_F$. We can iteratively compare $\mathbf{z}^{(q)}$ with $\mathbf{z}_f$, $f \in \{1,\ldots,F\}$ in lines 3 to 6 to remove duplicates in $\mathbf{z}^{(q)}$, and thus obtain distributed unique patterns $\tilde{\mathbf{z}}^{(q)}$. Hence, $\tilde{\mathbf{z}}^{(i)} \cap \tilde{\mathbf{z}}^{(j)} = \emptyset$ whenever $i \neq j$. Distributed unique observed patterns $\tilde{\mathbf{z}}_R^{(q)}$ and missing patterns $\tilde{\mathbf{z}}_M^{(q)}$ can be partitioned from $\tilde{\mathbf{z}}^{(q)}$ in line 7. In lines 8 and 9, we simultaneously write $\tilde{\mathbf{z}}_R^{(q)}$ and $\tilde{\mathbf{z}}_M^{(q)}$ as ${}^*\tilde{\mathbf{z}}_R$ and ${}^*\tilde{\mathbf{z}}_M$ on the hard drive via the parallel file system.

Algorithm 1 Parallel cell construction

Input: raw data ${}^*\mathbf{y}$
Output: imputation cells ${}^*\mathbf{z}$, unique observed patterns ${}^*\tilde{\mathbf{z}}_R$, unique missing patterns ${}^*\tilde{\mathbf{z}}_M$, donor list $\mathbf{L}$

1: Invoke function CAT(${}^*\mathbf{y}$) $\rightarrow$ ${}^*\mathbf{z}$
2: Invoke function ZMAT(${}^*\mathbf{z}$) $\rightarrow$ ${}^*\tilde{\mathbf{z}}_R, {}^*\tilde{\mathbf{z}}_M$
3: **if** v==0 **then**
4: Invoke function nDAU(${}^*\tilde{\mathbf{z}}_R, {}^*\tilde{\mathbf{z}}_M$) $\rightarrow \boldsymbol{M}, \mathbf{L}$
5: **else**
6: Invoke function RANK(${}^*\mathbf{y}$) $\rightarrow \mathbf{V}$
7: Invoke function nDAU(${}^*\tilde{\mathbf{z}}_R, {}^*\tilde{\mathbf{z}}_M, \mathbf{V}$) $\rightarrow \boldsymbol{M}, \mathbf{L}$
8: **end if**
9: m = min($\boldsymbol{M}$)
10: **if** $m < 2$ **then**
11: Invoke function KNN(${}^*\tilde{\mathbf{z}}_R, {}^*\tilde{\mathbf{z}}_M, \boldsymbol{M}, \mathbf{L}$) $\rightarrow \mathbf{L}$
12: **end if**

Algorithm 2 Parallel function CAT

Input: raw data ${}^*\mathbf{y}$
Output: imputation cells ${}^*\mathbf{z}$

1: UniformDistr(p)$\rightarrow s, e$
2: MPI_RD(${}^*\mathbf{y}^{(q)}$) $\rightarrow \mathbf{y}^{(q)}$
3: **for** $\forall$ k in $s:e$ **do**
4: Construct cumulative proportions $\{\hat{q}_{a_1},\ldots,\hat{q}_{a_G}\}$
5: **for** $\forall$ i in $1:n$ **do**
6: **if** $\hat{q}_{a_g-1} < y_{ik} < \hat{q}_{a_g}$ **then**
7: $z_{ik} = g$
8: **end if**
9: **end for**
10: **end for**
11: MPI_WR($\mathbf{z}^{(q)}$) $\rightarrow {}^*\mathbf{z}^{(q)}$
12: ${}^*\mathbf{z} = \Omega_1^{Q-1}{}^*\mathbf{z}^{(q)}$

Algorithm 3 Parallel function ZMAT

Input: imputation cells ${}^*\mathbf{z}$
Output: unique observed patterns ${}^*\tilde{\mathbf{z}}_R$ and unique missing patterns ${}^*\tilde{\mathbf{z}}_M$

1: UniformDistr(n)$\rightarrow s, e$
2: MPI_RD(${}^*\mathbf{z}^{(q)}$) $\rightarrow \mathbf{z}^{(q)}$
3: **for** $\forall$ f in $1:F$ **do**
4: MPI_RD(${}^*\mathbf{z}_f$) $\rightarrow \mathbf{z}_f$
5: Compare($\mathbf{z}^{(q)}, \mathbf{z}_f$) $\rightarrow \tilde{\mathbf{z}}^{(q)}$
6: **end for**
7: $\tilde{\mathbf{z}}^{(q)} \rightarrow \tilde{\mathbf{z}}_R^{(q)}, \tilde{\mathbf{z}}_M^{(q)}$
8: MPI_WR$\left(\tilde{\mathbf{z}}_R^{(q)}; \tilde{\mathbf{z}}_M^{(q)}\right) \rightarrow {}^*\tilde{\mathbf{z}}_R^{(q)}, {}^*\tilde{\mathbf{z}}_M^{(q)}$
9: ${}^*\tilde{\mathbf{z}}_R = \Omega_1^{Q-1}\left({}^*\tilde{\mathbf{z}}_R^{(q)}\right)$; ${}^*\tilde{\mathbf{z}}_M = \Omega_1^{Q-1}({}^*\tilde{\mathbf{z}}_M^{(q)})$

Let $\boldsymbol{v}_k = \{l \mid 1 \leqslant l \leqslant p; \left|r_k^l\right| \text{ is among the top of largest } t\}$, where r_k^l is a simple correlation computed by observed values of y_k and y_l. Let $\mathbf{V} = \sum_{k=1}^{p}\boldsymbol{v}_k$ be an index set that has top t correlations. Note that t is a user-defined integer where $t > v$. In high dimensional space, the use of t significantly reduces the memory usage and guarantees one can extract v reduced variables from $\mathbf{V}$. In lines 1 and 2 of Algorithm 4, read distributed raw data $\mathbf{y}^{(q)}$ to each slave processor. Consider $F \in \mathbb{N}$ such that $\mathbf{y}_1 \cup \cdots \cup \mathbf{y}_F = \mathbf{y}$. In line 6, derive an index list $\boldsymbol{v}_k$ based on the computed correlations between $y_k^{(q)}$ and $\mathbf{y}_f$. If f equals one, $\mathbf{V}^{(q)}$ can be initialized by including $\boldsymbol{v}_k$ where $k \in \{1,\ldots,p/(Q-1)\}$. Otherwise, we need to update the existing $\boldsymbol{v}_k$ in $\mathbf{V}^{(q)}$ in contrast with the $\boldsymbol{v}_k$ generated in the current iteration. Generally, we always retain the latest index sets in $\mathbf{V}^{(q)}$ after updates within iterations. See the SIS algorithm in Section 3.1 for more explanations. In line 14, we can integrate $\mathbf{V}^{(q)}$ on the master processor and broadcast $\mathbf{V}$ to slaves in line 15.

In lines 1 and 2 of Algorithm 5, we read distributed recipients $\tilde{\mathbf{z}}_M^{(q)}$ to each slave processor. From lines 3 to 15, pair donors for each recipient $\tilde{\boldsymbol{z}}_{M,k}^{(q)} \in \tilde{\mathbf{z}}_M^{(q)}$. Let $F \in \mathbb{N}$ such that $\tilde{\mathbf{z}}_R = \tilde{\mathbf{z}}_{R1} \cup \cdots \cup \tilde{\mathbf{z}}_{RF}$. Likewise, let $\tilde{\boldsymbol{A}}_f$ be an index set of $\tilde{\mathbf{z}}_{Rf}$ such that $\tilde{\boldsymbol{A}}_R = \tilde{\boldsymbol{A}}_1 \cup \ldots \tilde{\boldsymbol{A}}_F$. We iteratively compare $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ with $\tilde{\mathbf{z}}_{Rf}$ from lines 6 to 13 to obtain donor list $\boldsymbol{D}_k$ and the number of donors M_k by

$$\boldsymbol{D}_k = \sum_{t\in\tilde{\boldsymbol{A}}_f} tI(\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}), \tag{26}$$

$$M_k = \sum_{t\in\tilde{\boldsymbol{A}}_f} I(\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}). \tag{27}$$

where $\tilde{\mathbf{z}}_{Rf,t} \in \tilde{\mathbf{z}}_{Rf}$. Note that $t \in \tilde{\boldsymbol{A}}_f$ in Eq. (26) will be mapping to $t \in \tilde{\boldsymbol{A}}_R$ globally before adding to $\boldsymbol{D}_k$. When no variable reduction is applied by setting $v = 0$, $I(\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}) = 1$ if $\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}$ is true concerning all observed variables $\boldsymbol{w}$ of $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ (lines 7 and 8). Otherwise, we need to first obtain an index set of selected variables $\boldsymbol{v}$ for $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ by either of the SIS methods in line 10. Note that the proposed SIS method in Section 3.1 is denoted as global herein. Then $I(\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}) = 1$ if $\tilde{\boldsymbol{z}}_{M,k}^{(q)} = \tilde{\mathbf{z}}_{Rf,t}$ is true concerning only the selected variables $\boldsymbol{v}$ of $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ (lines 11 and 12). In lines

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

16 and 17, we can integrate $\mathbf{L}^{(q)}$ and $\boldsymbol{M}^{(q)}$ on the master processor and broadcast $\mathbf{L}$ and $\boldsymbol{M}$ to slaves (line 18).

Algorithm 4 Parallel function RANK

Input: raw data ${}^*\mathbf{y}$
Output: ranking list $\mathbf{V}$

1: UniformDistr(p)$\longrightarrow s, e$
2: MPI_RD(${}^*\mathbf{y}^{(q)}$) $\longrightarrow \mathbf{y}^{(q)}$
3: **for** $\forall\ f$ in $1:F$ **do**
4: MPI_RD(${}^*\mathbf{y}_f$) $\longrightarrow \mathbf{y}_f$
5: **for** $\forall\ k$ in $1:p/(Q-1)$ **do**
6: Correlation($y_k^{(q)}, \mathbf{y}_f$) $\longrightarrow \boldsymbol{v}_k$
7: **if** f==1 **then**
8: $\mathbf{V}^{(q)}$.add($\boldsymbol{v}_k$)
9: **else**
10: $\mathbf{V}^{(q)}$.update($\boldsymbol{v}_k$)
11: **end if**
12: **end for**
13: **end for**
14: MPI_SR($\mathbf{V}^{(q)}$); $\mathbf{V} = \Omega_1^{Q-1}\mathbf{V}^{(q)}$
15: MPI_Bcast($\mathbf{V}$)

Algorithm 5 Parallel function nDAU

Input: unique observed patterns ${}^*\tilde{\mathbf{z}}_R$ and unique missing patterns ${}^*\tilde{\mathbf{z}}_M$, ranking list $\mathbf{V}$
Output: donor list $\mathbf{L}$, number set of donors $\boldsymbol{M}$

1: UniformDistr($\tilde{n}_M$)$\rightarrow s, e$
2: MPI_RD(${}^*\tilde{\mathbf{z}}_M^{(q)}$) $\rightarrow \tilde{\mathbf{z}}_M^{(q)}$
3: **for** $\forall\ k$ in $1:\tilde{n}_M^{(q)}$ **do**
4: **for** $\forall\ f$ in $1:F$ **do**
5: MPI_RD$\left({}^*\tilde{\mathbf{z}}_{Rf}\right) \rightarrow \tilde{\mathbf{z}}_{Rf}$
6: **if** v==0 **then**
7: Compare($\tilde{\boldsymbol{z}}_{M,k}^{(q)}, \tilde{\mathbf{z}}_{Rf}$) $\rightarrow \mathbf{L}^{(q)}$.add($\boldsymbol{D}_k$)
8: Compare($\tilde{\boldsymbol{z}}_{M,k}^{(q)}, \tilde{\mathbf{z}}_{Rf}$) $\rightarrow \boldsymbol{M}^{(q)}$.add($M_k$)
9: **else**
10: $\boldsymbol{v} = \cap_{k\in\boldsymbol{w}}\boldsymbol{v}_k$ $\|\boldsymbol{v}= \cup_{k\in\boldsymbol{w}}\boldsymbol{v}_k$ $\|\boldsymbol{v}=$ global$_{k\in\boldsymbol{w}}\boldsymbol{v}_k$
11: Compare($\tilde{\boldsymbol{z}}_{M,k}^{(q)}, \tilde{\mathbf{z}}_{Rf}, \boldsymbol{v}$) $\rightarrow \mathbf{L}^{(q)}$.add($\boldsymbol{D}_k$)
12: Compare($\tilde{\boldsymbol{z}}_{M,k}^{(q)}, \tilde{\mathbf{z}}_{Rf}, \boldsymbol{v}$) $\rightarrow \boldsymbol{M}^{(q)}$.add($M_k$)
13: **end if**
14: **end for**
15: **end for**
16: MPI_SR($\mathbf{L}^{(q)}$); $\mathbf{L} = \Omega_1^{Q-1}\mathbf{L}^{(q)}$
17: MPI_SR($\boldsymbol{M}^{(q)}$); $\boldsymbol{M} = \Omega_1^{Q-1}\boldsymbol{M}^{(q)}$
18: MPI_Bcast($\mathbf{L}$; $\boldsymbol{M}$)

In lines 1 and 2 of Algorithm 6, distributed recipients $\tilde{\mathbf{z}}_M^{(q)}$ are read from the hard drive. From lines 3 to 15, we will determine deficient donors for recipients $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ that has $M_k < 2$. Initialize global minimum ED (denoted as e_{t_1}) and global second-minimum ED (denoted as e_{t_2}) as an infinitely large constant ε in line 4. Let $t_1 \in \tilde{\boldsymbol{A}}_R$ be an instance index that has an ED of e_{t_1}. Likewise, let $t_2 \in \tilde{\boldsymbol{A}}_R$ be an instance index that has an ED of e_{t_2} and $t_1 \neq t_2$. Let $F \in \mathbb{N}$ such that $\tilde{\mathbf{z}}_R = \tilde{\mathbf{z}}_{R1} \cup \cdots \cup \tilde{\mathbf{z}}_{RF}$. From lines 5 to 10, compute ED $\boldsymbol{e}$ between $\tilde{\boldsymbol{z}}_{M,k}^{(q)}$ and $\tilde{\mathbf{z}}_{R1}, \ldots, \tilde{\mathbf{z}}_{RF}$ iteratively by Eq. (2) and update t_1 and t_2 within iterations. In line 11, add t_1 to donor list $\mathbf{L}$ and update M_k. If M_k is still less than two, continue to add t_2 to $\mathbf{L}$ such that two donors will be guaranteed. In line 16, we can integrate $\mathbf{L}^{(q)}$ on the master processor and broadcast $\mathbf{L}$ to slaves in line 17.

Algorithm 6 Parallel function KNN

Input: unique observed patterns ${}^*\tilde{\mathbf{z}}_R$, unique missing patterns ${}^*\tilde{\mathbf{z}}_M$, donor list $\mathbf{L}$, number set of donors $\boldsymbol{M}$
Output: donor list $\mathbf{L}$

1: UniformDistr($\tilde{n}_M$)$\rightarrow s, e$
2: MPI_RD(${}^*\tilde{\mathbf{z}}_M^{(q)}$) $\rightarrow \tilde{\mathbf{z}}_M^{(q)}$
3: **for** $\tilde{\boldsymbol{z}}_{M,k}^{(q)} \in \tilde{\mathbf{z}}_M^{(q)}$ such that $M_k < 2$ **do**
4: $t_1 = t_2 = 0$ and $e_{t_1} = e_{t_2} = \varepsilon$
5: **for** $\forall\ f$ in $1:F$ **do**
6: MPI_RD$\left({}^*\tilde{\mathbf{z}}_{Rf}\right) \rightarrow \tilde{\mathbf{z}}_{Rf}$
7: Euclidean($\tilde{\boldsymbol{z}}_{M,k}^{(q)}, \tilde{\mathbf{z}}_{Rf}$) $\rightarrow \boldsymbol{e}$
8: min($e_{t_1}, \boldsymbol{e}$) $\longrightarrow e_{t_1}, t_1$
9: second_min($e_{t_2}, \boldsymbol{e}$) $\longrightarrow e_{t_2}, t_2$
10: **end for**
11: $\mathbf{L}$.add(t_1); $M_k = M_k +$ occurrence of $\tilde{\boldsymbol{z}}_{R,t_1}$
12: **if** $M_k < 2$ **then**
13: $\mathbf{L}$.add(t_2);
14: **end if**
15: **end for**
16: MPI_SR($\mathbf{L}^{(q)}$); $\mathbf{L} = \Omega_1^{Q-1}\mathbf{L}^{(q)}$
17: MPI_Bcast($\mathbf{L}$)

4.2 Parallel imputation

The EM algorithm in the estimation of cell probability is an implicit and iterative process such that it does not support a simple divide-and-conquer parallelism. Hence, we only apply necessary parallelisms to multiple linear search operations (see [1] for details of parallel cell probability). In line 1 of Algorithm 7, each slave is aware of boundary indices of distributed tasks. From lines 2 to 13, it selects FHDI donors among FEFI donors for each recipient and computes corresponding fractional weights. In line 3, compute fractional weights $\boldsymbol{w}_{kj,FEFI}^*$ for FEFI donors by Eq. (8). If the kth recipient has less than M donors such that $M_k \leqslant M$, we will adopt all possible donors in $\boldsymbol{l}_k$, where $\boldsymbol{l}_k \in \mathbf{L}$ is the donor index list for the kth recipient. In line 5, import imputed values to memory regarding $\boldsymbol{l}_k$ and add imputed values along with $\boldsymbol{w}_{kj,FEFI}^*$ to $\hat{\mathbf{Y}}^{(q)}$ in line 6. If the kth recipient has more than M donors in $\boldsymbol{l}_k$ such that $M_k > M$, we apply the tailored systematic sampling scheme [18] to efficiently select M donors as $\boldsymbol{l}_M$ in line 8. In line 9, it computes fractional weights $\boldsymbol{w}_{kj}^*$ for FHDI donors by Eq. (10). Likewise, import imputed values to memory regarding $\boldsymbol{l}_M$ and add imputed values along with $\boldsymbol{w}_{kj}^*$ to $\hat{\mathbf{Y}}^{(q)}$ in line 11. In lines 14 and 15, it simultaneously writes distributed imputed values $\hat{\mathbf{Y}}^{(q)}$ to the hard drive and integrates them as ${}^*\hat{\mathbf{Y}}$.

4.3 Parallel variance estimation

Full details about the parallel Jackknife variance estimation are available in [1]. This section will focus on the parallel linearization techniques in Algorithm 8.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

Algorithm 7 Parallel imputation

Input: raw data ${}^*\mathbf{y}$, unique missing patterns ${}^*\tilde{\mathbf{z}}_M$, donor list $\mathbf{L}$, number set of donors $\boldsymbol{M}$
Output: imputed valus ${}^*\hat{\mathbf{Y}}$

1: UniformDistr($\tilde{n}_M$)$\rightarrow s, e$
2: **for** $\forall\ k$ in $s:e$ **do**
3: compute $\boldsymbol{w}^*_{kj,FEFI}$
4: **if** $M_k \leqslant M$ **then**
5: $\boldsymbol{l}_k \rightarrow {}^*\mathbf{y}_k$; MPI_RD($^*\mathbf{y}_k$) $\rightarrow \mathbf{y}_k$
6: $\hat{\mathbf{Y}}^{(q)}$.add($\boldsymbol{w}^*_{kj,FEFI}$); $\hat{\mathbf{Y}}^{(q)}$.add($\mathbf{y}_k$)
7: **else**
8: select $\boldsymbol{l}_M = \{l \mid l \in \boldsymbol{l}_k\}$ &$|\boldsymbol{l}_M| = M$
9: Compute $\boldsymbol{w}^*_{kj}$
10: $\boldsymbol{l}_M \rightarrow {}^*\mathbf{y}_k$; MPI_RD($^*\mathbf{y}_k$) $\rightarrow \mathbf{y}_k$
11: $\hat{\mathbf{Y}}^{(q)}$.add($\boldsymbol{w}^*_{kj}$); $\hat{\mathbf{Y}}^{(q)}$.add($\mathbf{y}_k$)
12: **end if**
13: **end for**
14: MPI_WR($\hat{\mathbf{Y}}^{(q)}$) $\rightarrow {}^*\hat{\mathbf{Y}}^{(q)}$
15: ${}^*\hat{\mathbf{Y}} = \Omega_1^{Q-1} {}^*\hat{\mathbf{Y}}^{(q)}$

Algorithm 8 Parallel linearized variance estimation

Input: raw data ${}^*\mathbf{y}$, response indicator ${}^*\mathbf{r}$, imputation cells ${}^*\mathbf{z}$, imputed values ${}^*\hat{\mathbf{Y}}$
Output: variance estimator $\widehat{\boldsymbol{V}}$ of $\hat{\mu}_{FHDI}$

1: UniformDistr(p)$\rightarrow s, e$
2: MPI_RD($^*\mathbf{y}^{(q)}$) $\rightarrow \mathbf{y}^{(q)}$
3: MPI_RD($^*\mathbf{r}^{(q)}$) $\rightarrow \mathbf{r}^{(q)}$
4: MPI_RD($^*\mathbf{z}^{(q)}$) $\rightarrow \mathbf{z}^{(q)}$
5: MPI_RD($^*\widehat{\mathbf{Y}}^{(q)}$) $\rightarrow \widehat{\mathbf{Y}}^{(q)}$
6: Compute $\overline{\mathbf{y}}^{(q)}$
7: Compute $\widehat{\boldsymbol{\Theta}}^{(q)}$
8: Compute $\overline{\boldsymbol{\theta}}^{(q)}$
9: Compute $\widehat{\boldsymbol{V}}^{(q)}(\hat{\mu}_{FHDI})$
10: MPI_SR($\widehat{\boldsymbol{V}}^{(q)}(\hat{\mu}_{FHDI})$)
11: $\widehat{\boldsymbol{V}} = \Omega_1^{Q-1}\widehat{\boldsymbol{V}}^{(q)}(\hat{\mu}_{FHDI})$

In lines 1 to 5 of Algorithm 8, we read distributed raw data $\mathbf{y}^{(q)}$, response indicator $\mathbf{r}^{(q)}$, imputation cells $\mathbf{z}^{(q)}$, and imputed values $\widehat{\mathbf{Y}}^{(q)}$ from the hard drive, respectively. In line 6, one can determine sample means $\overline{\mathbf{y}}^{(q)} \in \mathbb{R}^{G\times\frac{p}{Q-1}}$ among the respondents in different imputation groups by

$$\overline{\mathbf{y}}^{(q)} = \sum_{l=s}^{e}\sum_{g=1}^{G}\left\{\frac{1}{r_{gl}}\sum_{i\in\boldsymbol{S}_{gl}}\delta_{il}y_{il}\right\}. \tag{28}$$

where $\boldsymbol{S}_{gl} \in \mathbb{N}^{n_{gl}}$ is the index set of the units in imputation cell g. And r_{gl} is the number of observed units in cell g of y_l. However, if it has multiple donors, one may not determine an imputation group for a missing unit (i.e., $\delta = 0$). UP-FHDI uses conditional probability to determine the imputation cells for missing units. Suppose a recipient $\boldsymbol{z}_i = (\boldsymbol{z}_{i,mis}, \boldsymbol{z}_{i,obs})$ has two donors $(\boldsymbol{z}^{*(h_1)}_{i,mis}, \boldsymbol{z}_{i,obs})$ and $(\boldsymbol{z}^{*(h_2)}_{i,mis}, \boldsymbol{z}_{i,obs})$. Given conditional probability $\hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_1)}_{i,mis} \mid \boldsymbol{z}_{i,obs})$ and $\hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_2)}_{i,mis} \mid \boldsymbol{z}_{i,obs})$ of donors, one can determine a imputation cell for $\boldsymbol{z}_{i,mis}$ by

- $\boldsymbol{z}_{i,mis} \in \boldsymbol{S}_{g_1}$ if $\hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_1)}_{i,mis} \mid \boldsymbol{z}_{i,obs}) > \hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_2)}_{i,mis} \mid \boldsymbol{z}_{i,obs})$.
- $\boldsymbol{z}_{i,mis} \in \boldsymbol{S}_{g_2}$ if $\hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_1)}_{i,mis} \mid \boldsymbol{z}_{i,obs}) < \hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_2)}_{i,mis} \mid \boldsymbol{z}_{i,obs})$.
- $\boldsymbol{z}_{i,mis} \in \boldsymbol{S}_{g_1}$ or $\boldsymbol{S}_{g_1}$ if $\hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_1)}_{i,mis} \mid \boldsymbol{z}_{i,obs}) = \hat{\pi}(\boldsymbol{z}_{i,mis} = \boldsymbol{z}^{*(h_2)}_{i,mis} \mid \boldsymbol{z}_{i,obs})$.

Note that conditional probabilities can be obtained from the EM algorithm in the estimation of cell probability. In line 7, compute $\hat{\boldsymbol{\Theta}}^{(q)} \in \mathbb{R}^{n\times\frac{p}{Q-1}}$ by

$$\begin{aligned}\hat{\boldsymbol{\Theta}}^{(q)} &= \sum_{l=s}^{e}\sum_{i=1}^{n}\hat{\eta}_{il} \\ &= \sum_{l=s}^{e}\sum_{i=1}^{n}\left\{\delta_{il}\bar{y}_{gl} + (1-\delta_{il})\bar{y}^*_{il} + \delta_{il}\frac{n_{gl}}{r_{gl}}(y_{il}-\bar{y}_{gl})\right\}.\end{aligned} \tag{29}$$

where $\bar{y}_{gl} \in \overline{\mathbf{y}}^{(q)}$ and $\bar{y}^*_{il}$ is defined in Eq. (19). Compute $\bar{\boldsymbol{\theta}}^{(q)} \in \mathbb{R}^{\frac{p}{t-1}}$ by

$$\bar{\boldsymbol{\theta}}^{(q)} = \sum_{l=s}^{e}\left\{\frac{1}{n}\sum_{i=1}^{n}\hat{\boldsymbol{\Theta}}^{(q)}_{il}\right\}. \tag{30}$$

In line 9, the variance estimator $\widehat{\boldsymbol{V}}(\hat{\mu}_{FHDI})^{(q)} \in \mathbb{R}^{\frac{p}{Q-1}}$ on slave processor q is given by

$$\widehat{\boldsymbol{V}}(\hat{\mu}_{FHDI})^{(q)} = \sum_{l=s}^{e}\left\{\frac{1}{n(n-1)}\sum_{i=1}^{n}\left(\widehat{\boldsymbol{\Theta}}^{(q)}_{il} - \bar{\boldsymbol{\theta}}^{(q)}_{l}\right)^2\right\}. \tag{31}$$

and we integrate $\widehat{\boldsymbol{V}}(\hat{\mu}_{FHDI}) \in \mathbb{R}^p$ on the master processor in lines 10 and 11 such that $\widehat{\boldsymbol{SE}}$ is computed by $\widehat{\boldsymbol{SE}} = \sqrt{\widehat{\boldsymbol{V}}}$.

5 Validation

This section conducts Monte Carlo (MC) simulations, a comparison between Jackknife and linearized variance estimation, and a performance comparison of UP-FHDI against baseline imputations (naive method and an advanced machine learning-based method) with large synthetic and real-world datasets. We provide a step-by-step illustration in APPENDIX I on how to use UP-FHDI.

5.1 Monte Carlo simulations versus UP-FHDI

Let $\mathbf{U}(n, p, \eta)$ denote a finite population with n instances and p variables issued by η missing rate in proportion. Unless otherwise stated, missingness under MCAR is created by the Bernoulli distribution as $\delta_{il} \sim Ber(\tau)$ where τ is the probability of response. Let B be the MC simulation size. MC simulations follow steps:

(MC1) Generate synthetic data by

$$\begin{aligned}Y_1 &= 1 + e_1 \\ Y_2 &= 2 + \rho e_1 + \sqrt{1-\rho^2}e_2 \\ Y_3 &= Y_1 + e_3 \\ Y_4 &= -1 + 0.5Y_3 + e_4.\end{aligned} \tag{32}$$

where e_1, e_2, e_4 are randomly generated by the standard normal distribution. The term e_3 is randomly generated by the gamma distribution. Thus, $\mu_1 = 1, \mu_2 = 2, \mu_3 = 2$, and $\mu_4 = 0$.
(MC2) Create 30% missingness to the synthetic data.
(MC3) Perform imputation of UP-FHDI with random donor selection.
(MC4) Perform Jackknife variance estimation.
(MC5) Repeat steps (MC1) to (MC4) over B times.

The relative bias (RB) of the standard error of the imputed point estimator (i.e., mean estimator) $\widetilde{Y}_{FHDI}$ is defined as

$$RB = \frac{E(\widehat{SE}) - SE(\widetilde{Y}_{FHDI})}{SE(\widetilde{Y}_{FHDI})}, \tag{33}$$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

where $\widehat{SE}$ is the square root of the variance estimator $\widehat{V}(\widehat{Y}_{FHDI})$ and $SE(\widetilde{Y}_{FHDI})$ is the square root of the sampling variance of the mean estimator $\widetilde{Y}_{FHDI}$.

$$E(\widehat{SE}) = \frac{1}{B}\sum_{b=1}^{B}\sqrt{\widehat{V}(\widehat{Y}_{FHDI})^{(b)}}, \tag{34}$$

where $\widehat{V}(\widehat{Y}_{FHDI})^{(b)}$ is the variance estimator from the bth MC sample.

$$SE(\widetilde{Y}_{FHDI}) = \sqrt{\frac{1}{B}\sum_{b=1}^{B}\left(\widetilde{Y}_{FHDI}^{(b)} - \overline{Y}_{FHDI}\right)^2}, \tag{35}$$

where $\widetilde{Y}_{FHDI}^{(b)}$ is the imputed point estimator from the bth MC sample and $\overline{Y}_{FHDI}$ is given by

$$\overline{Y}_{FHDI} = \frac{\sum_{b=1}^{B}\widetilde{Y}_{FHDI}^{(b)}}{B}. \tag{36}$$

We present MC simulation results of UP-FHDI without SIS in Table 1. Likewise, MC results using SIS are shown in Table 2. The mean estimates $\overline{Y}_{FHDI}$ from MC simulations have negligible differences with true column mean μ, and thus good accuracy of the UP-FHDI imputation is affirmed. Average RB in Tables 1 and 2 are 1.96% and 2.87% (less than 5 percent), validating the Jackknife variance estimator.

5.2 Linearized variance estimation vs Jackknife

In Fig. 3, we investigate whether the linearized variance estimator is an accurate substitute for the Jackknife variance estimator. Absolute difference of standard errors (ADSE) is defined as

$$\text{ADSE} = \frac{1}{p}\sum_{l=1}^{p} AD_l = \frac{1}{p}\sum_{l=1}^{p}\left|\frac{\widehat{SE}_{Linear,l} - \widehat{SE}_{Jack,l}}{\widehat{SE}_{Jack,l}}\right|. \tag{37}$$

where AD_l is the absolute difference between $\widehat{SE}_{Linear,l}$ and $\widehat{SE}_{Jack,l}$. Note that $\widehat{SE}_{Linear,l}$ and $\widehat{SE}_{Jack,l}$ are the standard errors of the mean estimator of y_l using linearization techniques and the Jackknife method, respectively. Remarkably, ADSE between Jackknife and linearization gradually converges to an acceptable datum as datasets become larger. Meanwhile, we confirm an alignment between linearized and the Jackknife variance estimators with a minor ADSE in Table 3 with an ultra dataset. Overall, the linearized variance estimator is a good alternative to the Jackknife variance estimator for ultra data (e.g., $n \geq 20,000$). It should be noted that the linearized variance estimation may not be recommended for a small-sized data curing due to substantial difference from the Jackknife (see large ADSE in the left range of Fig. 3). Although the Jackknife is recommended for small to medium-sized datasets, APPENDIX II affirms a significant advancement in the computational efficiency of the linearization techniques.

5.3 Baseline imputations versus UP-FHDI with large synthetic and real-world datasets

Next, we validate the performance and accuracy of UP-FHDI with large real-world and ultra synthetic datasets. Table 4 presents the adopted incomplete datasets, which are publicly accessible in IEEE DataPort [45]. Some variables in real-world datasets have significantly skewed data distributions, as shown in APPENDIX III. For generating

TABLE 1: MC results of UP-FHDI with the Jackknife method based on 1800 simulation runs. Input data is $\mathbf{U}(1000, 4, 0.3)$ and no variable reduction is applied

	y_1	y_2	y_3	y_4
$E(\widehat{SE})$	0.0350	0.0367	0.0497	0.0519
$SE(\widetilde{Y}_{FHDI})$	0.0356	0.0373	0.0516	0.0523
RB (%)	-1.64	-1.62	-3.74	-0.85
$\overline{Y}_{FHDI}$	1.002	1.999	2.001	0.001
true μ	1	2	2	0

synthetic data, Eq. (38) is used until we obtain total p variables:

$$\begin{aligned} Y_i &= \begin{cases} 1 + e_i & \text{if } i = 0 \parallel i \bmod 8 \equiv 0 \\ Y_{i-1} + e_i & \text{if } i \bmod 8 \not\equiv 0 \end{cases} \\ Y_{i+1} &= Y_i + 2 + \rho \times e_i + \sqrt{1-\rho^2}e_{i+1} \\ Y_{i+2} &= Y_{i+1} + e_{i+2} \\ Y_{i+3} &= -1 + Y_i + 0.25Y_{i+2} + e_{i+3}. \end{aligned} \tag{38}$$

where $\rho = 0.5$ and e_i, e_{i+1}, e_{i+3} are randomly generated by the normal distribution with a user-defined standard deviation (SD), and the term e_{i+2} is randomly generated by the gamma distribution. We create 30% missingness to all datasets by different missing mechanisms: MCAR and missing not at random (MNAR). Although missingness is generated by MCAR for simplicity, UP-FHDI should hold for response models based on MAR. Table 4 summarises resultant synthetic data and adopted real-world datasets to compare the performance of UP-FHDI and baseline imputation methods, including naive imputation and the recently proposed Generative Adversarial Imputation Network (GAIN). The naive imputation adopts a simple mean estimator computed using observed values. GAIN is a GAN-based framework that employs an imputer network to handle the missing data [19]. Experiments show that GAIN outperforms many state-of-the-art imputation techniques, and the summary of key theories of GAIN is presented in APPENDIX IV. This study adopts the default settings to build the GAIN model. Considering the stochastic nature of GAIN, we conduct ten experiments for each dataset and average the performance measures. Let the average standard error of the mean estimator be $\overline{SE} = 1/p\sum_{l=1}^{p}\widehat{SE}_l$ and $RMSE$ is given by

$$RMSE = \sqrt{\frac{\sum_{l=1}^{p}\sum_{i=1}^{n}(y_{il} - \hat{y}_{il})^2}{n \times p}}. \tag{39}$$

where y_{il} and $\hat{y}_{il}$ are normalized units of original data and imputed values by Min-Max normalization, respectively. Using large real-world datasets (Earthquake, Bridge Strain, Travel Time, and CT Slices), UP-FHDI performs well comparable to GAIN regarding RMSE results. Note that the default-setting GAIN aborted when it was applied to Swarm, p53, and Radar (the last three rows in Table 5). It appears that GAIN may require specific extensions for large/ultra data curing. Tables 5 and 6 show that UP-FHDI consistently outperforms naive imputation with smaller $\overline{SE}$ and $RMSE$ using large real-world and ultra synthetic datasets, respectively. Similar performance of UP-FHDI under MNAR assumption can be found in APPENDIX V.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

TABLE 2: MC results of UP-FHDI with the Jackknife method based on 2000 simulation runs. Input data is $\mathbf{U}(1000, 24, 0.3)$ and four selected variables are used

	y_1	y_2	y_3	...	y_{24}
$E(\widehat{SE})$	0.0367	0.0386	0.0525	...	0.0547
$SE(\widetilde{Y}_{FHDI})$	0.0354	0.0372	0.0511	...	0.0534
RB (%)	3.82	3.81	2.68	...	2.50
$\overline{Y}_{FHDI}$	1.001	2.000	2.001	...	-0.002
true μ	1	2	2	...	0

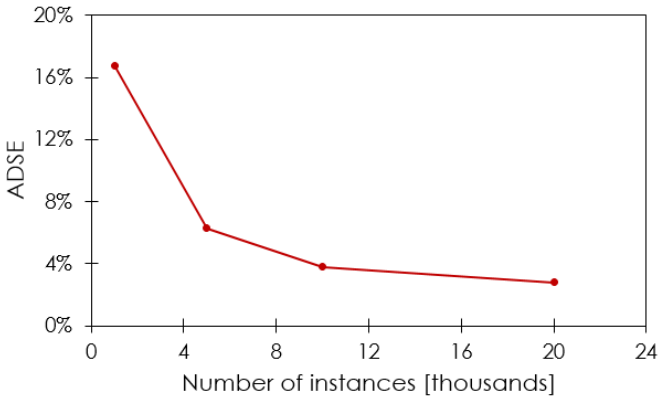


Fig. 3: Impact of the number of instances on ADSE defined in Eq. (37) with datasets $\mathbf{U}(n, 24, 0.3)$: 24 variables and 30% missing rate by varying n. SIS uses four selected variables.

TABLE 3: ADSE between the Jackknife and linearized variance estimators. Input data is $\mathbf{U}(10,000, 0.1M, 0.3)$ and SIS uses four selected variables

Methods	y_1	y_2	y_3	y_4	...	ADSE
$\widehat{SE}_{Jack}$	0.0096	0.0164	0.0189	0.0178	...	
$\widehat{SE}_{Linear}$	0.0097	0.0166	0.0193	0.0181	...	
AD	0.99%	1.17%	1.79%	1.73%	...	1.41%

TABLE 4: Summary of adopted incomplete synthetic and real-world datasets $\mathbf{U}$(instances, variables, missing rate)

Dataset	Dimension	Source
Synthetic 1	$\mathbf{U}(100, 80, 0.3)$	Eq. (38)
Synthetic 2	$\mathbf{U}(10000, 0.1M, 0.3)$	Eq. (38)
Synthetic 3	$\mathbf{U}(0.1M, 10000, 0.3)$	Eq. (38)
Earthquake	$\mathbf{U}(901512, 15, 0.3)$	USGS [46]
Bridge Strain	$\mathbf{U}(492641, 31, 0.3)$	InTrans [47]
Travel Time	$\mathbf{U}(23772, 50, 0.3)$	IEEE DataPort
CT Slices	$\mathbf{U}(53500, 380, 0.3)$	UCI [48]
Swarm	$\mathbf{U}(24016, 2400, 0.3)$	UCI
p53	$\mathbf{U}(31159, 5408, 0.3)$	UCI
Radar	$\mathbf{U}(325834, 175, 0.3)$	UCI

6 Cost Analysis and Scalability

The cost model is important since it gives audiences an insight into the computational performance of UP-FHDI. Given constant time for a unit operation, one can build a cost model based on the total number of unit operations in computation and communication. Let α be the basic computational cost per unit. Although there exist intra-node and inter-node communication transfer costs, let β be the overall communication transfer cost per unit for simplicity and $\mathcal{L}$ be the communication startup cost. Likewise, IO communication can be approximated by a startup cost and a unit transfer cost in the same way as inter-node communication. IO communication transfer cost can be classified as contiguous (denoted as γ_1) and non-contiguous types (denoted as γ_2). Contiguous IO makes a single IO request and allows access to a contiguous chunk of data at a time, whereas non-contiguous IO must be accessed by making separate function calls to access each individual contiguous piece iteratively, and thus $\gamma_2 \gg \gamma_1$ [49]. Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the associated startup cost for contiguous and non-contiguous IO, respectively. Typically, $\mathcal{L}_2 \gg \mathcal{L}_1$. Let $s = \max(n, p)$ and thus total running time $T(Q)$ with Q available processors can be expressed by

$$T(Q) \approx \frac{\alpha^{'}}{Q} + \beta^{'} Q. \tag{40}$$

where $\alpha^{'} = O(s^3)\alpha + O(s^2)\gamma_2$ and $\beta^{'} = O(\mathcal{L}_2)$. By substituting Eq. (40), the scalability of $\frac{T(Q)}{T(c_p Q)}$ is given by

$$\frac{T(Q)}{T(c_p Q)} = c_p \times \frac{\alpha^{'} + Q^2 \beta^{'}}{\alpha^{'} + c_p^2 Q^2 \beta^{'}}. \tag{41}$$

where $c_p \in \mathbb{N}^+$. See detailed inference of Eqs. (40) and (41) in APPENDIX VI. Figs. 4 and 5 study UP-FHDI scaling performance by varying Q. Overall, UP-

TABLE 5: Comparison of $\overline{SE}$ and $RMSE$ between UP-FHDI, naive imputation, and GAIN with incomplete real-world datasets under MCAR

Dataset	Method	Var. est	SIS	$\overline{SE}$	$RMSE$
Earthquake	Naive	×	×	0.0094	0.0445
	GAIN	×	×	×	0.0469
	UP-FHDI	Linearization	×	0.0088	0.0287
Bridge Strain	Naive	×	×	0.0724	0.0896
	GAIN	×	×	×	0.0869
	UP-FHDI	Linearization	×	0.0676	0.0341
Travel Time	Naive	×	×	0.3902	0.0164
	GAIN	×	×	×	0.0122
	UP-FHDI	Linearization	4	0.3788	0.0136
	UP-FHDI	Jackknife	4	0.3405	0.0136
CT Slices	Naive	×	×	0.0016	0.1319
	GAIN	×	×	×	0.2271
	UP-FHDI	Linearization	90	0.0015	0.0696
	UP-FHDI	Jackknife	90	0.0015	0.0696
Swarm	Naive	×	×	2.1644	0.0851
	UP-FHDI	Linearization	50	2.1312	0.0539
	UP-FHDI	Jackknife	50	2.0593	0.0539
p53	Naive	×	×	0.0076	0.0224
	UP-FHDI	Linearization	15	0.0073	0.0173
	UP-FHDI	Jackknife	15	0.0073	0.0173
Radar	Naive	×	×	2.6030	0.0557
	UP-FHDI	Linearization	35	2.4442	0.0291
	UP-FHDI	Jackknife	35	2.3476	0.0291

TABLE 6: Comparison of $\overline{SE}$ and $RMSE$ between UP-FHDI and naive imputation with ultra incomplete datasets $\mathbf{U}(10000, 0.1M, 0.3)$ by varying SD used in synthetic data generation. SIS uses four selected variables

SD	Method	$\overline{SE}$	$RMSE$	Method	$\overline{SE}$	$RMSE$
3		0.052	0.078		0.062	0.084
5	UP-FHDI	0.084	0.081	Naive	0.100	0.091
8		0.134	0.083		0.160	0.096

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

FHDI yields favorable scalability that agrees with the cost models of speedup and running time. Due to the inevitable use of non-contiguous IO, UP-FHDI does not exhibit the ideal linear speedup. Table 7 presents the execution time of UP-FHDI stages for curing extremely big data. Remarkably, UP-FHDI has no limitations to cure an incomplete dataset $\mathbf{U}(1M, 10000, 0.3)$ in a volume of 80 GB, although Stampede2 has the maximum run time limit (48 hours).

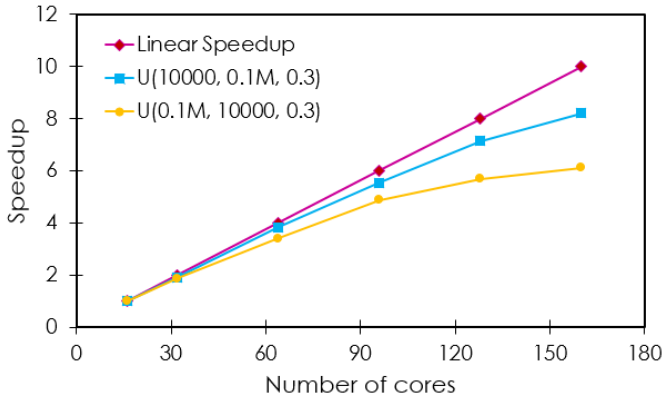


Fig. 4: Speedup of UP-FHDI with input datasets $\mathbf{U}(10000, 0.1M, 0.3)$ and $\mathbf{U}(0.1M, 10000, 0.3)$. Note that we adopt 4 selected variables for SIS.

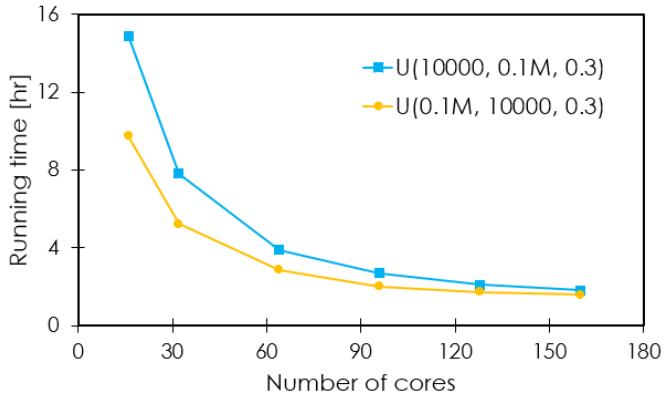


Fig. 5: Running time of UP-FHDI with input datasets $\mathbf{U}(10000, 0.1M, 0.3)$ and $\mathbf{U}(0.1M, 10000, 0.3)$. Note that we adopt 4 selected variables for SIS.

TABLE 7: Execution time of UP-FHDI stages with extremely big datasets. We adopt SIS with 8 selected variables for $\mathbf{U}(0.5M, 10000, 0.3)$ and 14 selected variables for $\mathbf{U}(1M, 10000, 0.3)$. Stage numbers i $\sim$ iv represent cell construction, estimation of cell probability, imputation, and variance estimation, respectively. Note that all execution time is measured in hours

Datasets	Q	i	ii	iii	iv	$T(Q)$
$\mathbf{U}(0.5M, 10000, 0.3)$	192	10.13	0.01	0.18	1.64	11.96
$\mathbf{U}(1M, 10000, 0.3)$	240	28.47	0.04	0.27	5.42	34.20

7 Impact of UP-FHDI on Deep Learning

The impact of FHDI on the subsequent ML has been extensively discussed in [3] with datasets of moderate size. Following a similar methodology, we investigate the impact of UP-FHDI on deep learning with ultra incomplete datasets. We segment the investigation as the following procedures:

(i) Generate an ultra synthetic dataset $\mathbf{U}(n, p)$ and p variables are indexed by $\{0, 1, \ldots, p-1\}$. Split $\mathbf{U}(n, p)$ as disjoint $\mathbf{U}_1(0.7n, p)$ and $\mathbf{U}_2(0.3n, p)$ such that $\mathbf{U}_1 \cup \mathbf{U}_2 = \mathbf{U}$.

(ii) Perform the two-staged feature selection on $\mathbf{U}(n, p)$ regarding the target variable indexed by $p-1$, and thus obtain a list of important features $\boldsymbol{v} \in \mathbb{N}^v$. Note that the target variable is guaranteed to be included in $\boldsymbol{v}$.

(iii) Create 30% missingness to $\mathbf{U}_1(0.7n, p)$ and cure it with either of imputation methods to derive imputed values $\widehat{\mathbf{U}}_1(0.7n, p)$.

(iv) Extract important features from $\mathbf{U}_1(0.7n, p)$, $\widehat{\mathbf{U}}_1(0.7n, p)$, and $\mathbf{U}_2(0.3n, p)$ regarding $\boldsymbol{v}$ such that we will have training datasets $\mathbf{U}_1(0.7n, v)$ and $\widehat{\mathbf{U}}_1(0.7n, v)$, and a test dataset $\mathbf{U}_2(0.3n, v)$.

(v) Build deep learning models with training datasets $\mathbf{U}_1(0.7n, v)$ and $\widehat{\mathbf{U}}_1(0.7n, v)$, respectively. Test different predictive models on the same $\mathbf{U}_2(0.3n, v)$ and compute $nRMSE$ by

$$nRMSE = \frac{\sqrt{\sum_{i=1}^{0.3n}\{(\hat{y}_{i,cured} - y_i)^2/0.3n\}}}{\sqrt{\sum_{i=1}^{0.3n}\{(\hat{y}_{i,ori} - y_i)^2/0.3n\}}}. \quad (42)$$

where y_i is the ith unit of the target variable in $\mathbf{U}_2(0.3n, v)$. And $\hat{y}_{i,cured}$ and $\hat{y}_{i,ori}$ are the predictions of y_i based on $\widehat{\mathbf{U}}_1(0.7n, v)$ and $\mathbf{U}_1(0.7n, v)$, respectively.

For deep learning predictions, this study adopts the R package $h2o$ [50]. We enable the ReLU activation function and four hidden layers, and twenty neurons in each layer for the implementation.

As a complement to the above investigation, the two-staged feature selection is presented as follows. Ultrahigh dimensional data (e.g., $p > 10,000$) may pose challenges to the direct use of existing ML methods. Feature selection has been proven to be effective in removing redundant features to improve subsequent learning performance [51]. Generally, feature selection techniques can be classified into three groups: (i) wrapper methods [52], [53], (ii) filter methods [54], [55], and (iii) embedded methods [56], [30]. To investigate the impact of UP-FHDI on deep learning, we need to have a subset of v features such that $v \ll p$. This paper proposes a two-staged feature selection method that leverages the mutual information (shrink the large-sized to the medium-sized features) and then graphical LASSO (shrink the medium-sized to a final subset of v features). Firstly, we develop a parallel filter method based on mutual information (MI) to reduce to v_I (e.g., up to 100) features. The MI provides a measure of how much useful information is conveyed to the target variable by including a predictor. Hence, including features with high MI will help to boost prediction accuracy. Consider $\mathbf{y}_I = \{y_0, \ldots, y_{p-2}\}$

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

as predictors and y_{p-1} as the target variable. Let $\mathbf{y}_I^{(q)}$ be the distributed predictors of size $p^{(q)}$ on the slave processor q. Let $\mathbf{z}_I^{(q)}$ be discrete values of $\mathbf{y}_I^{(q)}$. Denote MI of the distributed predictors as $\boldsymbol{MI}^{(q)}$. Key steps of the MI-based parallel filter method are:

(MI1) Discretize $\mathbf{y}_I^{(q)}$ to $\mathbf{z}_I^{(q)}$ by categories $\boldsymbol{G}^{(q)}$ using the estimated sample quantiles. Similarly, discrete y_{p-1} to z_{p-1} by the category G_{p-1}.

(MI2) Compute $\boldsymbol{MI}^{(q)}$ by:

$$\boldsymbol{MI}^{(q)} = \sum_{l=0}^{p^{(q)}-1} \left\{ \sum_{g=1}^{G_l} \sum_{k=1}^{G_{p-1}} p(g,k) \log \frac{p(g,k)}{p(g)p(k)} \right\}, \tag{43}$$

where $G_l \in \boldsymbol{G}^{(q)}$, $p(g)$ and $p(k)$ are marginal probability, and $p(g,k)$ is joint probability.

(MI3) Aggregate $\boldsymbol{MI}^{(q)}$ on the master processor by

$$\boldsymbol{MI} = \Omega_1^{Q-1} \boldsymbol{MI}^{(q)}. \tag{44}$$

(MI4) Select v_I features with top MI with respect to $\boldsymbol{MI}$.

Secondly, we build graphical models based on the inverse covariance matrix (denoted as $\boldsymbol{\Sigma}^{-1}$) to further reduce v_I features to a final small-sized subset of v features. The inverse covariance matrix, commonly referred to as the precision matrix displays information about the partial correlations of variables. The zero element of $\boldsymbol{\Sigma}^{-1}$ implies the conditional independence of two variables given the rest. The sparsity pattern of the precision matrix $\boldsymbol{\Sigma}^{-1}$ reflects the graphical structure as a consequence of the Hammersley-Clifford theorem [57]. We use R package *glasso* [58] to estimate a sparse precision matrix $\boldsymbol{\Sigma}^{-1}$ using a lasso (L1) penalty (see [59] for details about graphical lasso). Assume N multivariate normal observations of dimension p such that $X_i \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $i = 1, \ldots, N$ and $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are unknown mean and covariance matrix. Let $\boldsymbol{\Theta} = \boldsymbol{\Sigma}^{-1}$ be a precision matrix. The estimation of $\boldsymbol{\Theta}$ follows steps:

(1) Let $\mathbf{W} = \mathbf{S} + \rho I$ be the estimate of $\boldsymbol{\Sigma}$ and ρ is a regularization parameter. Express $\mathbf{W}$ and $\mathbf{S}$ by

$$\mathbf{W} = \begin{pmatrix} W_{11} & w_{12} \\ w_{12}^T & w_{22} \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} S_{11} & s_{12} \\ s_{12}^T & s_{22} \end{pmatrix}. \tag{45}$$

The diagonal entries of $\mathbf{W}$ remain unchanged throughout the following steps.

(2) For $j = 1, \ldots, p$, solve the lasso problem by

$$\hat{\beta} = \min_{\beta} \left\{ \frac{1}{2} \left\| W_{11}^{1/2} \beta - b \right\|^2 + \rho \|\beta\|_1 \right\}. \tag{46}$$

where $b = W_{11}^{-1/2} s_{12}$. Fill in the W using $w_{12} = W_{11}\hat{\beta}$.

(3) Continue until the convergence of $\mathbf{W}$.

Without inverting the estimated $\mathbf{W}$, the graphical lasso algorithm enables the estimation of $\boldsymbol{\Theta}$ in a computationally efficient way. Expanding the relation $\mathbf{W}\boldsymbol{\Theta} = I$ by

$$\begin{pmatrix} W_{11} & w_{12} \\ w_{12}^T & w_{22} \end{pmatrix} \begin{pmatrix} \Theta_{11} & \theta_{12} \\ \theta_{12}^T & \theta_{22} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0^T & 1 \end{pmatrix}, \tag{47}$$

$$\begin{aligned} \theta_{12} &= -W_{11}^{-1} w_{12} \theta_{22} \\ \theta_{22} &= 1/\left(w_{22} - w_{22}^T W_{11}^{-1} w_{12} \right). \end{aligned} \tag{48}$$

Since $\hat{\beta} = W_{11}^{-1} w_{12}$, it follows that

$$\begin{aligned} \hat{\theta}_{12} &= -\hat{\beta}\hat{\theta}_{22} \\ \hat{\theta}_{22} &= 1/\left(w_{22} - w_{22}^T \hat{\beta} \right). \end{aligned} \tag{49}$$

Since we compute $\hat{\beta}$ by solving Eq. (46), the $\hat{\theta}_{12}$ has many zeros and thus estimated $\boldsymbol{\Theta}$ becomes sparse. One can gradually increase ρ to make $\boldsymbol{\Theta}$ more sparse until we obtain v selected features connected to the target variable, as shown in Fig. 6. Regarding memory usage and speedup, the parallel MI is linearly scalable with ultra data. It considers relationships between predictors and the target. By contrast, the *glasso* package is not directly applicable to ultra data but takes relationships among the medium-sized set of variables into consideration. We illustrate using the parallel MI program and *glasso* package in APPENDIX VII and VIII, respectively.

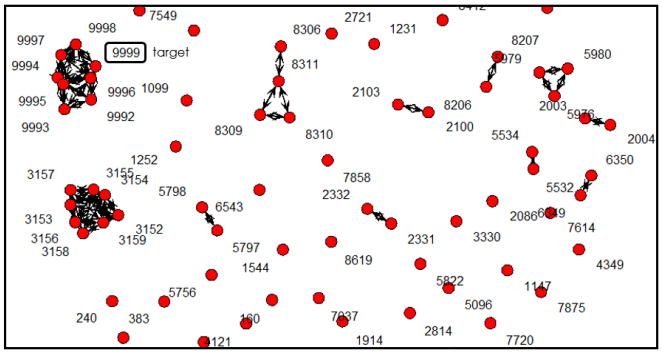


Fig. 6: Part of the graphical network model generated by graphical lasso for $\rho = 0.6$ based on $\mathbf{U}(0.1M, 100)$, which is extracted from raw data $\mathbf{U}(0.1M, 10000)$ by parallel MI. Note that node (9999) in the left top corner is the target variable while nodes 9992-9998 are its important features.

After the two-staged feature selection, this study follows the proposed investigation procedures with synthetic datasets $\mathbf{U}(0.1M, 10000)$ generated by Eq. (38). The formula indicates that seven variables are closely connected to the target variable. This fact is well captured by the two-staged feature selection. In particular, the final graphical model of seven important features (nodes indexed by 9992-9998) connected to the target variable (node indexed by 9999) is visualized as a network in Fig. 6. The validity of the two-staged feature selection is confirmed. Lastly, we investigate the impact of different imputation methods on deep learning predictions in terms of the normalized RMSE $nRMSE$ (see Table 8). In essence, $nRMSE$ being closer to 1 means that an imputation method does not disturb (or negatively impact) the prediction performance. Given Eq. (38), we prepared three ultra datasets by varying underlying SD used in the synthetic data generation. Throughout all three ultra datasets, Table 8 shows that UP-FHDI positively impacts deep learning performance compared to the naive method. The difference between UP-FHDI and naive imputation may not be significant due to the simple structure of the adopted synthetic data.

8 Future Research

There is room to improve UP-FHDI's scalability due to non-contiguous IO communication between nodes and the hard drive disk. An optimal workflow shall upgrade the parallel file system for more efficient IO communication. By new theories, the adopted Euclidean distance-based KNN shall be improved to embrace fully categorical datasets.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

TABLE 8: The impact of naive imputation and UP-FHDI on the subsequent deep learning performance. An increasing standard deviation (SD) is used for different datasets

SD	3		12		16	
Methods	Naive	UP-FHDI	Naive	UP-FHDI	Naive	UP-FHDI
$RMSE$	2.239	2.154	7.526	7.422	9.647	9.438
$nRMSE$	1.053	1.013	1.021	1.006	1.026	1.004

9 Conclusion

In a new era of big data and powerful computing, there is a strong need for a big data-oriented imputation paradigm for data-driven scientific discovery. By inheriting all existing functionalities of the general-purpose, assumption-free data-curing tool P-FHDI, this paper proposed the ultra data-oriented P-FHDI (UP-FHDI). We document the full details of UP-FHDI with respect to the adopted parallel file system, ultra data-oriented parallelisms, parallel linearization techniques, and impacts on the subsequent deep learning. The Monte Carlo simulations and comparative studies against baseline imputation methods affirm the validity of UP-FHDI, and analytical cost models exhibit its promising scaling performance. This program enables big incomplete data imputation and efficient parallel variance estimation, and ultimately improves the subsequent deep learning with the cured big data. We share all the developed codes, examples, and documents in [60].

Acknowledgments

This research is supported by National Science Foundation (NSF) grant number OAC-1931380. The high-performance computing facility used for this research is partially supported by the HPC@ISU equipment at ISU, some of which have been purchased through funding provided by NSF CNS 1229081 and CRI 1205413. Ultra data applications of this paper used the Extreme Science and Engineering Discovery Environment (XSEDE), NSF ACI-1548562.

References

[1] Y. Yang, J. kwang Kim, and I. H. Cho, "Parallel fractional hot deck imputation and variance estimation for big incomplete data curing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3912–3926, 2020.

[2] H. Kang, "The prevention and handling of the missing data," *Korean Journal of Anesthesiology*, vol. 64, no. 5, p. 402, 2013.

[3] I. Song, Y. Yang, J. Im, T. Tong, C. Halil, and I. Cho, "Impacts of fractional hot-deck imputation on learning and prediction of engineering data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 12, pp. 2363–2373, 2019.

[4] T. A. Myers, "Goodbye, listwise deletion: Presenting hot deck imputation as an easy and efficient tool for handling missing data," *Communication Methods and Measures*, vol. 5, no. 4, pp. 297–310, 1999.

[5] J. W. Graham, "Missing data analysis: Making it work in the real world," *Annual review of psychology*, vol. 60, pp. 549–576, 2009.

[6] A. R. T. Donders, G. J. van der Heijden, T. Stijnen, and K. G. Moons, "Review: A gentle introduction to imputation of missing values," *Journal of Clinical Epidemiology*, vol. 59, no. 10, pp. 1097–1091, 2006.

[7] J. W. Graham, A. E. Olchowski, and T. D. Gilreath, "Review: A gentle introduction to imputation of missing values," *Prevention Science*, vol. 8, no. 3, pp. 206–213, 2007.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[9] D. Rubin, "Multiple imputation after 18+ years," *Journal of the American Statistical Association*, vol. 91, no. 434, pp. 473–489, 1996.

[10] S. van Buuren and K. Groothuis-Oudshoorn, "mice: Multivariate imputation by chained equation in r," *Journal of Statistical Software*, pp. 1–68, 2010.

[11] S. Yang and J. K. Kim, "A note on multiple imputation for method of moments estimation," *Biometrika*, vol. 103, no. 1, pp. 244–251, 2016.

[12] X. Xie and X.-L. Meng, "Dissecting multiple imputation from a multi-phase inference perspective: what happens when god's, imputer's and analyst's models are uncongenial?" *Statistica Sinica*, vol. 27, no. 4, pp. 1485–1594, 2017.

[13] K. Graham and L. Kish, "Some efficient random imputation methods," *Communication in Statistic-Theory and Methods*, vol. 13, no. 16, pp. 1919–1939, 1984.

[14] R. E. Fay, "Alternative paradigms for the analysis of imputed survey data," *Journal of the American Statistical Association*, vol. 91, no. 434, pp. 490–498, 1996.

[15] J. K. Kim and W. Fuller, "Fractional hot deck imputation," *Biometrika*, vol. 91, no. 3, pp. 559–578, 2004.

[16] G. B. Durrant and C. Skinner, "Using missing data methods to correct for measurement error in a distribution function," *Survey Methodology*, vol. 32, no. 1, p. 25, 2006.

[17] J. Im, J. K. Kim, and W. A. Fuller, "Two-phase sampling approach to fractional hot deck imputation," *In Proceedings of Survey Research Methodology Section*, pp. 1030–1043, 2015.

[18] J. Im, I. Cho, and J. K. Kim, "An R package for fractional hot deck imputation," *The R Journal*, vol. 10, no. 1, pp. 140–154, 2018.

[19] J. Yoon, J. Jordon, and M. van der Schaar, "Gain: Missing data imputation using generative adversarial nets," *35th International Conference on Machine Learning*, pp. 5689–5698, 2018.

[20] S. C.-X. Li, B. Jiang, and B. M. Marlin, "Misgan: Learning from incomplete data with generative adversarial networks," *arXiv*, 2019.

[21] T. Anwar, T. Siswantining, D. Sarwinda, S. M. Soemartojo, and A. Bustamam, "A study on missing values imputation using k-harmonic means algorithm: Mixed datasets," *AIP Conference Proceeding*, vol. 2202, no. 1, p. 020038, 2019.

[22] Nurzaman, T. Siswantining, S. M. Soemartojo, and D. Sarwinda, "Application of sequential regression multivariate imputation method on multivariate normal missing data," *International Conference on Informatics and Computational Science*, pp. 1–6, 2019.

[23] K. Aristiawati, T. Siswantining, D. Sarwinda, and S. M. Soemartojo, "Missing values imputation based on fuzzy c-means algorithm for classification of chronic obstructive pulmonary disease," *AIP Conference Proceeding*, vol. 2192, no. 1, p. 060003, 2019.

[24] E. F. Akmam, T. Siswantining, S. M. Soemartojo, and D. Sarwinda, "Multiple imputation with predictive mean matching method for numerical missing data," *International Conference on Informatics and Computational Science*, pp. 1–6, 2019.

[25] T. J. Durham, M. W. Libbrecht, J. Howbert, J. Bilmes, and W. S. Noble, "Predictd parallel epigenomics data imputation with cloud-based tensor decomposition," *Nature communication*, vol. 9, no. 1, pp. 1402–1402, 2018.

[26] X. Hu, "Acceleration genotype imputation for large dataset on gpu," *Procedia Environmental Science*, vol. 8, pp. 457–463, 2011.

[27] F. Li, Z. Gui, H. Wu, J. Gong, Y. Wang, and S. Tian, "Big enterprise registration data imputation: Supporting spatiotemporalanalysis of industries in china," *Computers, Environment and Urban Systems*, vol. 70, pp. 9–23, 2018.

[28] D. J. Stekhoven and P. Buhlmann, "Missforest—non-parametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 2012.

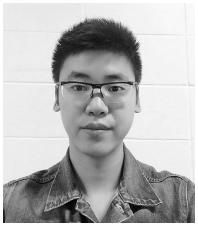
[29] B. Caffo, R. Peng, F. Dominici, T. A. Louis, and S. Zeger, "Parallel mcmc imputation for multiple distributed lag models: A case study in environmental epidemiology," *The Handbook of Markov Chain Monte Carlo*, pp. 493–512, 2011.

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

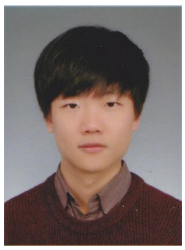
This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Transactions on Knowledge and Data Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2023.3249567

[31] A. E. Hoerl and R. W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
[32] L. Jolliffe, Ed., *Principle component analysis*. Berlin Heidelberg: Springer, 2011.
[33] J. Fan and R. Li, "Variable selection via nonconcave penalized likelihood and its oracle properties," *Journal of the American statistical Association*, vol. 96, no. 456, pp. 1348–1360, 2001.
[34] J. Fan and J. Lv, "Sure independence screening for ultrahigh dimensional feature space," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 5, pp. 849–911, 2008.
[35] E. Fix, *Discriminatory Analysis: Nonparametric Discrimination, Consistency Properties*. USAF School of Aviation Medicine, 1985, vol. 1.
[36] J. G. Ibrahim, "Incomplete data in generalized linear models," *Journal of the American Statistical Association*, vol. 85, no. 411, pp. 765–769, 1990.
[37] P. J. McCarthy, "Psuedo-replication: half samples," *Review of the International Statistical Institute*, vol. 12, no. 1, pp. 239–264, 1969.
[38] B. Efron and R. J. Tibshirani, *An introduction to the bootstrap*. Chapman and Hall/CRC, 1994.
[39] M. H. Quenouille, "Problems in plane sampling," *Annals of Mathematical Statistics*, vol. 20, no. 3, pp. 355–375, 1949.
[40] J. K. Kim and J. Shao, *Statistical Methods for Handling Incomplete Data*. Chapman and Hall/CRC, 2013.
[41] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT press, 1999, vol. 1.
[42] L. Huang and S. Liu, "Ooops: An innovative tool for io workload management on supercomputers," *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 486–493, 2020.
[43] Condo, "Condo2017: Iowa state university high-performance computing cluster system," 2017. [Online]. Available: https://www.hpc.iastate.edu/guides/condo-2017
[44] TACC, "Texas advanced computing center (tacc) at the university of texas at austin," 2017. [Online]. Available: http://www.tacc.utexas.edu
[45] Y. Yang, J. K. Kim, and I. H. Cho, "Incomplete datasets for up-fhdi," *IEEE DataPort*, 2021. [Online]. Available: https://ieee-dataport.org/open-access/incomplete-datasets-fhdi
[46] USGS, "Earthquake catalog," *U.S. Geological Survey*, 2020. [Online]. Available: https://earthquake.usgs.gov/earthquakes/search/
[47] I. H. Cho, A. Chen, A. Alipour, B. Shafei, S. Laflamme, I. Song, and J. Yan, "Development of a computational framework for big data-driven prediction of long-term bridge performance and traffic flow," *Intrans Project Reports*, 2018.
[48] D. Dheeru and G. Casey, "Uci machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[49] R. Thakur, W. Group, and E. Lusk, "Optimizing noncontiguous accesses in mpi-io," *Parallel Computing*, vol. 28, no. 1, pp. 83–105, 2002.
[50] E. LeDell, "h2o: R interface for the 'h2o' scalable machine learning platform," 2021. [Online]. Available: https://CRAN.R-project.org/package=h2o
[51] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.
[52] D. Bertsimas, A. King, and R. Mazumder, "Best subset selection via a modern optimization lens," *Annuals of Statistics*, vol. 44, no. 2, pp. 813–852, 2016.
[53] F. G. Blanchet, P. Legendre, and D. Borcard, "Forward selection of explanatory variables," *Ecology*, vol. 89, no. 9, pp. 2623–2632, 2008.
[54] J. R. Vergara and P. A. Estevez, "A review of feature selection methods based on mutual information," *Neural Computing and Applications*, vol. 24, no. 1, pp. 175–186, 2014.
[55] M. A. Hall, "Correlation-based feature selection for discrete and numeric class machine learning," *Proc. 17th International Conference Machine Learning*, pp. 359–366, 2000.
[56] A. E. Hoerl and R. W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
[57] P.-L. Loh and M. J. Wainwright, "Structure estimation for discrete graphical models: Generalized covariance matrices and their inverses," *The Annals of Statistics*, vol. 41, no. 6, pp. 3022–3049, 2013.
[58] J. Friedman, T. Hastie, and R. Tibshirani, "glasso: Graphical lasso: Estimation of gaussian graphical models," 2019. [Online]. Available: https://CRAN.R-project.org/package=glasso
[59] F. Jerome, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, no. 3, pp. 432–441, 2008.
[60] I. Cho, "Data-driven, computational science and engineering platforms repository," 2017. [Online]. Available: https://sites.google.com/site/ichoddcse2017/home/type-of-trainings/ultra-data-oriented-parallel-fhdi-up-fhdi



Yicheng Yang received his PhD degree from the department of civil, construction, and environmental engineering (CCEE) of Iowa State University (ISU) in 2021. He is currently a master's student in the department of computer science with an emphasis on data mining. His research interests include parallel imputation, ML, and data-driven engineering.



Yonghyun Kwon is a current PhD student at Iowa State university and he is a graduate research assistant at Center for Survey Statistics & Methodology(CSSM). He received the BS degree in Statistics from Seoul National University in 2020. His research interests include survey sampling, missing data analysis, and ML.

Jae-Kwang Kim received the PhD degree in Statistics from ISU in 2000. He is a fellow of American Statistical Association and Institute of Mathematical Statistics and currently a LAS Dean's professor in the department of statistics at ISU. His research interests include survey sampling, statistical analysis with missing data, measurement error models, multi-level models, causal Inference, data integration, and ML.

In Ho Cho (corresponding author) received the PhD degree in civil engineering and minor in Computational Science and Engineering from California Institute of Technology, USA in 2012. He is currently an associate professor of CCEE department, ISU. His research interests include data-driven engineering and science, computational statistics, computational science and engineering, and parallel computing.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

Appendix to Ultra Data-Oriented Parallel Fractional Hot-Deck Imputation with Efficient Linearized Variance Estimation

APPENDIX I: Detailed Explanations and Examples of UP-FHDI

The HPC environment used in this paper is Stampede2 [1], one of the Texas Advanced Computing Center's (TACC) flagship supercomputers. Stampede2 hosts 4200 Knights Landing (KNL) nodes and 1736 Skylake (SKX) nodes. Each KNL and SKX nodes include 96GB and 192GB of memory, respectively. Stampede2 mounts three shared Lustre file systems available from all nodes. Meanwhile, we tested UP-FHDI on Condo 2017 [2] of Iowa State University to confirm its compatibility. As long as the Intel modules $15.0.2 - 19.4$ are available, users can deploy the developed program for missing data curing. All the developed codes, examples, and documents are shared with associated GPL-2 in [3]. We provide full details of running UP-FHDI in the following subsections.

A. Preparation of input files

Generally, the program must start with the initial setups (summarized in Table 1), raw missing data **y**, and response indicator matrix **r**. We will give full explanations of each parameter in the initial setups hereafter. Users can either enable P-FHDI for big-n and big-p data or UP-FHDI for ultra data by adjusting i_option_ultra. For the convenience of users, the program provides various alternatives to read data efficiently. For example, P-FHDI has two options to import data. If $i\_option\_read\_data = 1$, P-FHDI will read **y**, **r**, and the rest of the setups from separate TXT files. Otherwise, P-FHDI imports all required inputs from a single TXT file for ease of use. Likewise, UP-FHDI provides two approaches to import data: MPI IO for binary files and POSIX IO for TXT files. If $i\_option\_read\_data = 1$, UP-FHDI will read **y** from two binary files in opposite data distribution by MPI IO to minimize non-contiguous reading. The data distributions in "$daty\_column\_binary.bin$" and "$daty\_row\_binary.bin$" must be column-oriented and row-oriented, respectively. Matrix **r** and initial setups will be read from another binary file and a TXT file separately. Note that the data distributions in "$datr\_column\_binary.bin$" must be column-oriented. Otherwise, UP-FHDI will read **y**, **r**, and the rest of the setups by POSIX IO from separate TXT files. $i\_option\_read\_data = 1$ is strongly preferred for UP-FHDI if data is ultra-large since MPI IO is much faster and scalable rather than POSIX IO. $i\_option\_read\_data = 0$ is designed to test UP-FHDI with easy-to-use examples. The maximum data volume imported by POSIX IO is strictly restricted by the available memory of the master processor. Overall, users shall pay extra attention to an appropriate choice of IO method based on their purpose.

$i\_option\_perform$ has four options for P-FHDI: the options of 1 or 4 perform all stages of P-FHDI using automatic or user-defined imputation cells, respectively. Note that $i\_user\_defined\_datz = 1$ is mandatory if users enable P-FHDI with user-defined imputation cells. Users can individually perform cell construction or estimation of cell probability by options 2 or 3 for specific needs. However, UP-FHDI only supports $i\_option\_perform = 1$ and thus all stages of UP-FHDI will be performed. $i\_option\_imputation$ provides two options for P-FHDI, where 1 for FEFI and 2 for FHDI. In contrast, UP-FHDI only allows FHDI with the default option 2. Moreover, users can skip variance estimation by setting $i\_option\_variance = 0$. Otherwise, the program performs variance estimation as the default. i_option_merge controls the C++ standard random number generator. A choice of 0 is desired if users intend to reproduce the results for validation. Otherwise, we enable randomness in cell collapsing, k-nearest neighbors (KNN), and donor selection. i_donor defines the total number of donors used in the imputation process. [4] recommends five as the default number of donors after detailed case studies. $i\_option\_collapsing > 0$ activates the sure independent screening (SIS) to reduce all variables to a user-defined number of selected variables. Otherwise, no variable reduction is adopted with $i\_option\_collapsing = 0$. $i\_option\_SIS\_type$ provides different methods for SIS and is meaningful only if $i\_option\_collapsing > 0$. $i\_option\_cellmake = 1$ enables the cell collapsing to generate artificial donors in cell construction. Alternatively, the default $i\_option\_cellmake = 2$ performs KNN to determine deficient donors. Options 1 and 2 for $i\_option\_var\_type$ run the Jackknife and linearized variance estimation, respectively. Note that this option is meaningful only if $i\_option\_variance = 1$ and P-FHDI merely allows the Jackknife variance estimation.

The following parameters are designed to reduce memory usage. We suggest users adjust these parameters accordingly for different ultra datasets if more efficient memory usage is desired. $top\_correlation$ defines the number of top-ranking variables used for the extraction of selected variables in SIS. An optimal $top\_correlation$ is the least value that guarantees extraction of $i\_option\_collapsing$ selected variables. $memory$ defines available memory per MPI task. The default value of 8 is suitable for 24 tasks per SKX node on Stampede2. If UP-FHDI runs out of memory, try to request fewer tasks per node to give each task more memory. Here is an example of initial settings:

```
i_option_read_data      1
i_option_ultra          1
i_option_perform        1
nrow                    100
ncol                    80
i_option_imputation     2
i_option_variance       1
i_option_merge          0
i_donor                 5
i_user_defined_datz     0
i_option_collapsing     4
i_option_SIS_type       3
top_correlation         1000
i_option_cellmake       2
i_option_var_type       2
memory                  8
```

Except for initial setups in Table 1, users must provide additional information about variable categories, variable types, and instance sampling weights. The variable categories define the number of categories of all variables created in imputation cells. The maximum value of a category is 35 and [4] suggests categories of each variable between 30 and 35 after detailed case studies. In practice, a large category is not always substantially beneficial, but computations increase monotonically. Hence, starting with a small number of categories is reasonable. For example, users can define three categories for all variables by:

```
category
3          3          3          3          ...
```

Moreover, UP-FHDI allows reserving non-collapsible categorical variables in the cell construction process by setting

```
NonCollapsible_categorical
1          0          0          0          ...
```

The first variable is strictly considered non-collapsible categorical, and the rest of the variables are considered continuous or collapsible categorical. As a result, UP-FHDI will skip the categorization for the first variable. Lastly, users should provide sampling weights for all instances by

```
weight
 1
 1
 1
 1
 1
...
```

B. Compiling of UP-FHDI

This study uses the default version of the Intel compiler (i.e., Intel/18.0.2) on Stampede2. To compile UP-FHDI, run the command:

```
mpicxx -o main_MPI main_MPI.cpp
```

where -o ** represents the name of an executable exact file and "`main_MPI.cpp`" is the actual file name of the main function of UP-FHDI. Slurm is the adopted cluster job management system on Stampede2 for job scheduling. Submitting a job script to Slurm is a standard method to run an MPI program on a high-performance computing (HPC) system. This method allows users not to connect while the job is pending or executing. One can submit a job script into the queue by issuing:

TABLE 1: Summary of options for the initial settings.

- i_option_ultra (default: 1):
  0: Perform P-FHDI 1: Perform UP-FHDI
- $i\_option\_read\_data$ (default: 1):
  0: If $i\_option\_ultra = 1$, UP-FHDI reads raw data from "$daty.txt$", the response indicator matrix from "$datr.txt$", and the rest of the settings from "$input.txt$", separately. If $i\_option\_ultra = 0$, P-FHDI reads all inputs from "$input.txt$"
  1: If $i\_option\_ultra = 1$, UP-FHDI reads raw data from "$daty\_column\_binary.bin$" and "$daty\_row\_binary.bin$", the response indicator matrix from "$datr\_column\_binary.bin$", and the rest of the settings from "$input.txt$", separately. If $i\_option\_ultra = 0$, P-FHDI reads raw data from "$daty.txt$", the response indicator matrix from "$datr.txt$", and the rest of the settings from "$input.txt$", separately
- $i\_option\_perform$ (default: 1):
  1: Perform all stages using the automatic $\mathbf{z}$ 2: Perform cell construction only
  3: Perform estimation of cell probability only 4: Perform all stages using a user-defined $\mathbf{z}$
- $nrow$: The number of instances in $\mathbf{y}$
- $ncol$: The number of variables in $\mathbf{y}$
- $i\_option\_imputation$ (default: 2):
  1: Perform FEFI 2: Perform FHDI
- $i\_option\_variance$ (default: 1):
  0: Skip variance estimation 1: Perform variance estimation
- i_option_merge (default: 0):
  0: Turn on the fixed seed 1: Turn on the standard random seed generator
- i_donor (default: 5):
  Adopt a user-defined integer as the number of donors used to fill in each missing item
- $i\_user\_defined\_datz$ (default: 0):
  0: Adopt the automatic $\mathbf{z}$ matrix 1: Adopt a user-defined $\mathbf{z}$ matrix
- $i\_option\_collapsing$ (default: 4):
  Activate the sure independent screening by a user-defined number of selected variables
- $i\_option\_SIS\_type$ (default: 3):
  Sure independent screening with 1: an intersection of simple correlations; 2: a union of simple correlations; 3: a global ranking of simple correlations
- $top\_correlation$ (default: 1000):
  Number of top-ranking variables based on simple correlations to be used for the extraction of selected variables in SIS
- $i\_option\_cellmake$ (default: 2):
  1: Adopt the cell collapsing method 2: Adopt the k-nearest neighbor method
- $i\_option\_var\_type$ (default: 2):
  1: Adopt the Jackknife variance estimation 2: Adopt the linearized variance estimation
- $memory$ (default: 8):
  Available memory in gigabyte per MPI task

```
sbatch run.sbatch
```

where "`run.sbatch`" is the actual name of a job script with its expansion. [5] provides various example job scripts for MPI jobs in the different Slurm partitions, and an example script for UP-FHDI is shown below:

```
#!/bin/bash
#SBATCH -J UP-FHDI        # Job name
#SBATCH -o UP-FHDI.o%j    # Name of the output file
#SBATCH -e UP-FHDI.e%j    # Name of the error file
#SBATCH -p skx-normal     # Queue (partition) name
#SBATCH -N 1              # Total number of nodes
#SBATCH -n 24             # Total number of MPI tasks
#SBATCH -t 00:30:00       # Maximum run time (hh:mm:ss)
module load ooops         # Load OOOPS
set_io_param_batch $SLURM_JOBID 0 high # Set configurations of OOOPS
module load intel/18.0.2 # Load the Intel compiler
ibrun ./main_MPI          # Launch MPI code
```

The above example describes the most common sbatch command options. The first line of a job script must specify the interpreter that will parse non-Slurm commands by issuing `#!/bin/bash`. Users can use `#SBATCH` directives to request

computing resources, and these directives should precede all shell commands. Slurm writes all console outputs to the file "`UP-FHDI.o%j`" and error messages to the file "`UP-FHDI.e%j`", where `%j` is the numerical job ID. The directive "`-p`" specifies the used Slurm partition for your job. The currently available queues on Stampede2 are described in [5]. Users can reserve computing resources by specifying the requested number of nodes, the number of MPI tasks per node, and the maximum run time. An appropriate request of necessary computing resources is of importance rather than requesting as many resources as possible since the scheduler will have an easier time finding a slot for your job. Users can deploy OOOPS to avoid overloading the shared file system. The directive "`set_io_param_bacth`" sets the configurations of OOOPS. "`$SLURM_JOBID`" is the submitted job ID and the use of "`0`" represents the `$SCRATCH` file system. OOOPS provides four options to throttle the frequency of IO activities: `unlimit`, `high`, `medium`, and `low` from the least to the most. Moreover, one can dynamically change throttling by running the command `set_io_param` without interrupting users' applications. Finally, load the Intel compiler and launch a single application by `ibrun`.

C. Explanation of output files

Considering an incomplete dataset $\mathbf{U}(n, p, \eta)$, let $\tilde{n}_M$ and $\tilde{n}_R$ be the number of unique missing patterns and unique observed patterns in the imputation cells, respectively. We have $\tilde{n}_M + \tilde{n}_R \approx n$ if the dataset is high-dimensional. Let M be the number of selected donors. UP-FHDI generates output files summarized in Table 2 that cover all results in the R package $FHDI$. Note that the total size of output files is estimated in the worst scenario. If UP-FHDI does not successfully converge in a maximum run time, it will provide an additional $debug.txt$ file for possible reasons and the suggested solutions.

TABLE 2: Descriptions of output files of UP-FHDI.

Output files	Description	Size (bytes)	Total
$datz\_binary.bin$	Imputation cells	pn	
$uox\_binary.bin$	Unique observed patterns	$p\tilde{n}_R$	
$mox\_binary.bin$	Unique missing patterns	$p\tilde{n}_M$	
$fmat\_FHDI\_binary.bin$	Imputed values	pnM	
$final\_daty\_binary.bin$	Final complete data	pn	
$summary.txt$	Mean and variance estimates	$20p$	$pn(M+3)$

D. Example datasets for UP-FHDI

Table 4 in the manuscript provides three synthetic datasets and seven real-world datasets, which are publicly accessible in IEEE DataPort [6]. Synthetic 1 is an easy-to-use example to study different IO methods (i.e., POSIX IO for TXT files and MPI IO for binary files). Synthetic 2 and 3 are ultra incomplete data to test the strength of UP-FHDI. In particular, users can test the use of OOOPS with Synthetic 3.

APPENDIX II: Computational Advancement of Linearization Techniques

As an extension to Section 5.2 of the manuscript, Table 3 shows that parallel linearization is much faster than the parallel Jackknife method with various real-world datasets under different missing mechanisms. Therefore, a significant computational advancement of linearization techniques against the Jackknife method is affirmed.

TABLE 3: Comparison of execution time between UP-FHDI with linearization techniques and Jackknife method. All real-world datasets have 30% of missingness under MCAR and MNAR. Execution time is measured in seconds.

Mechanism	Method	Swarm	CT Slices	p53	Radar	Travel Time
MCAR	Parallel linearization	385	777	852	5462	87
	Parallel Jackknife	10763	31829	15611	16834	5149
MNAR	Parallel linearization	376	693	898	5141	88
	Parallel Jackknife	10278	30819	13335	16378	4779

APPENDIX III: Data Distributions of Example Real-World Datasets

Table 4 in the manuscript provides ten incomplete datasets to validate the proposed method. Synthetic datasets are generated by normal and gamma distributions. This section exhibits data distributions of three adopted real-world datasets (Bridge Strain, Swarm, and Earthquake). Figs. 13, 14, and 15 show that many variables in real-world datasets are significantly skewed. Therefore, UP-FHDI is adequate to handle real-world datasets with significantly skewed data distributions.

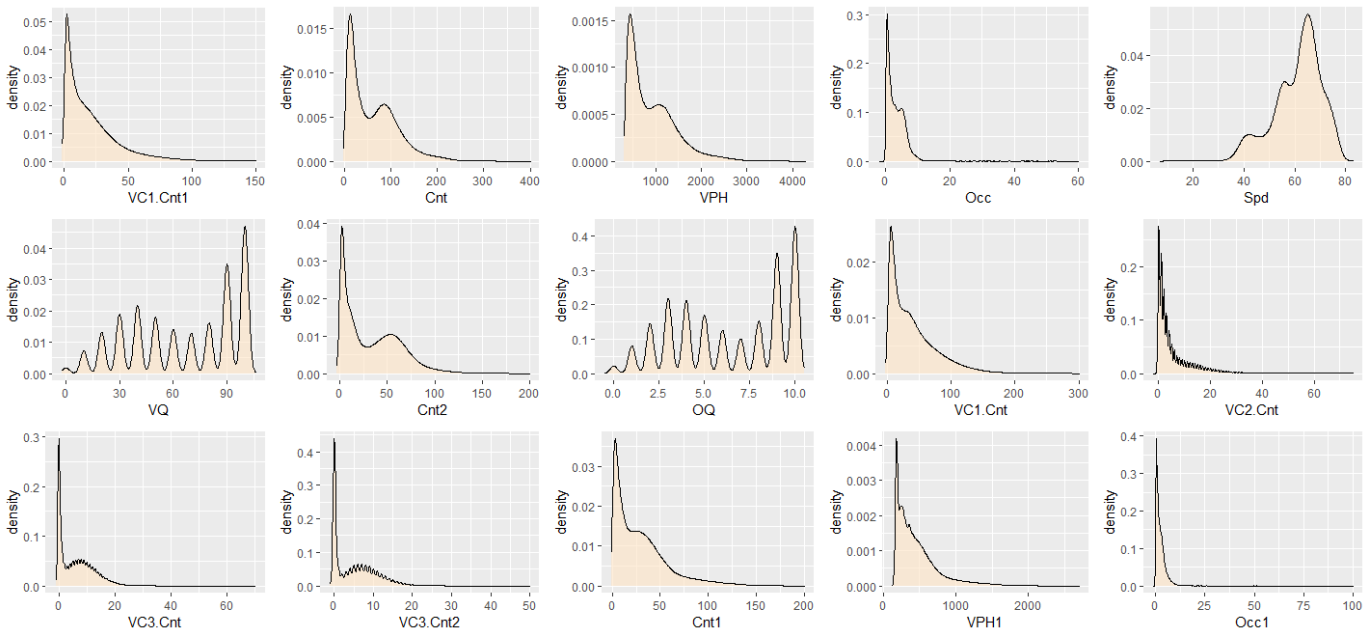


Fig. 13: Density plots of the first fifteen variables of Bridge Strain.

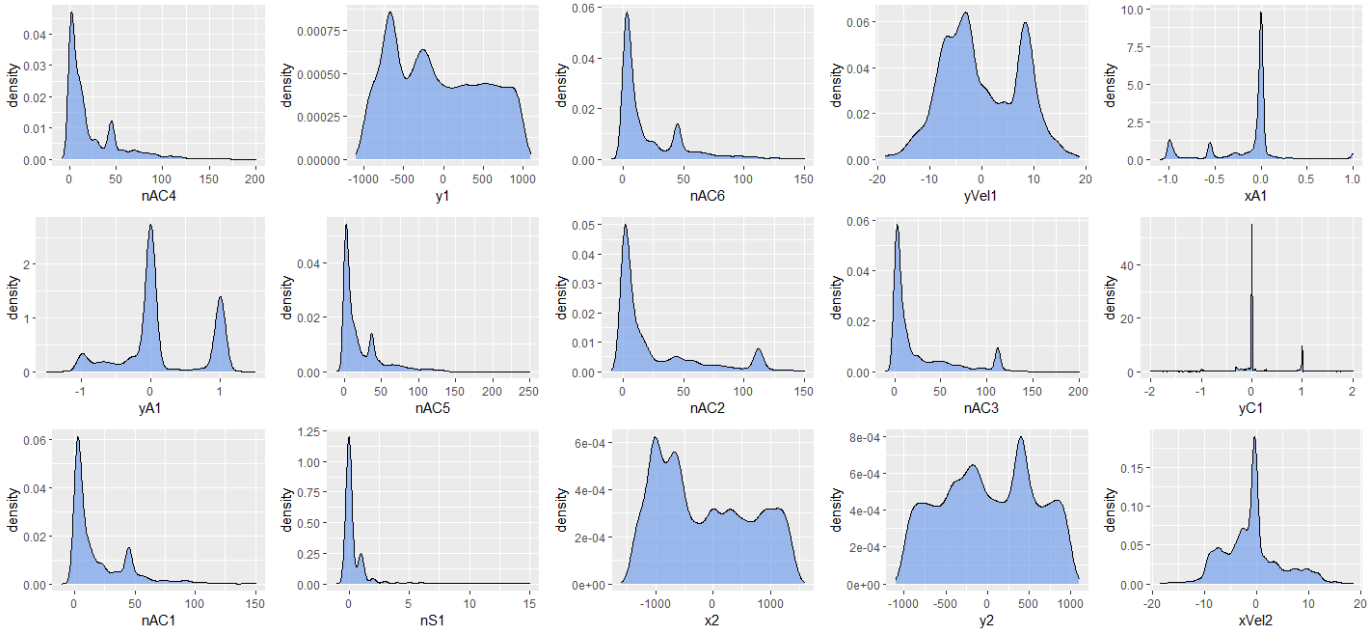


Fig. 14: Density plots of the first fifteen variables of Swarm.

APPENDIX IV: Summary of Methodologies of GAN-Based Imputation Methods

The Generative Adversarial Imputation Nets (GAIN) was proposed by [7] to accurately impute missing data. See [7] for details about GAIN. GAIN is composed of the generator (G) and the discriminator (D) with a hint vector $\mathbf{H}$. Considering a d-dimensional space, let $\mathbf{X} = (X_1, \ldots, X_d)$ be a data vector with missing values. Let $\mathbf{M}$ be the mask vector that indicates observed parts $\tilde{\mathbf{X}}$ of the data vector $\mathbf{X}$. The hint vector $\mathbf{H}$ ensures G to learn the true data distribution, accurately impute missing data, and output a completed data vector $\widehat{\mathbf{X}}$. The D takes $\widehat{\mathbf{X}}$ as input and attempts to distinguish between observed components $\tilde{\mathbf{X}}$ and imputed components $\overline{\mathbf{X}}$. Considering $\widehat{\mathbf{M}} = D(\widehat{\mathbf{X}}, \mathbf{H})$, the GAIN objective function can be expressed by

$$\min_{G} \max_{D} \mathbb{E}\left[\mathcal{L}(\mathbf{M}, \widehat{\mathbf{M}})\right] \tag{III.1}$$

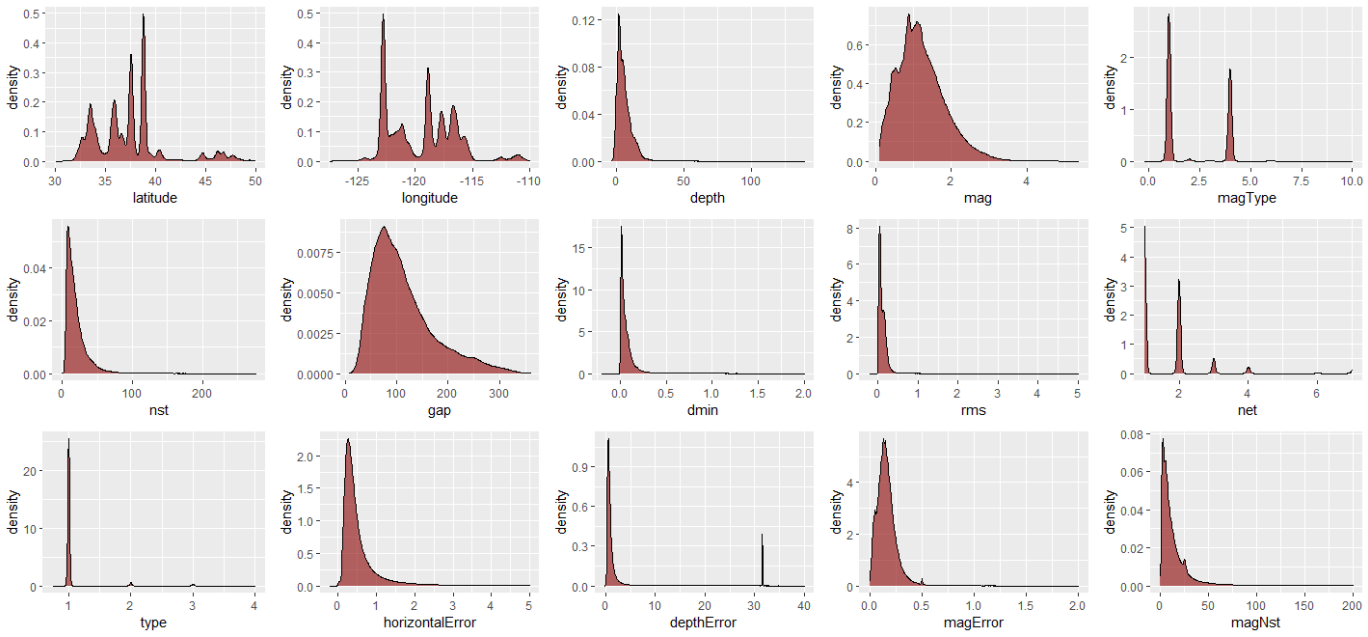


Fig. 15: Density plots of the first fifteen variables of Earthquake.

where the loss function $\mathcal{L}(a, b)$: $\{0,1\}^d \times [0,1]^d \to \mathbb{R}$ is defined by

$$\mathcal{L}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{d} \left[a_i \log(b_i) + (1 - a_i) \log(1 - b_i) \right] \tag{III.2}$$

Let $\mathbf{B} = (B_1, \ldots, B_d) \in \{0,1\}^d$ be a random variable. Suppose that the lowercase entity is always a sample of the corresponding uppercase entity, e.g., $\mathbf{b}$ is a sample of $\mathbf{B}$. The key steps of the GAIN algorithm to solve the minimax optimization problem in Eq. (III.1) are summarized as follows:

(1) Optimize D with a fixed G using mini-batches of size k_D. D is trained to minimize the sum of the discriminator loss $\mathcal{L}_D$ by

$$\min_D \sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), \widehat{\mathbf{m}}(j), \mathbf{b}(j)) \tag{III.3}$$

where $\widehat{\mathbf{m}}(j) = D(\widehat{\mathbf{x}}(j), \mathbf{m}(j))$.

(2) Optimize G using the updated D in step (1) with mini-batches of size k_G. G is trained to minimize the weighted sum of two losses by

$$\min_G \sum_{j=1}^{k_G} \mathcal{L}_G \left(\mathbf{m}(j), \widehat{\mathbf{m}}(j), \mathbf{b}(j) \right) + \alpha \mathcal{L}_M(\tilde{\mathbf{x}}(j), \widehat{\mathbf{x}}(j)) \tag{III.4}$$

where $\mathcal{L}_G$ and $\mathcal{L}_M$ are the loss applied to the missing components and observed components, respectively. The entity α is a hyper-parameter.

(3) Iterate steps (1) and (2) until training loss is converged.

GAIN was validated with various real-world missing data, and results show that GAIN outperforms many state-of-art imputation techniques.

A GAN-based framework (named MisGAN) was proposed by [8] to learn complex and high-dimensional data distribution from incomplete data. See [8] for details about MisGAN. The framework equipped an adversarially trained imputer to offer high-quality imputation. MisGAN is composed of three generator-discriminator (G, D) pairs: (G_m, D_m) for data vector $\mathbf{x}$, (G_x, D_x) for the masks $\mathbf{m} \in \{0,1\}^n$, and (G_i, D_i) for the imputer. Note that G_x and G_m are independent of each other with their own noise distribution p_ε and p_z, respectively. Let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{m}_i)\}_{i=1,\ldots,N}$ be an incomplete dataset with a distribution $p_\mathcal{D}$ and $\mathbf{z}$ be the noise. Define $\mathcal{F}_x$, $\mathcal{F}_m$, and $\mathcal{F}_i$ such that D_x, D_m, and D_i are all 1-Lipschitz. To train an imputer-equipped MisGAN, the data generating process and the imputer will be optimized by three objective functions. The (G_i, D_i) for the imputer will be optimized by the objective function:

$$\min_{G_i} \max_{D_i \in \mathcal{F}_i} \mathcal{L}_i(D_i, G_i, G_x) \tag{III.5}$$

where the imputer loss is:

$$\mathcal{L}_i(D_i, G_i, G_x) = \mathbb{E}_{\mathbf{z}\sim p_z}[D_i(G_x(\mathbf{z}))] - \mathbb{E}_{(\mathbf{x},\mathbf{m})\sim p_{\mathcal{D}},\boldsymbol{\omega}\sim p_\omega}[D_i(G_i(\mathbf{x},\mathbf{m},\boldsymbol{\omega}))] \tag{III.6}$$

Note that $\boldsymbol{\omega}$ is a random vector drawn from a noise distribution p_ω. The (G_x, D_x) for data can be optimized by the objective function:

$$\min_{G_x}\max_{D_x\in\mathcal{F}_x} \mathcal{L}_x(D_x, G_x, G_m) + \beta\mathcal{L}_i(D_i, G_i, G_x) \tag{III.7}$$

where $\beta = 0.1$ empirically and the data loss is:

$$\mathcal{L}_x(D_x, G_x, G_m) = \mathbb{E}_{(\mathbf{x},\mathbf{m})\sim p_{\mathcal{D}}}\left[D_x(f_\tau(\mathbf{x},\mathbf{m}))\right] - \mathbb{E}_{\varepsilon\sim p_\varepsilon,\mathbf{z}\sim p_\varepsilon}\left[D_x(f_\tau(G_x(\mathbf{z}), G_m(\varepsilon)))\right] \tag{III.8}$$

The masking operator f_τ fills in missing elements with a constant value τ by:

$$f_\tau(\mathbf{x},\mathbf{m}) = \mathbf{x}\odot\mathbf{m} + \tau\bar{\mathbf{m}} \tag{III.9}$$

where $\bar{\mathbf{m}}$ is the complement of $\mathbf{m}$. The (G_m, D_m) for the masks will be optimized by the objective function:

$$\min_{G_m}\max_{D_m\in\mathcal{F}_m} \mathcal{L}_m(D_m, G_m) + \alpha\mathcal{L}_x(D_x, G_x, Gm) \tag{III.10}$$

where $\alpha = 0.2$ empirically and the mask loss is:

$$\mathcal{L}_m(D_m, G_m) = \mathbb{E}_{(\mathbf{x},\mathbf{m})\sim p_{\mathcal{D}}}[D_m(\mathbf{m})] - \mathbb{E}_{\varepsilon\sim p_\varepsilon}\left[D_m(G_m(\varepsilon))\right] \tag{III.11}$$

MisGAN adopted convolutional networks for both generators and discriminators and was validated with three incomplete image datasets. The Frechet Inception Distance (FID) was computed between imputed data and original fully-observed data as the evaluation metric. The results show that MisGAN consistently outperforms baseline imputation techniques with smaller FIDs.

Our GAIN implementation is adapted from the codes released by the authors [9]. This study adopts the default settings to build the GAIN model. Considering the stochastic nature of GAIN, we conduct ten experiments for each dataset and average the performance measures. Thus, the number of iterations is adjusted for tolerable execution time. MisGAN was proved to be powerful towards incomplete image datasets [8]. Yet, the extension of MisGAN to real-valued incomplete datasets has not been discussed. Hence, this paper did not implement MisGAN for comparative studies.

APPENDIX V: Contrast between UP-FHDI and Baseline Imputation Methods under MNAR

As an extension to Section 5.3 of the manuscript, we provide a contrast between UP-FHDI and baseline imputation methods with incomplete real-world datasets under missing not at random (MNAR). Note that we create 30% missingness to all real-world datasets. The response indicator δ_{il} under MNAR is generated from $\delta_{il} \sim Ber(\zeta_{il})$ such that

$$\text{logit } \zeta_{il} = \phi_0 + \phi_1 y_{il} \tag{IV.1}$$

where ϕ_0 and ϕ_1 can be adjusted to control 30% missingness. Table 4 shows that UP-FHDI outperforms naive imputation with smaller $\overline{SE}$ and $RMSE$. Using large real-world datasets (Earthquake, Bridge Strain, Travel Time, and CT Slices), UP-FHDI performs well comparable to GAIN regarding RMSE results. Note that the default-setting GAIN aborted when it was applied to Swarm, p53, and Radar (the last three rows in Table 4). It appears that GAIN may require specific extensions for large/ultra data curing.

APPENDIX VI: Cost Analysis of UP-FHDI

As an extension to Section 6 of the manuscript, we provide a detailed inference about the cost models in Eqs. (40) and (41). Generally, we can derive the cost at four primary stages of UP-FHDI: (i) Cell construction (ii) Estimation of cell probability (iii) Imputation, and (iv) Variance estimation. For the convenience of readers, we recap important notations used in this section as follows. A finite sampled population is denoted by $\mathbf{U}$(instances, variables, missing rate) $\in \mathbb{R}^{n\times p}$ with n instances, p variables, and the associated missing rate η. Let $s = \max(n, p)$ be the number of dimensions in the worst scenario. UP-FHDI can activate SIS by setting the number of selected variables $v \neq 0$. Otherwise, UP-FHDI without variable reduction is adopted. Let t be a user-defined number of top-ranking variables used in SIS. Given Q total available processors, let α be the computational cost per element, β be the communication transfer cost per unit, and $\mathcal{L}$ be its startup cost. Consider γ_1 and γ_2 as contiguous and non-contiguous IO communication transfer costs, respectively. Similarly, let $\mathcal{L}_1$ and $\mathcal{L}_2$ be associated startup cost for contiguous and non-contiguous IO, respectively. The total execution time T can be partitioned into the computational time (denoted as $\mathcal{C}$) and the communication time (denoted as $\mathcal{H}$) such that

$$T = \mathcal{C} + \mathcal{H} \tag{VI.1}$$

where $\mathcal{C} = \mathcal{C}_i + \mathcal{C}_{ii} + \mathcal{C}_{iii} + \mathcal{C}_{iv}$, and $\mathcal{H} = \mathcal{H}_i + \mathcal{H}_{ii} + \mathcal{H}_{iii} + \mathcal{H}_{iv}$, respectively. Note that the cost models only include dominating terms in the worse cases and thus ignore trivial cost.

TABLE 4: Comparison of $\overline{SE}$ and $RMSE$ between UP-FHDI, naive imputation, and GAIN with incomplete real-world datasets under MNAR. Note that SIS refers to the number of selected variables used in the sure independent screening.

Dataset	Method	Var. est	SIS	$\overline{SE}$	$RMSE$
Earthquake	Naive	×	×	0.0101	0.0543
	GAIN	×	×	×	0.0719
	UP-FHDI	Linearization	×	0.0092	0.0388
Bridge Strain	Naive	×	×	0.0673	0.1028
	GAIN	×	×	×	0.0916
	UP-FHDI	Linearization	×	0.0652	0.0335
Travel Time	Naive	×	×	0.4357	0.0104
	GAIN	×	×	×	0.0115
	UP-FHDI	Linearization	4	0.3943	0.0092
	UP-FHDI	Jackknife	4	0.3712	0.0092
CT Slices	Naive	×	×	0.0016	0.1409
	GAIN	×	×	×	0.2289
	UP-FHDI	Linearization	90	0.0015	0.0714
	UP-FHDI	Jackknife	90	0.0015	0.0714
Swarm	Naive	×	×	2.3130	0.1281
	UP-FHDI	Linearization	50	2.2482	0.0653
	UP-FHDI	Jackknife	50	2.1638	0.0653
p53	Naive	×	×	0.0082	0.0294
	UP-FHDI	Linearization	15	0.0077	0.0212
	UP-FHDI	Jackknife	15	0.0079	0.0212
Radar	Naive	×	×	2.3089	0.0701
	UP-FHDI	Linearization	30	2.3104	0.0386
	UP-FHDI	Jackknife	30	2.3103	0.0386

(i) **Cell construction:** In Algorithm 2 for the categorization of imputation cells, the contiguous reading in line 2 and non-contiguous writing in line 11 take the majority of the running time such that communication cost (denoted as $\mathcal{H}_i^{(1)}$) is given by

$$\mathcal{H}_i^{(1)} = \frac{np}{Q-1}(2\gamma_1 + \gamma_2) + (Q-1)(2\mathcal{L}_1 + \mathcal{L}_2) \tag{VI.2}$$

In Algorithm 3 for the extraction of imputation patterns, the contiguous reading in lines 2 and 4 and contiguous writing in lines 8 and 9 (denoted as $\mathcal{H}_i^{(2)}$) dominate the IO communication cost by

$$\mathcal{H}_i^{(2)} = \frac{(Q+1)np}{Q-1}\gamma_1 + 3(Q-1)\mathcal{L}_1 \tag{VI.3}$$

The major computational cost in line 5 (denoted as $\mathcal{C}_i^{(2)}$) is given by

$$\mathcal{C}_i^{(2)} = \frac{n^2}{Q-1}\alpha \tag{VI.4}$$

In Algorithm 4 for the generation of the ranking list, this function will be skipped if $v = 0$. Otherwise, the computational cost in line 6 (denoted as $\mathcal{C}_i^{(3)}$) dominates by

$$\mathcal{C}_i^{(3)} = \frac{p^2 n + p n^2}{Q-1}\alpha \tag{VI.5}$$

And communication cost in lines 14 and 15 (denoted as $\mathcal{H}_i^{(3)}$) is given by

$$\mathcal{H}_i^{(3)} = \frac{Qpt}{Q-1}\beta + 2(Q-1)\mathcal{L} \tag{VI.6}$$

In Algorithm 5 for the determination of donors, if $v = 0$, the primary computational cost in lines 7 and 8 (denoted as $\mathcal{C}_i^{(4)}$) is

$$\mathcal{C}_i^{(4)} = \frac{n^2 p}{Q-1}\alpha \tag{VI.7}$$

Otherwise, the prime computational cost in lines 10 to 12 is

$$\mathcal{C}_i^{(4)} = \frac{np^2 t + n^2 v}{Q-1}\alpha \tag{VI.8}$$

In Algorithm 6 to determine deficient donors, the cost will be trivial if $v \neq 0$. Otherwise, the major computational cost in line 7 (denoted as $\mathcal{C}_i^{(5)}$) is

$$\mathcal{C}_i^{(5)} = \frac{2n^2p}{Q-1}\alpha \tag{VI.9}$$

By summing up the major cost in each parallel function of the cell construction, we have

$$\mathcal{C}_i = \begin{cases} \frac{3s^3+s^2}{Q-1}\alpha & \text{if } v = 0 \\ \frac{(t+2)s^3+(1+v)s^2}{Q-1}\alpha & \text{if } v \neq 0 \end{cases} \tag{VI.10}$$

and

$$\mathcal{H}_i = \begin{cases} \frac{(Q+3)s^2}{Q-1}\gamma_1 + \frac{s^2}{Q-1}\gamma_2 + (5Q-5)\mathcal{L}_1 + (Q-1)\mathcal{L}_2 & \text{if } v = 0 \\ \frac{(Q+3)s^2}{Q-1}\gamma_1 + \frac{s^2}{Q-1}\gamma_2 + \frac{Qst}{Q-1}\beta + (5Q-5)\mathcal{L}_1 + (Q-1)\mathcal{L}_2 + (2Q-2)\mathcal{L} & \text{if } v \neq 0 \end{cases} \tag{VI.11}$$

(ii) **Estimation of cell probability:** The cost in this stage is negligible in T and thus it is unnecessary to present the cost details.

(iii) **Imputation:** The computational cost in this stage is trivial. Considering the worst scenario with M donors for each recipient, the primary IO communication cost in line 10 of Algorithm 7 is given by

$$\mathcal{H}_{iii} = \frac{npM}{Q-1}\gamma_2 + (Q-1)\mathcal{L}_2 \tag{VI.12}$$

(iv) **Variance estimation:** Likewise, the computational cost in this stage is trivial in T. The major IO communication cost in lines 2 to 5 of Algorithm 8 turns out to be

$$\mathcal{H}_{iv} = \frac{2np}{Q-1}\gamma_1 + \frac{2np}{Q-1}\gamma_2 + 2(Q-1)(\mathcal{L}_1 + \mathcal{L}_2) \tag{VI.13}$$

By substituting $\mathcal{C}$ and $\mathcal{H}$ of each stage of UP-FHDI derived above, we can simplify total execution time $T(Q)$ in the worst scenario as

$$\begin{aligned} T(Q) &= \mathcal{C}_i + \mathcal{H}_i + \mathcal{H}_{iii} + \mathcal{H}_{iv} \\ &\approx \frac{\alpha^{'}}{Q} + \beta^{'}Q \end{aligned} \tag{VI.14}$$

where $\alpha^{'} = O(s^3)\alpha + O(s^2)\gamma_2$ and $\beta^{'} = O(\mathcal{L}_2)$. By substituting Eq. (VI.14), the scalability of $\frac{T(Q)}{T(c_pQ)}$ is expressed by

$$\frac{T(Q)}{T(c_pQ)} = c_p \times \frac{\alpha^{'} + Q^2\beta^{'}}{\alpha^{'} + c_p^2Q^2\beta^{'}} \tag{VI.15}$$

where $c_p \in \mathbb{N}^+$.

APPENDIX VII: Example of Parallel Filter Method Based on Mutual Information

This section provides instructions on running the parallel filter method based on mutual information (MI). An easy-to-use example is available in [10]. This program requires two input files, including initial setups (summarized in Table 5) and a binary dataset. i_col_target is the index of the target variable and indices of all variables are 1-indexed. $i\_selection$ represents the total number of selected features. Noticeably, the data distribution in the binary dataset must be column-oriented. To compile the program on the Stampede2 using the default Intel compiler, run the command:

```
mpicxx -o main_MPI main_MPI.cpp
```

Afterward, submit a job script into the queue by issuing:

```
sbatch run.sbatch
```

An example job script for parallel MI is shown below:

```
#!/bin/bash
#SBATCH -J P-MI           # Job name
#SBATCH -p normal         # Queue name
#SBATCH -N 1              # Total number of nodes
#SBATCH -n 6              # Total number of MPI tasks
#SBATCH -t 00:30:00       # Max run time (hh:mm:ss)
module load intel/18.0.2 # Load the Intel compiler
ibrun ./main_MPI          # Launch MPI code
```

The parallel MI has two output binary files summarized in Table 6.

TABLE 5: Initial settings of parallel MI.

• $nrow$: The number of instances of input data
• $ncol$: The number of variables of input data
• $category$: The number of categories for all variables
• i_col_target: The index of the target variable
• $i\_selection$: The number of selected features

TABLE 6: Descriptions of output files of parallel MI.

Output files	Description
$selected\_daty.txt$	Selected features
$selected\_id.txt$	Indices of selected features

APPENDIX VIII: Example of Graphical Lasso

This section illustrates the use of graphical lasso by the R package $glasso$. The example aims to further extract important features from selected features by parallel MI. Suppose we store selected features by parallel MI as a matrix in R (denoted as $selected\_daty$). Compute covariance matrix S by:

```
S <- cov(selected_daty)
```

and apply the glasso to estimate a sparse inverse covariance matrix P by:

```
a <- glasso(S, 0.6)
P <- a$wi
```

where the regularization parameter for graphical lasso is set to be 0.6. A larger value of the regularization parameter gives a more spare model. Construct a graphic model g regarding the estimated precision matrix P by

```
A <- ifelse(P!=0 & row(P)!=col(P),1,0)
g <- network(A)
```

One can extract important features connected to the target variable in the graphic model g. A toy dataset and simple R code are available in [10].

References

[1] TACC, "Texas advanced computing center (tacc) at the university of texas at austin," 2017. [Online]. Available: http://www.tacc.utexas.edu

[2] Condo, "Condo2017: Iowa state university high-performance computing cluster system," 2017. [Online]. Available: https://www.hpc.iastate.edu/guides/condo-2017

[3] I. Cho, "Data-driven, computational science and engineering platforms repository," 2017. [Online]. Available: https://sites.google.com/site/ichoddcse2017/home/type-of-trainings/ultra-data-oriented-parallel-fhdi-up-fhdi

[4] I. Song, Y. Yang, J. Im, T. Tong, C. Halil, and I. Cho, "Impacts of fractional hot-deck imputation on learning and prediction of engineering data," *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[5] Stampede2, "Stampede2 user guide," 2017. [Online]. Available: https://portal.tacc.utexas.edu/user-guides/stampede2

[6] Y. Yang, J. K. Kim, and I. H. Cho, "Incomplete datasets for up-fhdi," *IEEE DataPort*, 2021, https://ieee-dataport.org/open-access/incomplete-datasets-fhdi.

[7] J. Yoon, J. Jordon, and M. van der Schaar, "Gain: Missing data imputation using generative adversarial nets," *35th International Conference on Machine Learning*, pp. 5689–5698, 2018.

[8] S. C.-X. Li, B. Jiang, and B. M. Marlin, "Misgan: Learning from incomplete data with generative adversarial networks," *arXiv*, 2019.

[9] J. Yoon, J. Jordon, and M. van der Schaar, "Codebase for generative adversarial imputation networks (gain)," *GitHub*, 2018, https://github.com/jsyoon0823/GAIN.

[10] Y. Yang, J. K. Kim, and I. H. Cho, "Codebase for two-staged feature selection," *Cybox*, 2021, https://iastate.app.box.com/s/tcl8u50mlk0hh270z2cgscdr3z17azfu.