

Private Data Imputation

Abdelkarim Kati
University of Waterloo
Email: akati@uwaterloo.ca

Florian Kerschbaum
University of Waterloo
Email: florian.kerschbaum@uwaterloo.ca

Marina Blanton
University at Buffalo
Email: mblanton@buffalo.edu

Abstract—Data imputation is an important data preparation task where the data analyst replaces missing or erroneous values to increase the expected accuracy of downstream analyses. The accuracy improvement of data imputation extends to private data analyses across distributed databases. However, existing data imputation methods violate the privacy of the data rendering the privacy protection in the downstream analyses obsolete. We conclude that private data analysis requires private data imputation.

In this paper, we present the first optimized protocols for private data imputation. We consider the case of horizontally and vertically split data sets. Our optimization aims to reduce most of the computation to private set intersection (or at least oblivious programmable pseudo-random function) protocols which can be very efficiently computed. We show that private data imputation has – on average across all evaluated datasets – an accuracy advantage of 20% in case of vertically split data and 5% in case of horizontally split data over imputing data locally. In case of the worst data split we observed that imputing using our method resulted in an increase of up to 32.7 times in the quality of imputation over the vertically split data and 3.4 times in case of horizontally split data. Our protocols are very efficient and run in 2.4 seconds in case of vertically split data and 8.4 seconds in case of horizontally split data for 100,000 records evaluated in the 10 Gbps network setting, performing one data imputation.

1. Introduction

Consider two hospitals seeking to improve patient diagnosis through collaborative analytics. Hospital A has 50,000 patient records; Hospital B has 30,000. When Hospital B conducts research, incomplete records would benefit from imputation using similar cases from Hospital A’s larger dataset. Yet privacy regulations like HIPAA prohibit sharing raw patient records. This creates a dilemma: imputation accuracy improves with more data, yet privacy requirements prevent data sharing. This challenge extends beyond health-care to banks, research consortia, and any domain where distributed parties hold similar information about different entities.

Data cleaning [17] is a necessary step before any data analysis. This applies even to private analyses such as those supported by secure multi-party computation or private

set intersection (PSI) [37]. However, existing data cleaning methods operate on a single database, which prevents their use in privacy-preserving data analysis over databases distributed across mutually distrusting parties. Blass and Kerschbaum [5] have studied the problem of detecting errors across distributed databases. Their solution reduces to PSI which makes it very efficient. However, correcting errors is still an open problem.

Data imputation helps to correct errors. It is the task of replacing non-existent data entries with the most likely data value. It is well known among statisticians that analysis over an imputed dataset leads to better models than ignoring records with missing entries [33]. However, existing data imputation algorithms operate on a single database. Operating over multiple databases increases accuracy of the imputation and hence accuracy of any downstream analysis task. In our evaluation we show that, on average, imputing over horizontally split databases (where parties hold different records with the same attributes) improves accuracy by 20%, while vertically split databases (where parties hold different attributes for the same records) yield a 5% improvement for the most commonly evaluated type of missing data (MCAR – missing completely at random), even when considering all our performance-enhancing modifications. In worst-case data splits, our methods achieve improvements of up to 32.7× for vertical splits and 3.4× for horizontal splits. These substantial accuracy gains demonstrate that distributed imputation is essential for high-quality data analysis.

However, privately imputing over multiple databases is non-trivial. While there exist privacy-preserving training algorithms over distributed data using secure multi-party computation, e.g., [28], [40] or federated learning, e.g., [24], these require already imputed data. Yet, when exchanging the data in the clear for the purpose of data imputation, privacy of the entire training dataset gets violated, undermining the privacy guarantees of downstream privacy-preserving analyses. Hence, we need private data imputation to complete the data science process from start to end before we can engage in privacy-preserving training of machine learning models. We study this problem in this paper.

Secure multi-party computation [14] can certainly be used to implement private data imputation. However, in current data analysis problems we often deal with millions, if not billions, of data records. Generic multi-party computation is not likely to scale to such problem sizes and

hence we need to employ optimizations. One optimization we share with Blass and Kerschbaum [5] is that we make extensive use of PSI which is a very fast privacy-enhancing technology. However, data imputation is necessarily more complicated than just detecting errors as in [5] and we need to design more complicated protocols while preserving privacy.

Jagannathan and Wright [18], [19] have already studied data imputation over horizontally partitioned databases with privacy. They infer the missing value using a decision tree. However, a recent benchmark study [23] on data imputation recommends other data imputation algorithms with higher accuracy. The benchmark demonstrates that simple methods like k -nearest neighbor (k -NN) and random forest consistently outperform complex deep learning approaches. Critically, when trained on incomplete data—the scenario our protocols address—ML methods’ advantages largely disappear while k -NN maintains robust performance. Furthermore, federated learning approaches would require multiple training rounds with secure aggregation plus expensive privacy-preserving inference over encrypted or secret-shared data for each imputation query. The benchmark shows ML training times often exceed simple methods by 10–100 $\times$, while adding privacy-preserving neural network inference would increase costs by another 100–1000 $\times$ compared to our PSI-based approach. Even with these substantial computational costs, federated ML approaches would deliver inferior imputation quality according to comprehensive benchmark evidence (detailed discussion in Section 3).

In this paper, we study the problem of *private data imputation with high accuracy*. We assume two parties, Alice and Bob, each with their own private database. Bob has a missing data value and it should be imputed. We consider both horizontally and vertically split data.

We solve the problem using a secure multi-party computation which uses PSI to perform the most time-consuming operations and hence our protocol is very efficient. A recent benchmark study [23] recommends nearest-neighbor approaches as top performers with low computational complexity. Unlike traditional k -NN which selects exactly k neighbors regardless of their distance, we use a radius-based nearest-neighbor algorithm that considers all tuples within a learned distance threshold r_j for each attribute j .

Our Contributions. In summary, our private data imputation protocol provides the first practical solution to private data imputation across distributed datasets. Our contributions are:

- We present the first private data imputation protocols for horizontally and vertically split data without prerequisite learning.
- We show that our private data imputation protocols have higher imputation accuracy than local data imputation on a single database *even when using performance-enhancing modifications such as quantization*.
- We show that our private data imputation protocols are cryptographically secure in the semi-honest model with

clearly specified small leakage, such as which attribute is being imputed.

- We show that our private data imputation is efficient by imputing over 100,000 records with 10 attributes in less than 2.4 seconds in case of vertically split data, and 8.4 seconds in case of horizontally split data evaluated in the 10 Gbps network setting.

2. Related Work

Conventional data imputation and cleaning. Data imputation can be realized by a variety of different mechanisms from simple such as the mean (average) or mode (most frequent value) computation to more complex techniques based on deep learning (DL) such as discriminative DL (e.g., [22]) or generative DL (e.g., [26], [39]) imputation. However, according to [23], the k -nearest neighbor (k -NN) approach that samples a random value from the nearest neighbors provides superior accuracy compared to other solutions.

Privacy-preserving data imputation and cleaning. Privacy-preserving data imputation was first considered by Jagannathan and Wright [18], [19]. The work considered horizontally partitioned data between two parties and used a lazy decision tree imputation algorithm. Clifton et al. [9] follow a different approach and proposes a differentially private mechanism for multiple item data imputation. There are other publications (e.g., [29], [21]), but they invoke standard tools, do not show accuracy improvement, or do not provide sufficient treatment of security.

Privacy-preserving data cleaning is a related problem of identifying malformed or misclassified items or outliers. Publications in this space, e.g., [5] generally consider different techniques and are not suitable for data imputation. But more fundamentally, data cleaning is the first step of the process, while data correction, as we do in this work, is to be performed afterwards.

Sublinear lookup methods such as Private Information Retrieval (PIR) [8] and Oblivious RAM (ORAM) [16] offer asymptotically better amortized complexity for database queries when invoked a superlinear number of times than our linear approach of using PSI. Recent work on secure approximate nearest-neighbor search [6] achieves sublinear complexity using locality-sensitive hashing and ORAM. However, these approaches incur significant computational overhead in practice due to large constant factors even when ignoring the linear setup cost, and they are designed for high-dimensional vector spaces. Our setting differs fundamentally: we perform multi-attribute comparisons with quantized values across distributed datasets. PSI-based approaches, while linear in dataset size, benefit from highly optimized implementations [32] with small constant factors that remain practical for datasets up to several millions of records, particularly when invoked a constant or sublinear number of times. Furthermore, sublinear methods would require complex composition of multiple private queries for our multi-attribute neighbor computation adding additional complexity to the protocol.

3. Preliminaries

We first precisely define the problem we are addressing in this work. We then follow with defining the building blocks that our constructions utilize.

3.1. Problem Formulation

Let T be a relational table consisting of n tuples $t_i = \langle x_{i,1}, \dots, x_{i,m} \rangle$ for $i \in [1, n]$. Some items are missing, i.e., $x_{i,j} = \perp$, and the goal is to fill, or impute, the missing items with their approximate value.

We consider different splits of table T . When the table T is *horizontally split* between Alice and Bob, Alice has n_A tuples t_1 through t_{n_A} with all m attributes and Bob has n_B tuples t_{n_A+1} through $t_{n_A+n_B}$ with all m attributes, where $n = n_A + n_B$. When the table T is *vertically split* between Alice and Bob, Alice has m_A attributes A_1 through A_{m_A} and Bob has m_B attributes A_{m_A+1} through $A_{m_A+m_B}$ for each of n tuples, where $m = m_A + m_B$. Alice and Bob can link tuples using a common identifier, e.g., the index i with vertically split data.

Let $x_{\alpha,\beta}$ denote the element to be imputed, i.e., it is the β th attribute of tuple t_α . Without loss of generality, we let $x_{\alpha,\beta}$ reside with Bob. That is, in the case of horizontally split data, t_α is among Bob's tuples, while in the case of vertically split data, β th attribute is in Bob's dataset.

The horizontal and vertical data splitting scenarios we consider arise naturally in collaborative settings. Horizontal splits occur when organizations independently collect records about different entities (e.g., hospitals in different regions with identical medical attributes). Vertical splits emerge when parties collect complementary attributes about the same entities (e.g., a hospital with clinical measurements and an insurer with claims data). Combining information from multiple parties substantially improves imputation accuracy. Our protocols enable such collaboration without revealing sensitive data, providing formal privacy guarantees throughout the computation.

We base our approach on Jäger et al. [23], which evaluated six imputation methods across 69 datasets with three missingness patterns (missing completely at random MCAR, missing at random MAR, and missing not at random MNAR) and missingness fractions from the range 1–50%. For numerical columns, k -NN ranks in the top three methods in 75% of cases. For MCAR and MAR patterns with 30–50% missingness, ML-based methods provide less than 1% improvement for classification and 0.5% for regression when training incomplete data [23], while k -NN maintains robust performance. While ML models offer reusability, this is outweighed by k -NN's superior accuracy on incomplete data and compatibility with efficient cryptographic protocols. Our radius-based variant (Section 4) requires no training and naturally integrates with privacy-preserving tools like PSI and OPPrF.

In order to replace $x_{\alpha,\beta}$ with its approximate value, we start with the approach that was determined to be among the best in [23]: Determine the set I of nearest neighbors of t_α .

This was accomplished in [23] using a k -nearest neighbor algorithm that computes k closest tuples to t_α .

The value to replace the missing item $x_{\alpha,\beta}$ with is determined from the neighboring records as follows:

- For *categorical* data: Uniformly select one of the tuples from I and use its β th attribute to impute $x_{\alpha,\beta}$. Note that the expected value of the selection is the mode, i.e., the most frequent element.
- For *numerical* data: Compute the mean of the β th attribute values in I and use it to impute $x_{\alpha,\beta}$.

As far as data privacy goes, in the ideal case, Alice does not learn any information as a result of the computation and Bob learns only the imputed value $x_{\alpha,\beta}$. Note that the output itself discloses some information about Alice's dataset to Bob. For example, when the data is horizontally split, if the computed value of $x_{\alpha,\beta}$ does not appear as the β th attribute among Bob's tuples, Bob learns one of the Alice's values for that attribute.

We consider standard definitions of secure two-party computation that require that information about private inputs is not disclosed during the computation. We treat the parties as semi-honest adversaries, i.e., those that do not deviate from the computation but might try to deduce unauthorized information from their view during protocol execution. Security is normally shown via a simulation paradigm that allows one to simulate the adversary's view without access to the other party's private information and show that the simulated view is indistinguishable from the real view during the protocol execution.

3.2. Building Blocks

Oblivious Programmable Pseudo-Random Functions (OPPrF). An OPPrF is a secure, cryptographic protocol between two participants, Alice and Bob. Alice has as input a value x and Bob has as input a set of pairs of values $\mathbb{Y} = \{(y_i, r_i)\}$ and a key k for a pseudo-random function. As output of the protocol, Alice learns a value r and Bob learns no new information, i.e., has no output. It holds that if $(x, s) \in \mathbb{Y}$, then $r = s$ and if $(x, \cdot) \notin \mathbb{Y}$, then r is pseudo-random, i.e., computationally indistinguishable from a uniformly drawn random number. In a practical protocol instance, Alice's output is usually implemented as a keyed (punctured) pseudo-random function on Alice's value x with Bob's key k . Note that Alice does not learn whether $(x, \cdot) \in \mathbb{Y}$, i.e., Bob should choose its values r_i also randomly.

In our implementation, we use VOLE-OPPrF by Rindal and Schoppmann [32]. Each r_i is from the space $\{0, 1\}^\kappa$, where κ is the security parameter. While x and y_i s can come from a different space, in the implementation on which we rely these values are also cast to be from the same space. As the bitlength of the numbers is a function of a security parameter, we use $O(\kappa)$ to denote the cost of a single cryptographic operation. Then this construction has a communication and computation cost of $O(n\kappa)$ for building the OPPrF on a set of size n and performing n OPPrF evaluations.

Hashing and Cuckoo Hashing. We use a regular cryptographic hash function H that hashes inputs of arbitrary size into fixed-length digests $H : \{0, 1\}^* \rightarrow \{0, 1\}^k$ or into another suitable output space such as $H : \{0, 1\}^* \rightarrow \mathbb{G}$ for a group $\mathbb{G}$ defined above.

In addition, we rely on Cuckoo hashing that uses h ordinary hash functions $H_1, \dots, H_h$ with μ output bins, i.e., each $H_i : \{0, 1\}^* \rightarrow [\mu]$, where notation $[x]$ denotes the set $\{1, \dots, x\}$. To insert an element x into a Cuckoo hash, we compute $H_1(x), \dots, H_h(x)$, which are the indices of the candidate bins. If there is an empty bin with index $H_i(x)$ for some $i \in [h]$, we place x into that bin. Otherwise, we evict an element y from one of the candidate bins, place x in that bin, and re-insert y using the same process.

Given the size of the set $|\mathbb{X}|$ we intend to insert and statistical security parameter λ , one can compute parameters h and μ so that the number of bins is linear in the set size $\mu = O(|\mathbb{X}|)$ and with an overwhelming probability $1 - 2^{-\lambda}$ there is an allocation with every bin containing at most one element.

Private Set Intersection (PSI). PSI is a well-studied functionality (see, e.g., [12], [27], [10], [20], [4], [32]) that allows two participants, Alice and Bob, to compute the intersection of their private input sets without disclosing any information about their sets besides the set cardinality (or an upper bound on the set cardinality). That is, on input $\mathbb{X} = \{x_i\}$ from Alice and $\mathbb{Y} = \{y_i\}$ from Bob, one or both parties learn $\mathbb{X} \cap \mathbb{Y}$.

VOLE-PSI. VOLE-PSI [32] is a state-of-the-art PSI protocol. It uses an OPRF based on Vector-Oblivious Linear Evaluation (VOLE), an extension of the PaXoS data structure [30]. VOLE-PSI scales optimally with a constant number of public-key operations and linearly many symmetric key operations in the number of set elements.

VOLE-PSI can be used as a one-sided PSI protocol, where Bob computes the OPRF of his elements and Alice sends the PRFs of her elements for computing the intersection. Moreover, VOLE-PSI can also be used as a circuit-PSI protocol, where neither party learns the intersection and the parties can proceed with further computation on the private intersection. In that case, Bob learns the OPRF of his values and Alice retains one corresponding PRF which matches Bob's if and only if she has the value in his set.

This construction of circuit-PSI is based on previous work [31], [25] where the parties first place their elements in the bins, but we use a different bin placement strategy per party. In particular, Bob hashes his elements into the bins using Cuckoo hashing and pads all empty bins to one element. Alice hashes her elements using all Cuckoo hashing functions, i.e., she replicates the elements if necessary, and pads the size of each bin to a maximum that is only exceeded with negligible probability. Pinkas et al. [31] show how to compute this maximum number from the number of elements. Alice programs the VOLE-OPRF to the same random value for each of the elements in her bin, such that if Bob's element is in Alice's bin, they share the same random value. After such a circuit-PSI, Alice and Bob can run a two-

party computation in which they compare the PRFs and then perform any computation over the set intersection.

Secure Two-Party Computation. We rely on general-purpose secure computation techniques for components of our solutions. There are a variety of techniques that differ in the number of supported computation participants, participant collusion threshold, and the threat model [14]. Of interest to this work is secure two-party computation with semi-honest participants. The security expectations are such that as a result of computing on private inputs the parties do not learn any information about the private data they handle other than the agreed-upon computation outcome.

We choose ABY [11] for our implementation due to its versatility and the ability to combine different data representations and techniques. The operations we execute using ABY include equality, less-than comparisons, joint sampling of a random value, addition, multiplication, and multiplexor operations to realize branching with private conditions.

4. MPC-friendly Nearest Neighbor Computation

Let $t_\alpha \sim t_\omega$ denote a neighboring relation between tuples t_α and t_ω , where we seek to impute the β th attribute of t_α , $x_{\alpha,\beta}$. We say tuples t_α and t_ω are neighbors if all of their non-empty attributes are within a pre-defined distance r_j from each other. That is,

$$t_\alpha \sim t_\omega \text{ iff } \forall j \in \{1, \dots, m\} \setminus \{\beta\}, \\ x_{\alpha,j} = \perp \vee x_{\omega,j} = \perp \vee |x_{\alpha,j} - x_{\omega,j}| \leq r_j.$$

Because our goal is to perform imputation using sampling from neighboring records, only tuples with a non-empty β attribute are useful for our purpose. For that reason, we redefine the neighboring relation between t_α and any other tuple t_ω to require that $x_{\omega,\beta} \neq \perp$. That is,

$$t_\alpha \sim t_\omega \text{ iff } (x_{\omega,\beta} \neq \perp) \wedge (\forall j \in \{1, \dots, m\} \setminus \{\beta\}, \\ x_{\alpha,j} = \perp \vee x_{\omega,j} = \perp \vee |x_{\alpha,j} - x_{\omega,j}| \leq r_j).$$

In the rest of this work, unless mentioned otherwise, we will use this modified relation to denote neighboring relation. The optimal values for the radius parameters r_j are determined through an adaptive search process during the training phase, which we describe in detail in Section 9.1.

Evaluating the above constraints without leaking intermediate results on all tuples requires $O(nm)$ costly less-than comparisons, which we aim to avoid. We therefore reduce less-than comparisons to equality comparisons using quantization. We quantize the value of the j th attribute in each tuple as follows: The first quantization places values into ranges $[0, r_j - 1], [r_j, 2r_j - 1], [2r_j, 3r_j - 1], \dots$ and the second quantization places values into ranges $[0, r_j/2 - 1], [r_j/2, 3/2r_j - 1], [3/2r_j, 5/2r_j - 1], \dots$. We then represent each $x_{\alpha,j}$ (and $x_{\omega,j}$, respectively) as its two

PARAMETERS: There are n tuples consisting of m attributes horizontally or vertically partitioned between Alice and Bob.

FUNCTIONALITY:

- Receive private records from Alice.
- Receive private records as well indices α and β from Bob.
- Determine the neighbors of t_α and create an indicator bit vector of size $n - 1$, where the value in the position $\omega (\neq \alpha)$ is set to 1 if and only if t_ω is a neighbor of t_α and $x_{\omega,\beta}$ is not missing.
- Create two shares of each bit of the indicator vector and give one share to Alice and the other share to Bob.

Figure 1: Ideal functionality for neighbor computation.

ranges which we denote as $q_1(x_{\alpha,j})$ and $q_2(x_{\alpha,j})$. Now, the following relations hold:

$$|x_{\alpha,j} - x_{\omega,j}| \leq \frac{r_j}{2} \\ \implies q_1(x_{\alpha,j}) = q_1(x_{\omega,j}) \vee q_2(x_{\alpha,j}) = q_2(x_{\omega,j})$$

$$q_1(x_{\alpha,j}) = q_1(x_{\omega,j}) \vee q_2(x_{\alpha,j}) = q_2(x_{\omega,j}) \\ \implies |x_{\alpha,j} - x_{\omega,j}| \leq \frac{3}{2}r_j$$

We therefore replace testing for $|x_{\alpha,j} - x_{\omega,j}| \leq r_j$ with $q_1(x_{\alpha,j}) = q_1(x_{\omega,j}) \vee q_2(x_{\alpha,j}) = q_2(x_{\omega,j})$.

Our distance metric not only uses only equality comparisons instead of less-than comparisons, it is also composable across attributes. This means, in the vertical setting, that Alice can compute the distance to her attributes and Bob's to his. In this paper, we will show how to efficiently implement this algorithm as a protocol over distributed parties with privacy while preserving the accuracy of the imputation.

5. Solution Structure

We are interested in developing solutions for different types of data partitioning (such as horizontally and vertically split data) and different types of attributes (such as containing categorical and numerical values) that after optimizations result in significantly different protocols. However, we put forward a general computation structure that accommodates different types of data partitioning as well as different ways of determining the imputation value.

Recall that the goal is to compute the imputation value for the item $x_{\alpha,\beta}$, where α defines the tuple t_α and β is the attribute of the tuple. Conceptually, the computation consists of two main components: (i) privately determining the neighbors of the tuple t_α in the partitioned dataset and (ii) privately analyzing the β th attribute of the neighbor records to determining the imputation value for $x_{\alpha,\beta}$. Without loss of generality, let the item $x_{\alpha,\beta}$ be located among Bob's records.

PARAMETERS: There are n tuples consisting of m attributes horizontally or vertically partitioned between Alice and Bob.

FUNCTIONALITY:

- Receive private records and a share of the indicator vector v from Alice.
- Receive private records, indices α and β , and a share of the indicator vector v from Bob.
- Extract the β th attribute from all tuples t_ω where $\omega \neq \alpha$.
- Conditionally use each extracted value $x_{\omega,\beta}$ in the computation of the imputed value using the corresponding flag v_ω from the indicator vector v :
 - for numerical data: $x_{\alpha,\beta} = (\sum_{\omega, v_\omega=1} x_{\omega,\beta}) / (\sum_{\omega} v_\omega)$
 - for categorical data: $x_{\alpha,\beta} \xleftarrow{R} \bigcup_{\omega} \{x_{\omega,\beta} \mid v_\omega = 1\}$
- Give the computed value $x_{\alpha,\beta}$ to Bob.

Figure 2: Ideal functionality for imputation value computation.

Figure 1 describes the ideal functionality of the first component, where Alice and Bob privately enter their data and Bob also supplies the indices α and β . As a result of this phase of the computation, Alice and Bob obtain a secret-shared indicator vector, where each tuple t_ω (except tuple t_α) is marked with a bit indicating whether t_ω is a neighbor of t_α according to the neighbor computation described in Section 4 and the value $x_{\omega,\beta}$ is not missing. By including only neighbor tuples with a non-empty value in the desired attribute, we are able to directly use those records in the computation, without having to additionally check if the desired attribute is present.

The ideal functionality for the second component is given in Figure 2. It uses the indicator vector to conditionally include the β th attribute of a tuple into the computation of the imputation values. Note that the computation for determining the imputation value significantly differs for numerical data (mean computation) and categorical data (random sampling). Notation $x \xleftarrow{R} X$ means drawing an element uniformly at random from the set X .

Our solutions introduce optimizations that fit within this general structure. For example, in the case of horizontally split data Bob can locally determine the neighbors of t_α among his tuples and interactive computation is only needed to compare Alice's tuples to Bob's t_α . In the case of vertically partitioned data, Alice and Bob can determine local neighbors among all records using partial attributes known to each if Alice gets the value of α from Bob. This reduces the size of the joint computation and consequently the indicator vector that the parties need to maintain.

6. Horizontally Split Data

In the horizontally split case, Bob has a tuple t_α with a missing value for attribute β – i.e., $x_{\alpha,\beta}$ has no value – which he would like to impute. Bob can find neighbors of t_α among his tuples using local computation, while Alice needs to identify t_α 's neighbors among her tuples using secure comparisons. Note that in the context of the ideal functionality in Figure 1, the parties need to determine the indicator vector only for Alice's tuples, as Bob can locally determine the neighbor status of his tuples (and enter them directly into the computation that follows).

6.1. Determining Neighbor Tuples

Let I_A and I_B be the set of neighbors among Alice's and Bob's tuples, respectively. As the first component of the solution, the parties need to jointly determine I_A .

We can determine which records among Alice's tuples have identical quantized ranges using OPPRF as follows: Bob locates a tuple t_α which has a missing value for attribute β , i.e., $x_{\alpha,\beta}$ has no value. We could compare Bob's t_α to each t_ω in Alice's dataset by comparing their quantized values for each attribute. However, in that case we would still need to perform a logical-OR operation between compared pairs to obtain the final result. Instead, we reduce the computation to one invocation of an OPPRF as described next.

For each attribute j , let $q'_1(x_{i,j})$ be the closest to item $x_{i,j}$'s quantized range in the first quantization method that is not the true range $q_1(x_{i,j})$. Similarly, let $q'_2(x_{i,j})$ be the closest range $\neq q_2(x_{i,j})$ using the second quantization method. For each attribute j and each tuple index $\omega \in [n_A]$, Bob draws a random value $\sigma_{\omega,j}$ and programs the OPPRF as follows:

- $\text{OPPRF}(q_1(x_{\alpha,j}) || q_2(x_{\alpha,j})) = \sigma_{\omega,j}$
- $\text{OPPRF}(q_1(x_{\alpha,j}) || q'_2(x_{\alpha,j})) = \sigma_{\omega,j}$
- $\text{OPPRF}(q'_1(x_{\alpha,j}) || q_2(x_{\alpha,j})) = \sigma_{\omega,j}$
- $\text{OPPRF}(\perp) = \sigma_{\omega,j}$

where $||$ denotes concatenation.

For each tuple t_ω in Alice's dataset, Alice uses both quantizations of each of her $x_{\omega,j}$ (for $j \neq \beta$) or $\perp$ (if $x_{\omega,j} = \perp$) to query the OPPRF, i.e., Alice privately computes with Bob:

- $\tau_{\omega,j} = \text{OPPRF}(\perp)$ if $x_{\omega,j} = \perp$
- $\tau_{\omega,j} = \text{OPPRF}(q_1(x_{\omega,j}) || q_2(x_{\omega,j}))$ otherwise.

At this point, it holds that

$$t_\alpha \sim t_\omega \iff \sum_{j \in \{1, \dots, m\} \setminus \{\beta\}} \sigma_{\omega,j} = \sum_{j \in \{1, \dots, m\} \setminus \{\beta\}} \tau_{\omega,j}$$

Note that Bob independently chooses a uniformly random $\sigma_{\omega,j}$ for each of the $n_A \cdot (m-1)$ invocations of the OPPRF, so that Alice cannot infer any information from observing multiple invocations. That is, even if Alice queries the OPPRF on the same inputs, the outputs she observes are independent of each other.

Also note that the solution, as described above, can handle missing values in Alice's dataset, but not in Bob's tuple t_α . In practice, in addition to missing $x_{\alpha,\beta}$ in t_α , Bob may have a missing value, $x_{\alpha,\gamma}$, for another attribute γ in tuple t_α . Thus, we need to address this before we can proceed. Our solution to mitigating the issue is to have Bob program the OPPRF at γ to the same $\sigma_{\omega,\gamma}$ for each possible value of $x_{\omega,\gamma}$ in Alice's dataset. This means that OPPRF evaluation on Alice's value $x_{\omega,\gamma}$ will always produce the expected $\sigma_{\omega,\gamma}$ regardless of its value $x_{\omega,\gamma}$.

After evaluating the OPPRF on Alice's tuples, the next step is to compute which of Alice's tuples are in I_A . A simple solution is for Bob to send her the expected $\sigma_\omega = \sum_{j \in [m] \setminus \{\beta\}} \sigma_{\omega,j}$ for each $\omega \in [n_A]$. This allows Alice to determine whether any given t_ω is a neighbor of Bob's tuple t_α and thus compute I_A . As we show below, this variant leaks an undesirable amount of information; however, we leave it for comparison purposes and denote it as *plain neighbor access*.

A secure way of computing I_A consists of Alice and Bob privately comparing the computed $\tau_\omega = \sum_{j \in [m] \setminus \{\beta\}} \tau_{\omega,j}$ and the expected $\sigma_\omega = \sum_{j \in [m] \setminus \{\beta\}} \sigma_{\omega,j}$ for equality for each of Alice's tuples $\omega \in [n_A]$. This variant results in a secret-shared indicator vector, and we call it *blind neighbor access*.

The next step is to use the results of neighbor computation to determine the imputation value.

6.2. Imputation Value for Numerical Data

In the case of numerical data, the imputation value is computed as the mean of the β th attribute among the neighboring records in $I_A \cup I_B$.

6.2.1. Plain Neighbor Access. Computing the imputation value from the set of neighbors $I_A \cup I_B$ is much easier if Alice and Bob know their respective sets of neighbors I_A and I_B . After Alice receives the expected σ_ω s, she determines I_A and Bob knows I_B .

Next, Alice and Bob will need to compute the sum of β th attribute of their tuples in I_A and I_B and divide the sum by $|I_A \cup I_B|$. To achieve that, Bob computes a partial sum s_B of the values in I_B , while Alice, respectively, computes s_A using her tuples I_A . The remaining computation that we need to perform privately is $(s_A + s_B) / (|I_A| + |I_B|)$, with Alice possessing s_A and $|I_A|$ and Bob possessing s_B and $|I_B|$. This can be very efficiently evaluated using existing secure two-party computation tools such as ABY.

6.2.2. Blind Neighbor Access. The above solution with plain neighbor access reveals the set I_A of neighbors to Alice. This allows Alice to conduct a simple inference attack on Bob's t_α by simply averaging her tuples in I_A and treating the result as an approximation of t_α . To prevent this attack, we should not reveal I_A to Alice.

We thus can represent each tuple t_ω in Alice's set by two values: a private flag which is set iff $t_\omega \in I_A$ (i.e., the indicator vector) and the value of the β th attribute $x_{\omega,\beta}$. This

will allow us to compute with Alice’s dataset by entering all n_A tuples into the computation and adding only the values from the tuples where the flag is set.

This conditional addition can easily be implemented using existing secure computation tools using a single pass through Alice’s tuples. That is, after the OPPrF evaluation in Section 6.1, Alice and Bob securely compare the expected σ_ω and computed τ_ω values for equality and add the item $x_{\omega,\beta}$ to Alice’s sum s_A if the values are equal. The remaining computation to determine the mean proceeds as specified in Section 6.2.1.

6.3. Imputation Value for Categorical Data

For categorical data, we randomly sample an element from $I_A \cup I_B$ and extract its β th element.

6.3.1. Plain Neighbor Access. In this variant, Alice knows the set of neighbors I_A among her records and Bob has neighbors I_B in this records. Next, Alice and Bob will need to securely sample one element from the union $I_A \cup I_B$ of their neighbors to determine the imputation value.

Let Alice randomly sample one of the records in I_A and extract its β th attribute, which we denote by v_A . (Note that by the definition of neighbor relationship, Alice will only consider the tuples with non-empty β th attribute and any tuple in I_A can be chosen as her sample.) Similarly, Bob randomly samples one of the records in I_B and extracts its β th attribute, denoted by v_B (similarly, it is guaranteed to be non-empty). The parties now need to select v_A or v_B in such a way that the sample is drawn uniformly at random from the union $I_A \cup I_B$.

To accomplish that, the parties could draw a random integer $R \in [0, |I_A \cup I_B| - 1]$ and securely compare it to $|I_A|$. If $R \geq |I_A|$, Bob replaces $x_{\alpha,\beta}$ with his value v_B ; otherwise, he learns Alice’s v_A and replaces $x_{\alpha,\beta}$ with v_A . More precisely, if c denotes the result of the private comparison $R \geq |I_A|$, the parties privately compute $c \cdot v_B + (1 - c) \cdot v_A = c \cdot (v_B - v_A) + v_A$ using a single multiplication.

A complication is that the parties do not know the sizes of each other’s set I_A or I_B to draw R from the desired range. Our solution is to draw a value from the fixed range $[0, 1)$ and scale it by a factor $|I_A| + |I_B|$ using a single multiplication. Individual sizes $|I_A|$ and $|I_B|$ can serve the purpose of additive shares of $|I_A| + |I_B|$, making it very efficient.

In our implementation, sampling random R from $[0, 1)$ is realized using p bits of precision, i.e., the parties draw from the set $[0, 2^p - 1]$. This is accomplished by Alice and Bob each entering p private bits into the secure computation and XORing them prior to scaling the result by $|I_A| + |I_B|$.

Next, notice that with this solution Bob learns the result of imputation v , which is either v_A or v_B and Bob knows v_B . This means that if the imputed value is $v = v_A$ while v_B differs, Bob knows that the sampled value came from Alice and thus Alice has v_A in her I_A records. If v_A also appears in Bob’s records I_B , this is the knowledge he would not be

able to deduce from the output alone. That is, if Bob learns that $v = v_A$ and v_A appears in his records, he should not know whether the value came from his or Alice’s records.

To eliminate this leakage, we modify the solution so that Bob does not learn v_B sampled from his records. In the scenario above, this would prevent him from knowing that v_A originated from Alice’s records. To achieve this, we have Bob enter all of the β attributes from the records in I_B into secure computation, one of which is sampled privately. Specifically, we draw a random value in the range $[0, |I_B| - 1]$ inside secure two-party computation and scan through Bob’s input to obliviously select the element at that index. The remaining computation proceeds as before.

6.3.2. Blind Neighbor Access. This time, to prevent inference attacks, Alice does not learn which of her tuples are in I_A . Rather, Alice and Bob securely compare their σ_ω and τ_ω sums for each of Alice’s tuples t_ω and secret share a private flag that indicates whether the tuple is in I_A .

Let there be ℓ neighbors in a set of n_A tuples. If $\ell \ll n_A$ and Alice samples blindly from her set of tuples, the probability that she samples a neighbor is low. This means that many samples are needed to ensure that she samples a neighbor at least once. To overcome this, our solution is to combine c tuples together into a single value by adding their neighbor flags and the β th attribute’s values. If the combined neighbor count is exactly 1, then we can use the corresponding value v_A that contains the sum of the attribute values in the sampling procedure above.

We can determine the optimal value of c as follows: If the probability that each tuple is a neighbor is $\frac{\ell}{n_A}$ and the probabilities are independent of each other, the probability that exactly one flag will be set among c tuples is

$$q(c) = c \cdot \frac{\ell}{n_A} \left(1 - \frac{\ell}{n_A}\right)^{c-1}.$$

That is, there are c disjoint combinations for one of the draws to be a neighbor (with probability $\frac{\ell}{n_A}$) and $c - 1$ other draws to be not a neighbor (with probability $1 - \frac{\ell}{n_A}$ each). The value of c that maximizes the probability function $q(c)$ is $c = \frac{-1}{\ln(1 - \frac{\ell}{n_A})}$ and we set it based on the expected number of neighbors among Alice’s tuples.

The next step is to pack the tuples into enough slots – each with c tuples – so that the probability that at least one of the slots will have exactly one neighboring tuple is above a certain threshold $1 - \epsilon$. Given the desired bound on the probability of failure ϵ , the number of sufficient slots d is determined as the lowest d that satisfies

$$(1 - q(c))^d \leq \epsilon.$$

When $d \cdot c \geq n_A$, a single tuple can be packed more than once, but the work to compute their neighbor status is bounded by n_A . When $d \cdot c < n_A$, on the other hand, we only need to evaluate the OPPrF and compute the neighbor flags for the $d \cdot c$ tuples chosen to be packed.

That is, Bob and Alice first compute c and d based on an estimated number of neighbors in Alice’s dataset. If $d \cdot c \geq$

n_A , Alice and Bob compute OPPRF values σ_ω and τ_ω for $1 \leq \omega \leq n_A$. Otherwise, they compute these values only for the $d \cdot c$ tuples selected for this purpose. Comparison of the OPPRF evaluations and all consecutive steps are performed using secure computation. The parties then compute packed representation of d samples with c tuples each. To achieve random selection, tuples need to be selected and packed in a randomized order. Each of the d slots now contains the sum of c neighbor flags (the count) together with the sum of the β th attributes of the corresponding tuples.

Sampling can be performed by scanning the slots and obviously grabbing the value in the first slot with its count being 1. The chosen value will be a valid neighbor with probability at least $1 - \varepsilon$. That is, there is no valid sample across all slots with probability at most ε . (Sample validity can easily be checked and the computation aborted otherwise). The rest proceeds as described in Section 6.3.1 by drawing a random sample from Bob's neighbors and selecting either Alice's or Bob's sample.

Choosing between Alice's and Bob's sample requires using $|I_A|$ in the computation. However, if $d \cdot c < n_A$ and we do not perform OPPRF evaluation for all n_A Alice's tuples, we cannot compute $|I_A|$. In that case $|I_A|$ is approximated as $\frac{n_A}{d \cdot c} |I'_A|$, where $|I'_A|$ is the number of neighbors among the evaluated $d \cdot c$ tuples.

7. Vertically Split Data

In vertically split data, Bob also locates a tuple t_α which has a missing value for attribute β . This time, each party has a partial set of attributes and neighbor computation has a completely different structure.

7.1. Determining Neighbor Tuples

This time, Bob can determine t_α 's neighbors in this dataset by local comparisons using the attributes he possesses. This, however, is a superset of the actual neighbors, since he does not have access to Alice's attributes. Similarly, Alice can determine her neighbors (assuming she knows α) which are also a superset of the actual neighbors. Let I_A and I_B be the sets of row indices of Alice's and Bob's local neighbors, respectively.

Note that because the neighbor tuples are intended for imputation, the parties need to select only those tuples as neighbors where the β th attribute is not empty. With vertically split data, this attribute resides with Bob and thus he selects only those neighbors where the β th attribute is not missing.

In the context of the ideal functionality of Figure 1, we do not need to maintain the indicator vector for all tuples, as the parties can locally filter out some of the tuples as not being relevant. In particular, the tuples of interest are only those in I_A and I_B as potential neighbors, while the remaining tuples are automatically removed from consideration as non-matching.

The set of global neighbors can be determined as $I_A \cap I_B$. We therefore have Alice and Bob engage in a private set

intersection (PSI). To protect the number of local neighbors, the parties can pad their inputs by a constant fraction of the dataset size to remove any leakage about the number of local neighbors.

A simpler version involves the parties engaging in a conventional PSI, with the result being revealed to Bob. This undeniably discloses additional information to Bob beyond the indented output and we refer to it as plain neighbor access. A version that properly protects the data, blind neighbor access, involves the parties engaging in circuit-PSI that produces a secret-shared vector of bits, i.e., the indicator vector. That version will allow Alice and Bob to securely compute an imputation value using global neighbors.

7.2. Imputation Value for Numerical Data

7.2.1. Plain Neighbor Access. Computing the imputation value is easy with access to the neighboring tuples. That is, after Bob obtains the set intersection $I_A \cap I_B$, he can simply compute the mean of the attribute values in the set intersection since he has access to all values for attribute β .

7.2.2. Blind Neighbor Access. To reduce the information leakage above, the parties utilize circuit-PSI to obtain secret-shared output. We can use VOLE-PSI as our circuit-PSI protocol, which relies on Cuckoo and standard hashing as was described in Section 3.2. In that case, Alice and Bob each obtain a vector of random values as the PSI output. These random values are equal if Bob's element is in Alice's set and otherwise they are different. After comparing those random values using a two-party computation (ABY in our implementation), Alice and Bob hold secret shares of the indicator vector of neighboring elements.

What remains is to securely scan through the vector adding the β th attributes of the tuples in the intersection, after which the sum is divided by the size of the intersection. This is performed using secure two-party computation.

7.3. Imputation Value for Categorical Data

This time instead of computing the mean we sample one value from the intersection at random. The bulk of the computation remains the same as in Section 7.2 and we need to modify only the last step.

7.3.1. Plain Neighbor Access. As before, this variant is easy: after Bob learns the intersection, he simply samples one value from the β th attribute of the neighbor tuples.

7.3.2. Blind Neighbor Access. Similar to the solution in Section 7.2.2, Alice and Bob engage in circuit-PSI and obtain a vector of secret-shared bits that indicate which tuples are global neighbors. The parties now randomly select one of the global neighbors using secure two-party computation as follows. They compute the size η of the set intersection from the indicator vector and jointly sample a random value $\mu \in [0, \eta - 1]$ with p bits of precision. That is, Alice and Bob enter randomly chosen p -bit r_A and r_B , respectively,

TABLE 1: Complexity of proposed solutions.

Split method	Sampling type	Computation	Communication
Horizontal	Plain, Mean (Sec. 6.2.1)	$O(\kappa(m \cdot n_A + I_B))$	$O(\kappa(m \cdot n_A + I_B))$
	Plain, Random (Sec. 6.3.1)		
	Blind, Mean (Sec. 6.2.2)	$O(\kappa((m + \kappa)n_A + I_B))$	$O(\kappa((m + \kappa)n_A + I_B))$
	Blind, Random (Sec. 6.3.2)	$O(\kappa((m + \kappa) \min(n_A, c \cdot d) + I_B))$	$O(\kappa((m + \kappa) \min(n_A, c \cdot d) + I_B))$
Vertical	Plain, Mean (Sec. 7.2.1)	$O(\kappa(I_A + I_B))$	$O(\kappa(I_A + I_B))$
	Plain, Random (Sec. 7.3.1)		
	Blind, Mean (Sec. 7.2.2)	$O(\kappa(I_A + I_B))$	$O(\kappa(I_A + I_B))$
	Blind, Random (Sec. 7.3.2)		

add them using secure computation $r = r_A + r_B \bmod 2^p$, scale the result by multiplying it by η , and truncate the resulting $\eta \cdot r$ by p bits. Finally, they scan through the items and privately select the μ th element from the intersection using the indicator vector and output its value to Bob.

8. Analysis

We next analyze performance of our solutions as a function of the input size and security parameters, as well as show that they achieve security under the standard security definitions in the presence of semi-honest participants.

8.1. Complexity Analysis

Table 1 summarizes the computation and communication complexity of the developed solutions. We discuss horizontally and vertically partitioned constructions in turn.

Horizontally split data. The core of our solutions for horizontal partitioning is OPPrF evaluation of quantized tuple values for the purposes of determining nearest neighbors among Alice’s records. This involves building and evaluating an OPPrF on $O(m \cdot n_A)$ inputs. Because each evaluation involves cryptographic operations (although inexpensive) that depend on a security parameter κ , we express the overall cost as $O(\kappa \cdot m \cdot n_A)$. This is also the amount of communication associated with OPPrF evaluation.

The imputation value computation contributes additional cost. With plain neighbor access as described in Sections 6.2.1 and 6.3.1, computing the imputation value from the nearest neighbors among Alice’s records is simple. Doing so for Bob’s neighbors, on the other hand, involves work linear in the number of neighbors, $|I_B|$, performed using secure two-party computation, which contributes $O(\kappa \cdot |I_B|)$ to both computation and communication.

Blind neighbor access, as described in Sections 6.2.2 and 6.3.2, has two significant differences. First, determining the nearest neighbors among Alice’s records and sampling one of them is now more complex. For the mean computation, the largest component involves performing n_A equality comparisons on κ -bit inputs. While it is possible to perform this computation on 1-bit shares, e.g., using GMW [15] combined with oblivious transfer for multiplication triple generation [11], this will result in increasing the round complexity of the construction. In our implementation, we choose to proceed with garbled circuit evaluation [38] from

TABLE 2: Information disclosure summary.

Splitting	Sampling	Information disclosure	
		to Bob	to Alice
Horizontal	Plain		β, I_A
	Blind		β
Vertical	Plain	$I_A \cap I_B$	α
	Blind		α

ABY, which represents each bit as a κ -bit label. This means that the work and communication asymptotically rise to $O(\kappa^2 \cdot n_A)$ in our implementation. Second, for random sampling, Alice’s items are combined, which impacts the cost of OPPrF evaluation. OPPrF construction and evaluation in Section 6.3.2 involves $O(\kappa \cdot m \cdot \min(n_A, c \cdot d))$ computation and communication.

Vertically split data. Our solutions for vertical partitioning are fundamentally different and involve evaluating PSI as the first step. This can be implemented using $O(\kappa(|I_A| + |I_B|))$ computation and communication. This time, I_A and I_B correspond to the nearest neighbors computed using partial matching and thus are larger in size than the set of true neighbors (and consequently larger than the corresponding sets I_A and I_B from the horizontally split solutions).

Determining an imputation value with plain neighbor access, as described in Sections 7.2.1 and 7.3.1 does not have additional costs, while blind neighbor access from Sections 7.2.2 and 7.3.2 requires us to protect the result of set intersection and sample from the intersection using secure two-party computation. This component is also linear in the size of the local sets of nearest neighbors, where we invoke circuit-PSI that uses a hash table of linear size and scan through the result to determine the imputation value.

8.2. Security Analysis

We use a standard simulation-based formulation of security for two-party computation to demonstrate that the proposed protocols provide the expected security guarantees. Because the information disclosure with plain sampling or mean computation allows for inferences, our focus is on the blind variants of the protocols. Due to space limitations, detailed security analysis is given in Appendix A, while Table 2 summarizes information disclosure of different constructions.

In Appendix B, we propose modifications to our protocols which eliminate all leakage, but we leave whose

evaluation to future work due to their large expected run time.

9. Evaluation

In this section, we present a thorough evaluation of our private data imputation protocols, analyzing their performance across varying datasets and configurations. Our evaluation is centered on two main criteria: (i) accuracy, demonstrating the quality of imputed data when leveraging private inputs and (ii) efficiency, examining computational and communication costs. We show the effectiveness of combining private data in enhancing imputation accuracy and demonstrate that our protocols are computationally efficient, completing imputation tasks on datasets with up to 100,000 records within seconds, even in constrained network settings.

Before presenting our experimental results, we first detail our methodology for selecting the radius parameter referenced in Section 4, which is a critical component that distinguishes our approach from traditional k -NN imputation and directly impacts both the accuracy and privacy properties of our protocol.

9.1. Adaptive Radii Search

For each attribute j in our protocol, we determine an optimal radius r_j through a systematic optimization process during the training phase. This section details our empirical methodology for radius selection, which balances imputation accuracy with computational efficiency requirements.

Radius Optimization. Our radius selection process follows a data-driven approach:

- **Initialization:** We begin with an initial radius of 0.35 times the standard deviation of each attribute, providing a starting point that accounts for the natural variability of each feature.
- **Adaptive search:** We employ a convex optimization procedure that systematically explores the radius parameter space. The search begins with small radii values and incrementally increases them while monitoring imputation quality through RMSE on a validation dataset.
- **Stopping criteria:** The search terminates after observing three consecutive performance declines, indicating we have likely passed the optimal radius value. This approach ensures we find a near-optimal radius without exhaustively searching the entire parameter space.
- **Parameter adaptation:** We employ an adaptive step size strategy with a decay rate of 0.9. It progressively reduces the step size after performance declines to enable finer-grained exploration around promising values.

This optimization process occurs separately for each attribute, resulting in attribute-specific radii that account for their unique statistical properties and contribution to imputation accuracy. The final selected radii are then used during the testing phase to evaluate imputation performance.

Statistical Properties and Privacy Implications. Our radius-based approach offers several important statistical and privacy advantages:

- 1) **Attribute-specific calibration:** By optimizing each attribute’s radius independently, our method adapts to different scales and distributions across features, avoiding the one-size-fits-all limitation of traditional k -NN. This is particularly important in heterogeneous datasets with attributes of vastly different characteristics.
- 2) **Density adaptation:** Unlike k -NN which always selects exactly k neighbors regardless of data density, our approach naturally adapts to varying data densities by considering all records within the optimized distance threshold. This prevents forced inclusion of dissimilar records in sparse regions.
- 3) **Outlier robustness:** The radius approach inherently provides protection against outlier influence by excluding distant points regardless of the number of available neighbors, maintaining relevance of the imputation basis.
- 4) **Privacy-utility tradeoff management:** The radius directly influences the privacy-utility tradeoff by controlling both the neighbor set size and the quantization granularity. Smaller radii create finer-grained partitions that yield better imputation accuracy but potentially reveal more information about the underlying data distribution. Larger radii increase the anonymity set size for each comparison, enhancing privacy protection but potentially reducing imputation accuracy. Our optimization process finds an effective balance point on this privacy-utility curve.
- 5) **Compatibility with secure computation:** The quantization approach enabled by our fixed-radius method is particularly well-suited for secure multi-party computation environments, as it transforms expensive range comparisons into efficient equality checks that can be implemented using protocols like PSI and OPRF as described in Section 3.2.

The empirical results that follow demonstrate that our method consistently outperforms standard imputation techniques across different missingness types while maintaining practical computational efficiency, validating our approach to radius selection and optimization.

9.2. Accuracy

For our accuracy experiments, we used twelve current real-world datasets from OpenML [36], each containing numeric columns. The datasets are summarized in Table 3. Missing values are artificially introduced in both training and testing data, allowing us to assess imputation accuracy in a scenario where models are trained and evaluated on corrupted data.

Our accuracy results focus on demonstrating how leveraging additional private data from Alice can improve the imputation quality of Bob’s data. Since imputation generally benefits from increased data availability, our secure multi-party protocols allow Bob to impute missing values by

TABLE 3: Overview of the datasets used. Imp. is the dataset’s column we used to evaluate the imputation methods, Att. is the number of attributes, and Feat. is the number of numerical columns.

ID	Name	Imp.	Att.	Feat.
42636	Long	x18	4477	20
287	wine_equality	sulphates	6497	12
189	kin8nm	theta8	8192	9
197	cpu_act	pgin	8192	22
198	delta_elevators	curRoll	9517	7
23515	sulfur	a3	10081	7
42183	dataset_sales	day	10738	15
216	elevators	climbRate	16599	19
218	house_8L	P3	22784	9
215	2dplanes	x6	40768	11
1200	BNG(stock)	company1	59049	10
23395	COMET_MC_SAMPLE	wire_id	89640	6

incorporating information about neighbors in Alice’s dataset. We observed that the imputation quality is significantly improved when using Alice’s data in comparison to imputation using only Bob’s local data. This improvement is attributed to the inclusion of additional relevant data from Alice’s dataset, which provides a more robust basis for predicting missing values, particularly in vertically split settings where Alice holds attributes that complement those in Bob’s records.

We conducted experiments using three missingness patterns: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR). For each pattern, we introduced 10% missing values to the specified “to-be-imputed” column. The imputation quality was then evaluated by comparing the discarded values in this column, used as ground truth, with predictions generated by the imputation model.

MCAR, the most widely used missingness pattern, involves values missing in an entirely random fashion. In this setup, missingness is independent of any observed (i.e., in the dataset) or unobserved (i.e., missing) data. In other words, the likelihood of a data point being missing is not influenced by the values of other variables in the dataset or by the actual (but not observed) values in the column with missing data. MCAR is typically implemented by drawing a random number from a uniform distribution for each data entry, discarding a value if the generated number falls below a predefined missingness threshold.

In contrast, MAR and MNAR patterns are more complex and less commonly addressed in the literature [3], [34], [35]. MAR implies that missingness is related to observed data but not to the missing data itself, while MNAR, the most challenging case, involves missingness that depends on unobserved or missing data values.

To model these patterns realistically, we follow approaches inspired by observations in large-scale, real-world datasets. Using the method proposed by Schelter et al. [34], [35], we select two random percentiles in the value range of a column – defining upper and lower bounds. For MAR, we discard values in a target column when an entry in a different, randomly chosen column falls within the specified

percentile range. This method controls the missingness by linking it to observed data in a related column. For MNAR, values are discarded directly if they fall within the selected percentile range of the target column itself, making the missingness dependent on the values in the column being imputed.

To ensure statistical robustness, each experiment was conducted per dataset and missingness types, where we re-sampled the missing data for each evaluation and repeated the data splits between Alice and Bob 25 times each. This setup helped us account for the variability inherent in missingness patterns and data partitioning. To evaluate the accuracy of our imputation, we used root mean square error $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$, where n is the total number of observations, y_i is the actual (observed) value, and $\hat{y}_i$ is the predicted values.

Our findings show that utilizing Alice’s data in the computation in a privacy-preserving manner provides significant improvements in imputation accuracy over local imputation. This is due to the additional context and data points provided by Alice, which enables a more accurate imputation of the missing values. Figure 3 shows the results for three representative datasets, with a complete set of our results provided in Figures 4–6 in Appendix C. Splitting repetitions in each plot have been ordered by increasing vertical k -NN RMSEs. The width of the bands in each plot illustrates the 95% confidence intervals for the RMSE values.

For MCAR, as illustrated in Figure 3a, the RMSE results demonstrate the robustness of our protocols across different datasets. When comparing our imputation method – denoted r -NN – with the methods for vertically and horizontally partition data (50%) using k -NN on a numerical column (mean computation), we find that we perform on average across all datasets and all splitting repetitions 20% better than vertical k -NN, and 5% better than horizontal k -NN. The scaled max for MCAR, shown in Table 4, indicates that – for a specific (worse) splitting – our imputation method is up to 5 times more accurate in comparison with vertical k -NN and 4.2 times better than horizontal k -NN, highlighting the challenge of accurate imputation in this scenario.

For MAR, as shown in Figure 3b, the RMSE trends closely mirror those observed under MCAR, but with a slight decrease in error margins due to the conditional dependencies influencing the missing data mechanism. On average, vertical k -NN is 10.6% worse than our imputation method, while horizontal splitting performs marginally better, with an improvement of 0.7%. However, Table 4 highlights that in the worst splitting scenario, our imputation method can be up to 4.1 times more accurate than vertical k -NN and about 3 times more accurate than horizontal k -NN. Thus, even though the MAR missingness type can introduce additional complexity, our protocol remains robust.

The MNAR scenario, depicted in Figure 3c, presents the most significant challenge for accurate imputation where we experience a more pronounced decrease in RMSE due to its inherent bias in missingness. Our method on average outperforms vertical k -NN by 4%, while horizontal k -NN

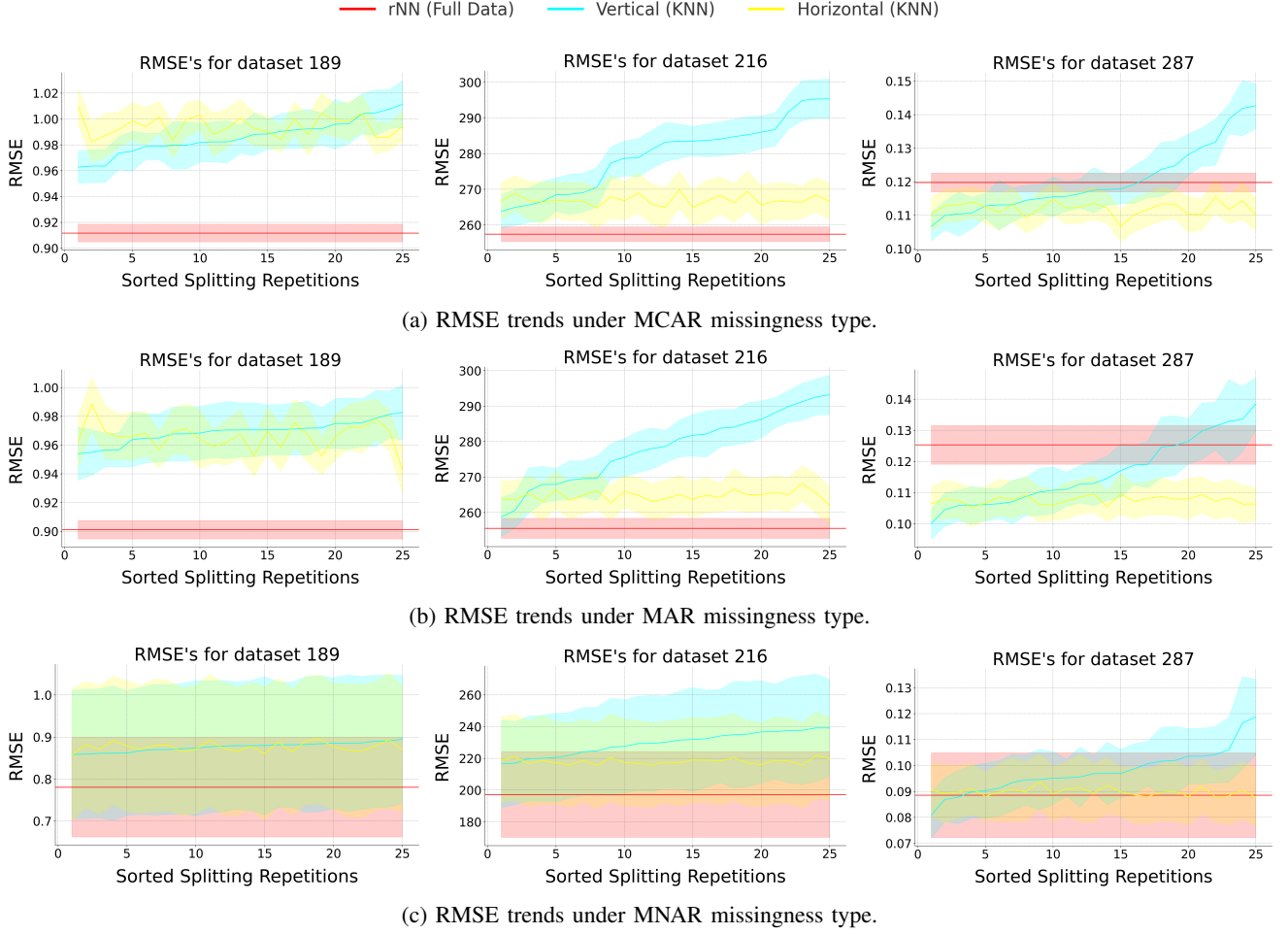


Figure 3: RMSE trends for datasets 189, 216, and 287 under MCAR, MAR, and MNAR missingness types. Splitting repetitions in each plot have been ordered by increasing Vertical k -NN RMSE. The width of the bands in each plot illustrates 95% confidence intervals.

exhibits an advantage of 7.6% over our imputation approach. Despite this comparative disadvantage to horizontal k -NN in the overall average, the worst splitting scenario reveals that our method can be up to 37.2 *times* more accurate than vertical k -NN and 3.4 *times* more accurate than horizontal k -NN. Notably, this represents the largest performance gap achieved in any missingness type – underscoring the substantial disparity posed by biased missingness and the randomness in the splitting.

The high-level summary of the error rates for vertical and horizontal k -NN relative to our method is provided in Table 4. The table consolidates the average and maximum error rates, averaged across 25 runs with 25 splitting repetitions per run. These results underline the variability in performance across missingness types. Our approach’s overage outperforms that of horizontal k -NN for typical missingness type MCAR and consistently outperforms vertical k -NN under all missingness types. Furthermore, the worst-case error for both vertical and horizontal k -NN is several times higher

TABLE 4: Scaled RMSE results across MCAR, MAR, and MNAR.

	Metric	MCAR	MAR	MNAR
Mean (average) error	r -NN (Full Data)	100.00	100.00	100.00
	k -NN (Full Data)	101.5	95.6	91.5
	Vertical k -NN	120.1	111.8	103.8
	Horizontal k -NN	104.9	99.3	92.5
Mean (worst) error	r -NN (Full Data)	100.00	100.00	100.00
	k -NN (Full Data)	117.06	96.88	306.75
	Vertical k -NN	499.2	410.7	3715.75
	Horizontal k -NN	426.1	299.9	341.71

for all missingness types than that of our solution. This shows that our proposed r -NN-based method consistently achieves strong performance relative to the vertically and horizontally k -NN-based approaches. These findings highlight the practical resilience of our method, and emphasize the importance of selecting the appropriate splitting method based on the missingness type and dataset characteristics.

TABLE 5: Performance evaluation of horizontal split protocols; the runtime is in seconds and communication in MB.

Sampling	Network	Medium Dataset					Large Dataset				
		OPPRF	ABY		Total	Comm.	OPPRF	ABY		Total	Comm.
			Build	Eval				Build	Eval		
Plain, Mean	10 Gbps	6.874	0.001	3.503	10.378	28.25	11.178	0.002	11.194	22.374	55.46
	1 Gbps	8.345	0.001	3.371	11.718		12.480	0.002	11.420	23.902	
	100 Mbps	12.601	0.001	4.114	16.717		19.869	0.002	12.211	32.082	
Plain, Random	10 Gbps	6.919	0.038	3.397	10.354	31.37	11.060	0.065	11.568	22.693	62.35
	1 Gbps	8.312	0.038	3.868	12.218		12.693	0.065	12.697	25.455	
	100 Mbps	12.622	0.038	5.671	18.331		19.290	0.065	14.508	33.863	
Blind, Mean	10 Gbps	6.903	5.177	8.6294	20.709	561.93	11.120	10.216	16.682	38.018	1123.1
	1 Gbps	8.037	5.159	10.546	23.742		12.976	10.214	19.673	42.863	
	100 Mbps	12.663	5.166	37.905	55.734		20.058	10.259	74.176	104.493	
Blind, Random	10 Gbps	3.393	0.923	1.697	6.013	76.44	3.859	1.694	2.854	8.407	151.57
	1 Gbps	4.116	0.947	2.716	7.779		4.702	1.718	4.122	10.542	
	100 Mbps	6.268	0.948	8.174	15.39		7.668	1.728	13.236	22.632	

TABLE 6: Performance evaluation of vertical split protocols; the runtime is in seconds and communication in MB.

Sampling	Network	Medium Dataset					Large Dataset				
		PSI	ABY		Total	Comm.	PSI	ABY		Total	Comm.
			Build	Eval				Build	Eval		
Plain	10 Gbps	0.532			0.532	0.88	0.570			0.570	0.89
	1 Gbps	0.949			0.949		1.156			1.156	
	100 Mbps	2.979			2.979		2.984			2.984	
Blind, Mean	10 Gbps	1.116	0.154	0.613	1.883	7.23	1.198	0.229	0.814	2.241	15.13
	1 Gbps	1.936	0.164	1.387	3.487		2.173	0.290	1.496	3.959	
	100 Mbps	5.209	0.151	2.839	8.199		5.116	0.263	3.687	9.066	
Blind, Random	10 Gbps	1.143	0.189	0.642	1.974	8.35	1.255	0.291	0.811	2.357	17.47
	1 Gbps	1.951	0.195	1.113	3.259		2.174	0.318	1.527	4.019	
	100 Mbps	5.355	0.170	2.945	8.471		5.770	0.328	3.644	9.742	

9.3. Performance

We provide performance results of our protocols for horizontally and vertically split data, showing their efficiency under realistic settings.

Implementation and Experimental Setup. Our implementation [2] is written in C++ utilizing ABY [11] for secure two-party computation and VOLE-PSI [32] for private set intersection. Experiments were conducted on a machine with an Intel Xeon E-2374G CPU @ 3.70GHz, 128 GB RAM, and Ubuntu 22.04.4 LTS. All times were averaged over 25 runs.

We used two synthetic datasets and varied (i) the dataset size to reflect typical real-world datasets and (ii) the percentage of neighbors of a tuple. The latter allows us to compute parameters c and d when performing blind random sampling on horizontally partitioned data and is also used as the number of local neighbors with vertically partitioned data.

Our medium dataset has 50,000 tuples with 10 attributes and 1% probability of a tuple being a neighbor. For a desired probability of failure $\varepsilon \leq 2^{-40}$, we set $c = 100$ and $d = 28$. Our large dataset has 100,000 tuples with 10 attributes and 0.5% neighbor probability. For the same ε , we obtain $c = 200$ and $d = 28$.

To emulate network settings for range of typical use cases, we used the Linux Netem and Traffic Control (TC)

tools. These allowed us to precisely vary the bandwidth and latency as follows:

- 10 Gbps with 2 ms round trip time (RTT),
- 1 Gbps with 15 ms RTT,
- 100 Mbps with 100 ms RTT.

We note that the VOLE-OPPRF implementation we use from [32] significantly influenced our performance evaluation setup. Currently, VOLE-OPPRF requires datasets to be sized as powers of 2 and restricts each set to a maximum of about two million blocks. To accommodate these limitations, we could not test our protocols on larger datasets and padded the input size in the horizontal partitioning solutions to the nearest power of 2. For instance, in the plain and blind mean solutions, padding expanded Alice’s dataset from 500K to over 524K entries for the large dataset and from 250K to over 262K in the medium dataset (5% increase for both). In the blind random sampling solutions, Alice’s dataset grew from 56K to over 65K for the large data and from 28K to almost 33K for the medium dataset (17% increase for both). This inflates both the computation and communication overhead of OPPRF evaluation.

Horizontally Split Data. The performance of horizontally split data imputation methods is given in Table 5. We subdivide the runtime into OPPRF building and evaluation and ABY offline (Build) and online (Eval) time. The Build time is primarily from Yao’s circuit garbling in ABY and is nearly absent when the majority of ABY computation uses

arithmetic representation. Four protocols were evaluated: plain mean computation, plain random sampling, blind mean computation, and blind random sampling, with the last two being more attractive based on their security properties.

Blind random sampling consistently delivers the fastest imputation times across all network settings for both datasets because of its lower complexity dependent on parameters c and d as opposed to the number of items in Alice’s dataset for all other solutions. It is followed by plain imputation methods (which also have the lowest communication), but we do not consider those as providing high security guarantees. Overall, secure imputation (blind methods) can be achieved for our datasets on the order of a few to tens of seconds.

Vertically Split Data. The performance of vertically split data imputation methods is shown in Table 6. Plain imputation (the same for mean and random sampling) achieves the lowest runtime and communication overhead across all settings, but of course is not as secure. Among the secure methods, blind mean computation slightly outperforms blind random sampling in both runtime and communication. Overall, these protocols are noticeably faster, performing on the order of seconds, which is due to their dependency only on the number of neighbors in Alice’s and Bob’s datasets.

10. Conclusions

In this work, we study data imputation as a necessary prerequisite to data analysis in practical scenarios when certain values in a dataset are missing and need to be filled in using values in similar records. We consider a dataset with sensitive data partitioned among two different entities, Alice and Bob, and construct privacy-preserving imputation solutions for both horizontally and vertically split data.

Our solution uses a variant of nearest neighbor computation that determines similar records, where we quantize the attributes in a novel approach that can be efficiently realized using cryptographic building blocks. Our construction of minimal leakage for horizontally partitioned data is a combination of OPPRF evaluation and optimized sampling from or aggregation of Alice’s and Bob’s neighbor records. Our construction of minimal leakage for vertically partitioned data relies on efficient private set intersection followed by private sampling from or aggregation of the result.

Our empirical evaluations demonstrate that collaborative imputation, on average, improves the accuracy of the results over locally determined imputed values. On some data splits, it improves accuracy dramatically. This is despite quantization that we employ to boost efficiency of the algorithms. Furthermore, imputing an item in a dataset with 100,000 tuples and 10 attributes can be realized within seconds on a fast network and within 10–20 seconds on a slow network with high latency.

References

- [1] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Association for Computing Machinery*, 51(1):117–122, 2008.
- [2] Anonymous. Private data imputation: Anonymized implementation. https://anonymous.4open.science/r/Private_Data_Imputation-FA42, 2025. Anonymized for review.
- [3] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. “deep” learning for missing value imputation in tables with non-numerical data. In *ACM International Conference on Information and Knowledge Management*, page 2017–2025, 2018.
- [4] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. *International Journal of Information Security*, 15(5):493–518, 2016.
- [5] Erik-Oliver Blass and Florian Kerschbaum. Private collaborative data cleaning via non-equi PSI. In *IEEE Symposium on Security and Privacy (S&P)*, pages 1419–1434, 2023.
- [6] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya Razenshteyn, and M. Sadegh Riazi. SANNS: Scaling up secure approximate k-nearest neighbors search. In *USENIX Security Symposium (USENIX Security 20)*, pages 2111–2128. USENIX Association, August 2020.
- [7] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Report 1998/003*, 1998.
- [8] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.
- [9] Chris Clifton, Eric J. Hanson, Keith Merrill, and Shawn Merrill. Differentially private k -nearest neighbor missing data imputation. *ACM Transactions on Privacy and Security*, 25(3):1–23, 2022.
- [10] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security (FC)*, pages 143–159, 2010.
- [11] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY – A framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium (NDSS)*, 2015.
- [12] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology – EUROCRYPT*, pages 1–19, 2004.
- [13] William Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82(72-107):113, 2004.
- [14] Oded Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.
- [15] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
- [16] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.
- [17] Ihab F. Ilyas and Xu Chu. *Data Cleaning*. Association for Computing Machinery, 2019.
- [18] Geetha Jagannathan and Rebecca N. Wright. Privacy-preserving data imputation. In *IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 535–540, 2006.
- [19] Geetha Jagannathan and Rebecca N. Wright. Privacy-preserving imputation of missing data. *Data & Knowledge Engineering*, 65(1):40–56, 2008.
- [20] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *International Conference on Security and Cryptography for Networks (SCN)*, pages 418–435, 2010.

- [21] Julia Jentsch, Ali Burak Ünal, Şeyma Selcan Mağara, and Mete Akgün. Privacy preserving data imputation via multi-party computation for medical applications. *arXiv Preprint Report 2405.18878*, 2024.
- [22] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.
- [23] Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. A benchmark for data imputation methods. *Frontiers in Big Data*, 4, 2021.
- [24] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2), 2021.
- [25] Florian Kerschbaum, Erik-Oliver Blass, and Rasoul Akhavan Mahdavi. Faster secure comparisons with offline phase for efficient private set intersection. In *Network and Distributed System Security Symposium (NDSS)*, 2023.
- [26] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.
- [27] L. Kissner and D. Song. Privacy-preserving set operations. In *Advances in Cryptology – CRYPTO*, pages 241–257, 2005.
- [28] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. Crypten: Secure multi-party computation meets machine learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.
- [29] Mohammed Z. Omer, Hui Gao, and Nadir Mustafa. Privacy-preserving of SVM over vertically partitioned with imputing missing data. *Distributed and Parallel Databases*, 35:363–382, 2017.
- [30] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In *Advances in Cryptology – EUROCRYPT*, 2020.
- [31] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security*, 21(2):7:1–7:35, 2018.
- [32] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-psi from vector-ole. In *Advances in Cryptology – EUROCRYPT*, 2021.
- [33] Joseph L. Schafer and John W. Graham. Missing data: our view of the state of the art. *Psychological methods*, 7, 2002.
- [34] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. Learning to validate the predictions of black box classifiers on unseen data. In *ACM SIGMOD International Conference on Management of Data*, pages 1289–1299, 2020.
- [35] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. Jenga: A framework to study the impact of data errors on the predictions of machine learning models. In *EDBT Industrial and Application Track*, 2021.
- [36] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explor. NewsL.*, 15(2):49–60, 2014.
- [37] Amanda Walker, Sarvar Patel, and Moti Yung. Helping organizations do more without collecting more data. Technical report, Google Security Blog, 2019.
- [38] Andrew C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS)*, pages 162—167, 1986.
- [39] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. Gain: Missing data imputation using generative adversarial nets. In *International Conference on Machine Learning (ICML)*, pages 5675—5684, 2018.
- [40] Ian Zhou, Farzad Tofigh, Massimo Piccardi, Mehran Abolhasan, Daniel Robert Franklin, and Justin Lipman. Secure multi-party computation for machine learning: A survey. *IEEE Access*, 12, 2024.

Appendix A. Security Analysis

Recall that in the ideal case Bob learns only the imputed value $x_{\alpha,\beta}$ (which is necessarily a function of Alice’s data) and no other information about Alice’s dataset, while Alice learns nothing. The solutions proposed in this work, however, may disclose additional information. Thus, in this section we analyze the protection offered by our constructions.

Horizontally partitioned data. As mentioned earlier, the solution where Alice learns the set of t_α ’s neighboring tuples among her items, I_A , discloses a significant amount of information because it allows Alice to approximate Bob’s tuple t_α . For that reason, we consider the initial solutions with plain mean computation or sampling in Sections 6.2.1 and 6.3.1 as too revealing and treat the final solutions in Sections 6.2.2 and 6.3.2 as our main construction.

In our solutions, Alice learns the value of β that corresponds to the missing attribute in one of Bob’s tuples. This is a small amount of information to learn and this information may not be secret to Alice: given that Alice possesses tuples with the same attribute, she may already know based on her own dataset that some of the records have their β th attribute missing. However, hiding β will inevitably increase the cost of the computation.

We consider a standard simulation-based definition of security, where all a party’s view can be simulated solely from their input and output. If such a simulated view is shown to be indistinguishable from their view during a real protocol execution, the protocol execution does not reveal any private information. A view is formed by all of the information a party possesses, including its inputs, messages received during the protocol execution, randomness used during the protocol, and the output obtained as a result of the protocol. We sketch security proofs for Alice and Bob for our construction.

Bob has his private input dataset and initially engages in OPPrF evaluations with Alice. OPPrF’s properties are such that Bob does not learn any new information as part of that interaction. Next, Alice and Bob engage in private sampling or mean calculation by means of secure two-party computation. Bob inputs (among other values) the β attribute from all of the neighbor records I_B among his tuples and learns a value to use for $x_{\alpha,\beta}$. Because the secure two-party protocol has the desired properties (i.e., the computation can be simulated without the other party’s input and the simulated view is indistinguishable from the real execution) and because Bob uses all of his neighbors, he is unable to learn any information beyond the final outcome.

As far as Alice’s view goes, she initially learns β and the result of OPPrF evaluation on each attribute in her tuples. Due to the properties of an OPPrF, any evaluation on a new point is indistinguishable from a randomly generated string. We program the OPPrF differently for each tuple and for each attribute, which ensures that all values Alice obtains

are independent of each other and indistinguishable from uniformly chosen random values.

Alice and Bob consequently engage in secure two-party computation. Alice enters the results of OPPrF evaluations for all of her tuples and receives no output as a result of that computation. Due to the properties of the secure two-party computation protocols, Alice’s view during the computation is indistinguishable from a simulation that does not use any Bob’s inputs and Alice does not learn any information, including which of her records were identified as neighbors of Bob’s tuple t_α .

Vertically partitioned data. In the case of vertically partitioned data, the computation involves PSI or circuit-PSI followed by secure two-party computation.

Consider Alice’s view. In our solution Alice learns the index α , i.e., that Bob is imputing a value in his tuple t_α . This is a small amount of information that allows us to achieve performance with communication and the number of cryptographic operations being sub-linear in the size of Alice’s dataset. Given that knowledge, Alice computes the set of neighbors of tuple t_α in her dataset and does not learn any other information throughout protocol execution.

Specifically, when engaging in a PSI protocol, either Bob learns the resulting set intersection or the parties obtain secret shares of the indicator vector. In both of those cases, security of our solution relies on the security of the underlying PSI protocols and the view can be simulated using PSI’s simulator.

With plain access in Sections 7.2.1 and 7.3.1, no additional computation with private values is needed, but Bob learns the tuples in the intersection, i.e., global neighbors, which is undesirable. Thus, we next analyze enhanced solutions, with blind access in Sections 7.2.2 and 7.3.2, that do not have this drawback.

In that case, we combine circuit-PSI with secure two-party computation, neither of which provide any information to Alice. That is, both the PSI and secure two-party computation techniques come with simulators that will allow us to create Alice’s view without Bob’s data which will be indistinguishable from the protocol execution. Similarly, Bob’s view can be simulated from his output alone by relying on the security of the building blocks we invoke.

Table 2 summarizes information disclosure of different constructions.

Appendix B. Extensions

Recall that our previously described solutions have a small amount of information leakage beyond the function output. In particular, in the horizontally split case, Alice learns the attribute β Bob wants to impute in his tuple; in the vertically split case, Alice learns the index α of the tuple in which Bob wants to impute a value. In this section, we comment on extensions to our protocols that eliminate that information disclosure.

B.1. Hiding β with Horizontally Partitioned Data

In Section 6, we have assumed that both Alice and Bob know β , but it is actually not necessary for Bob to reveal β to Alice. Remedying this requires Alice and Bob to perform an imputation for each possible value of β and then Bob privately selecting the correct one. This is very inefficient, yet there is an opportunity for Alice and Bob to reduce the cost in case of multiple imputations. Alice and Bob can first partition their data using a locality-sensitive hash function (LSH) [1] and then operate over each partition. LSHs are a family of hash functions satisfying the following property: for any given pair of vectors in space, if the vectors are close, the probability of their hashes colliding is high, else the probability of their hashes colliding is low. Note that Alice and Bob need to compute the same number of imputations for each bucket of the LSH to not reveal those that have missing values. Hence, this method only provides a performance gain in case there are multiple imputations.

B.2. Hiding α with Vertically Partitioned Data

So far, we have assumed in Section 7 Alice computes her neighbors locally knowing α . However, we can also hide α from Alice. For this, we use a technique called private information retrieval (PIR) and combine it with Diffie-Hellman based PSI. In Diffie-Hellman PSI, for each tuple index $m_i \in I_A$, Alice computes and sends $H(m_i)^a$ to Bob. Similarly, for each index $m_j \in I_B$, Bob computes and sends $H(m_j)^b$ to Alice. Alice computes $(H(m_i)^b)^a$ in $\mathbb{G}$ and Bob computes $H((m_j)^a)^b$. It holds that

$$m_i = m_j \Leftrightarrow H(m_i)^{ab} = H(m_j)^{ba}.$$

PIR achieves privacy by ensuring that no single server or group of colluding servers can infer the query's target, thereby preserving the client's data access patterns. In PIR, Alice has a database and Bob accesses an element at an index in the database without revealing which index it is, which naturally generalizes to retrieving buckets from a hash table [7], [8], [13].

We construct the database as follows. For each tuple t_k , Alice computes her set of neighbors $I_{A,k}$. For each $m_i \in I_{A,k}$ Alice computes the first step in Diffie-Hellman based PSI $H(m_i)^a$. The element at index k in the database is set $\{H(m_i)^a \mid m_i \in I_{A,k}\}$.

We continue the PSI protocol as follows. Instead of Alice sending $H(m_i)^a$, Bob retrieves $H(m_i)^a$ for index α using PIR. The protocol then proceeds as standard Diffie-Hellman based PSI (Bob computes and returns $H(m_i)^{ab}$) and if we stop after Alice's step (of exponentiating with a^{-1}) we can use it to build circuit-PSI.

Appendix C.

Additional Accuracy Evaluation Results

We report comprehensive accuracy results for all evaluated datasets under MCAR, MAR, and MNAR missingness types in Figures 4–6.

Each figure represents the performance of the imputation for a specific dataset as detailed in Table 3. Splitting repetitions in each plot have been ordered by increasing Vertical k -NN RMSE. The width of the bands in each plot illustrates 95% confidence intervals.

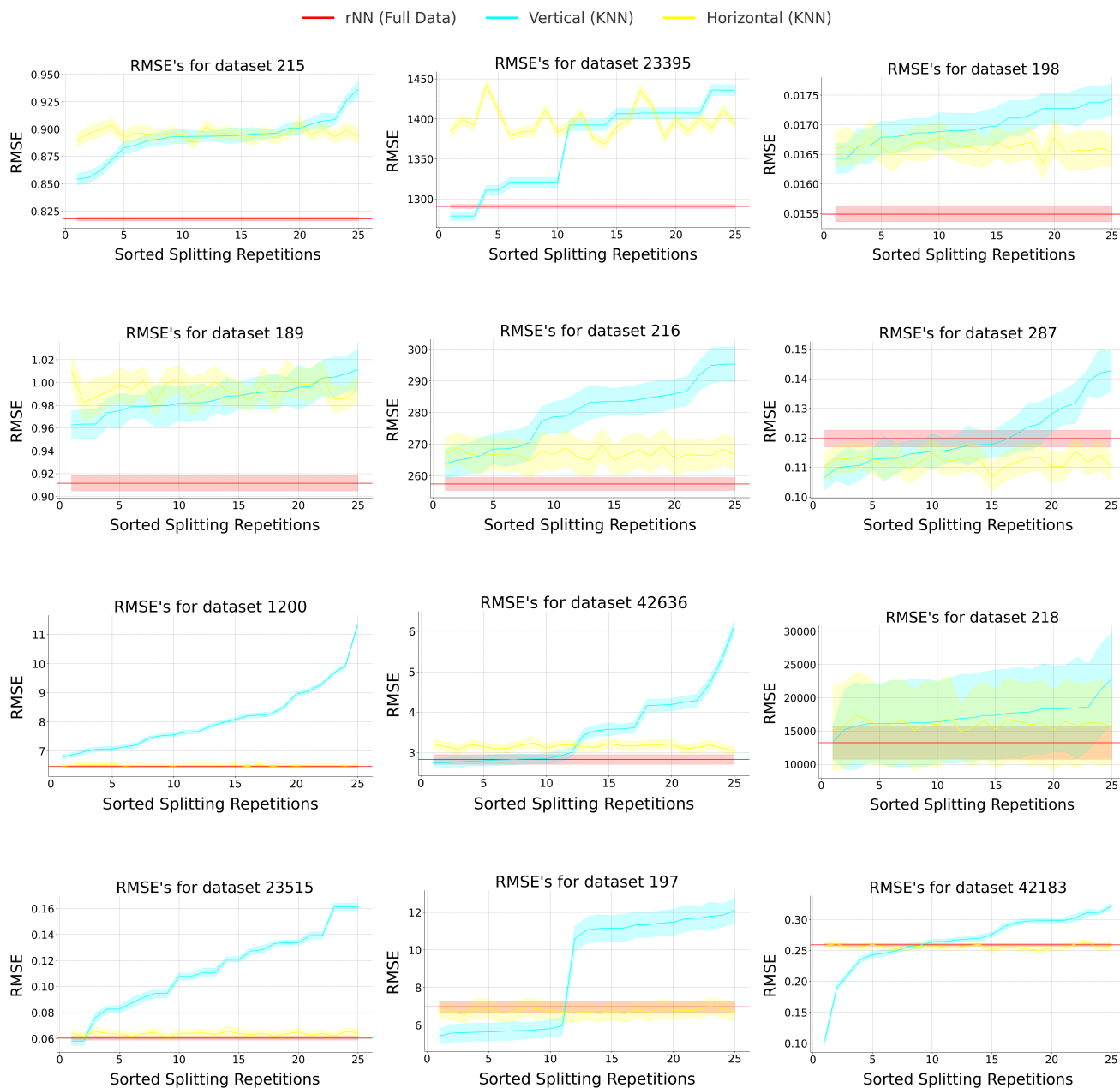


Figure 4: Detailed RMSE trends for all evaluated datasets under the MCAR missingness type.

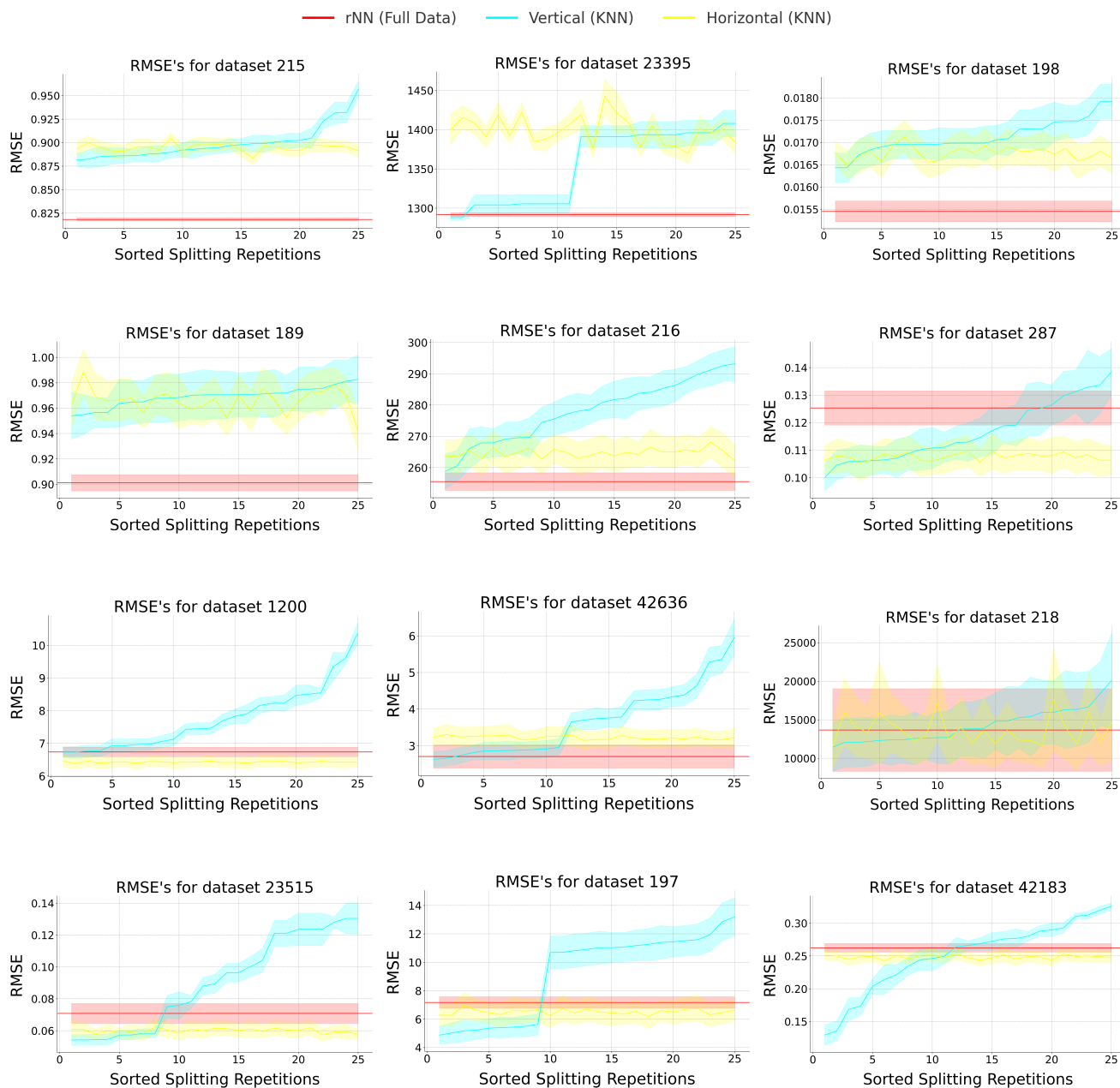


Figure 5: Detailed RMSE trends for all evaluated datasets under the MAR missingness type.

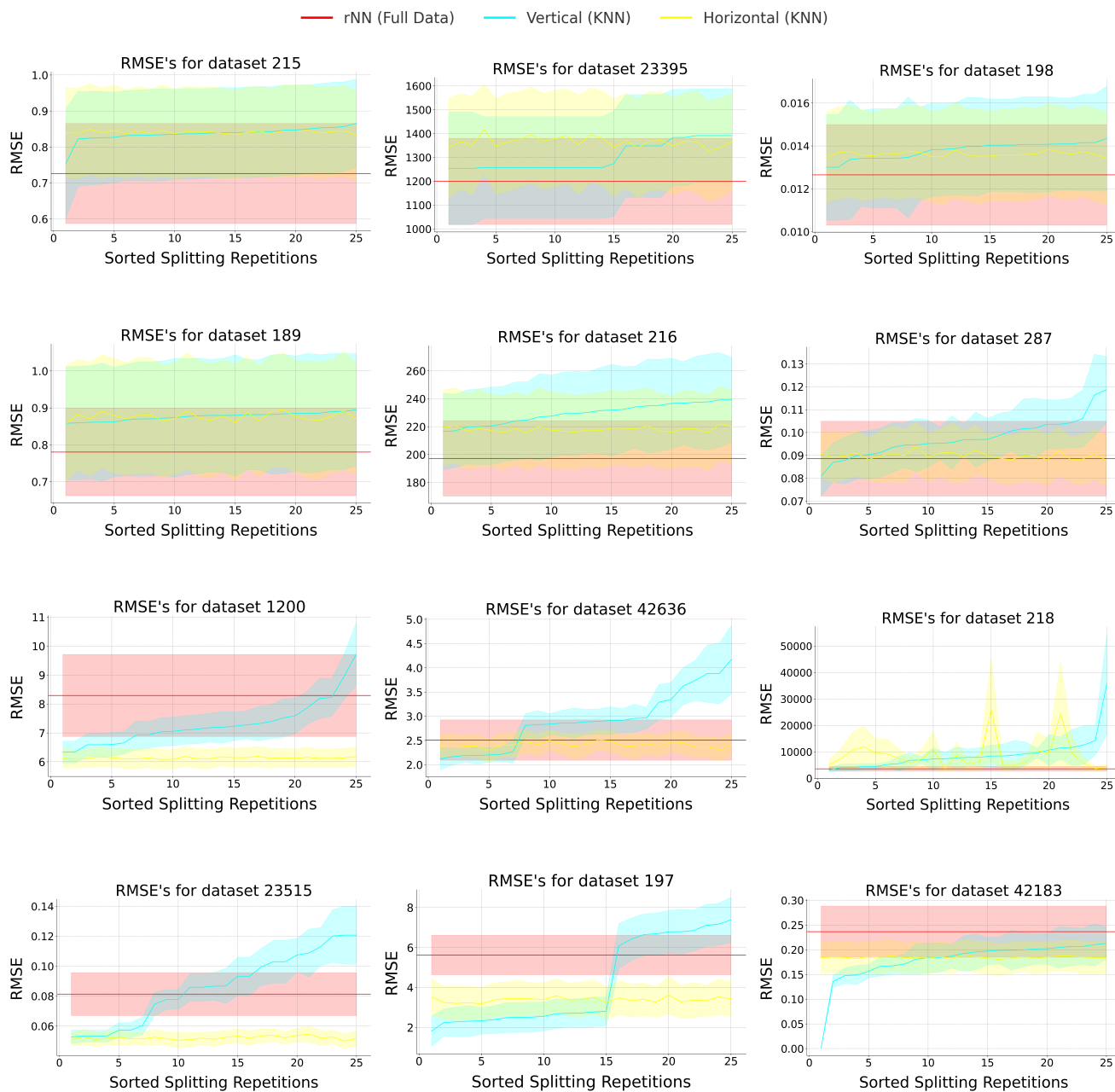


Figure 6: Detailed RMSE trends for all evaluated datasets under the MNAR missingness type.