Missing Value Imputation in Tabular Data Lakes Unleashed: A Hybrid Approach

Anonymous Author(s)

ABSTRACT

Missing values in tabular data lakes can severely impact data analysis and diminish the performance in downstream applications. We highlight that a robust imputation strategy should properly take three aspects of variety into consideration: source of imputed value (i.e., from the same table of the missing value or external tables), the types of tables involved, and the data types of the missing value. Unfortunately, no existing approaches effectively incorporate all three aspects of variety. To address this gap, we propose CESID, a novel framework that uses a Combination of Estimation-based and Search-based methods for missing value Imputation in Data lakes. CESID contains three core modules: the Contextual Search Module, the Acquisition-guided Estimation Module, and the Classification Module. The Contextual Search Module constructs a graph to identify the most relevant columns for imputing missing values and efficiently discovers candidate values from other tables. The Acquisition-guided Estimation Module introduces an influence function and a sampling-based exploration strategy to efficiently identify rows from other tables, yielding more accurate estimated values. The Classifier Module employs a learned classifier that exploits table-level and column-level statistics to determine the most suitable method for imputation. Extensive experiments conducted on three data lakes demonstrate that CESID effectively and efficiently addresses the missing value problem.

ACM Reference Format:
Anonymous Author(s). 2018. Missing Value Imputation in Tabular Data Lakes Unleashed: A Hybrid Approach. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Tabular data lakes provide flexible and scalable data storage and management capabilities, and have become increasingly critical to managing multiple tabular datasets [23, 53]. However, poor data collection processes [41], equipment malfunctions, and sensor failures [18, 33] often result in missing values in these datasets. These missing values can severely impact data analysis and reduce the performance of downstream applications [12, 36, 40].

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

Research Gap. We observe that designing a robust imputation method for tabular data within a tabular data lake[1] is notoriously difficult due to three main aspects of variety:
A1: *Source of Imputed Value* – a missing value in a table T can arise in three typical scenarios: (S1) the missing value can be retrieved from tables (in the data lake) other than T; (S2) missing values cannot be directly retrieved but have similar values available in tables other than T; (S3) exact or similar values are available only in T.
A2: *Table Types* – *HTML Tables* and *CSV Tables* are two typical table types in a data lakes. *HTML Tables* are often small, with very few rows and columns [25], but they benefit from rich metadata, like titles and captions. In contrast, *CSV Tables* tend to be larger, with more rows (e.g., tens of thousands) and columns (e.g., 20), but they come with limited metadata and often have incomplete schemas.
A3: *Data Types of Missing Value* – missing values can be either numerical or categorical, adding another layer of complexity to the imputation process.

Existing solutions for tabular data imputation can be categorized as estimation-based methods [9, 10, 42, 44, 59, 65] and search-based methods [6, 7, 22, 63, 68, 70]. Unfortunately, they often overlook at least one aspect of the variety and hence hinder their effectiveness, as we will discuss in Section 2.2 shortly. To put it more concrete, we present a summary of performance under various combinations of these aspects in Figure 1. Here, each radar chart compares the effectiveness our proposal with the best-performing search-based and estimation-based methods. Effectiveness is normalized by the highest-performing method in each case, with higher values indicating better performance.

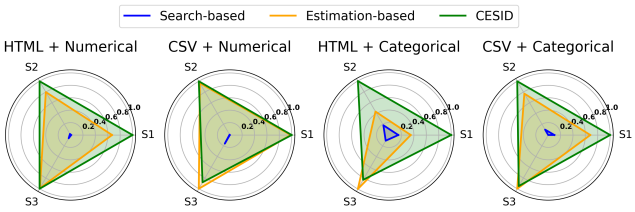


Figure 1: A summary of effectiveness comparison among different approaches.[2]

Our Solution. To fill this research gap, we propose CESID, a Combination of Estimation-based and Search-based methods for missing value Imputation in Data lakes. As shown in Figure 2, CESID contains three sub-modules: a Contextual Search Module to retrieve a value (accommodating S1), an Acquisition-guided Estimation Module to estimate a value (accommodating S2 and S3), and a Classifier Module to select which of the two modules above should

[1] In the rest of this paper, "tabular data lake" is simply referred to as "data lake".
[2] The results of *HTML Tables* and *CSV Tables* are reported based on data lakes WebTables and OpenData, respectively (§6).

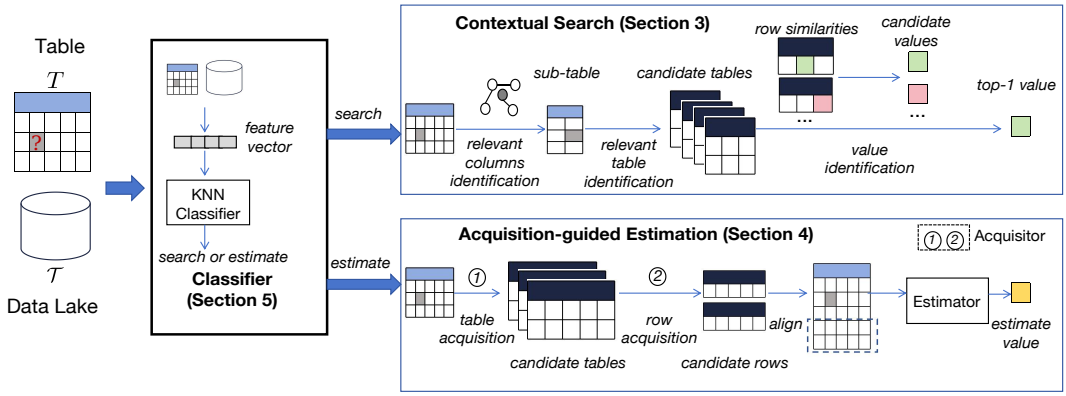


Figure 2: Overview of CESID.

be used to fill a missing value. This hybrid design makes CESID robust in handling different sources of imputed value, thereby addressing A1.

In the Contextual Search Module, we construct a tailored graph, namely *column-relevance graph*, which incorporates rich contextual information in the table for modeling the column relevance. Using this graph, we further identify the most relevant columns, in terms of contextual relevance, for a column containing a missing value. Then, upon the relevant columns identified, a relevant table identification process is implemented to retrieve the relevant tables, which may contain values that can be used to fill a missing value. By focusing on these relevant tables, we can significantly narrow our search space from the entire data lake (e.g., about 7,000 tables) to a small subset (e.g., 10 tables). In order to identify the candidate imputed value from the relevant tables, we define a relevance score for candidate values and select the candidate with the highest score to be used for imputation. In defining a proper relevance score, we carefully consider the impact of the data type and have tailored design for numerical and categorical data types, respectively (addressing A3).

In the Acquisition-guided Estimation Module, instead of proposing a new estimation-based method, our goal is to enhance the effectiveness of existing estimation-based methods by incorporating additional rows, especially for *HTML Tables* (addressing A2). To efficiently identify which rows would be most beneficial as additional training data, we introduce an influence function that assesses each row's impact on the model's estimation performance. We also design a sampling-based exploration strategy to efficiently acquire a set of rows that improve imputation performance. Notably, our acquisition-guided module also lays the groundwork for optimizing various choices of estimation-based methods within data lake environments.

In the Classifier Module, we carefully engineer 16 features, related to the cell with a missing value, to capture a wide range of cases resulting from varieties A1-A3. A KNN classifier is used to find the nearest neighbors in the training data for the cell in terms of the selected features and select which module should be employed via a voting strategy. This module seamlessly combines the strengths of both search-based and estimation-based methods in a unified framework, further enhancing its robustness.

In summary, we make the following contributions:

- We point out three key aspects of variety that affect imputation performance, i.e., source of imputed value, table types, and data types of missing value. To address them, we propose CESID, a hybrid method that effectively and efficiently combines the strengths of both search-based and estimation-based approaches, enabling its robustness in dealing with the above varieties.
- We develop a Contextual Search Module that leverages contextual relevance from the relevant columns, for the column containing a missing value, to identify candidates values from tables, and selects the most promising candidate. (§3)
- We develop an Acquisition-guided Estimation Module to efficiently incorporates additional rows to enhance the effectiveness of existing estimation-based methods. (§4)
- We train a Classifier Module to effectively select the most appropriate imputation method to fill each missing value. (§5)
- We conduct a comprehensive set of experiments to investigate the robustness of CESID in handling the variety of imputation. Overall, CESID achieves an average MAE reduction of 11% for numerical data, and an accuracy improvement of 75% for categorical data, over the best-performing baseline. We believe that our experimental setups, which encompass a wide range of varieties, will serve as a foundation for future research in robust missing value imputation in tabular data lakes. (§6)

2 PROBLEM FORMULATION AND LITERATURE REVIEW

2.1 Problem Definition

Let T be a table containing n rows $\{r_1, r_2, \ldots, r_n\}$ and m columns $\{c_1, c_2, \ldots, c_m\}$, some of which contain missing values. The value for the j-th column in the i-th row is denoted as $r_i[c_j]$. A binary matrix mask $\mathbf{N} \in \{0,1\}^{n \times m}$ is used to indicate the location of missing values in T, where $r_i[c_j]$ is a missing value only if $\mathbf{N}_{ij} = 0$. A data lake containing multiple tables is represented using $\mathcal{T}$.

Definition 1 (Missing Value Imputation in Data Lakes)**.** Given a data lake $\mathcal{T}$ with a set of tables, and a table T containing missing values, the goal is to provide a replacement value using values from $\mathcal{T}$, for each $\mathbf{N}_{ij} = 0$.

2.2 Literature Review

2.2.1 Tabular Data Imputation. Existing tabular data imputation methods fall into two primary categories: search-based and estimation-based.

Search-based methods [6, 7, 22, 63, 68, 70] find candidates to replace missing data using external information (e.g., from tables, knowledge bases, or texts). Some methods [6, 7, 22, 63, 68] leverage external tables to find potential values, similar to our approach. Specifically, methods in [7, 63, 68] treat the local table with missing data as a query table, retrieving relevant tables based on semantic similarities derived from metadata and schema, and then extracting potential values from these tables. However, these methods rely heavily on metadata and complete schemas, which may be ineffective when the metadata and complete schemas are unavailable. To address this, RetClean [6] first indexes the rows from the data lake using text embeddings from language models, for identifying the relevant rows. Then, given a row with a missing value, it retrieves the most relevant rows using embedding-based similarity, and uses the language model to extract potential candidates to replace the missing value. Building on this approach, the more recent method, RATA [22], introduces a retrieval-augmented, self-trained transformer model to enhance imputation effectiveness in each data lake.

However, these methods have several limitations. First, these methods retrieve values from other tables, which can be ineffective in addressing S3, thereby failing to deal with the variety of source of imputed value (as described in **A1**). Second, they still have limited accuracy with *CSV Tables*. *CSV Tables* often contain more columns, which can complicate embeddings, leading to imprecise embedding similarities [28, 67]. This reduces their effectiveness in dealing with the variety of table types (as described in **A2**). Last, these methods treat tables as text and retrieve values based on the textual similarity, which ineffectively handles numerical data, which can comprise about 58% for a table according to [25]. As a result, they struggle to handle a variety of data types of missing value (as described in **A3**).

Other methods like Auto-complete [70] and MAVE [64] are designed to search candidate values from external knowledge bases and unstructured texts respectively. AutoComplete [70] proposes a novel framework, which consists of preprocessing, candidate value finding, and value ranking components, for value identification. MAVE [64] leverages a question-answering model to extract values from textual data.

Estimation-based approaches usually assume that missing values of a table can be addressed by leveraging the data available in itself [8, 13, 44, 51, 59, 65]. Accordingly, most of the estimation-based methods develop the parametric model to capture the data distribution or data correlation to predict values for imputation. To utilize the data distribution information, GAIN [65] provides a generative model that includes a generator and a discriminator to learn the data distribution for estimation. However, it is susceptible to overfitting and may fail to capture the local information (i.e., specific subsets or regions of distribution). To resolve this, NOMI [59] proposes an uncertainty network that sequentially integrates local data retrieval, neural network-based Gaussian processes, and uncertainty-based calibration for imputation. To capture the data correlation information, MICE [51] and MissForest [54] estimate missing values by using multiple regression models or random forests. Datawig [8] combines feature extractors with automatic hyperparameter tuning in a deep-learning library to predict the missing value. The above methods rely on global information (i.e., general patterns in the table), while overlooking local information (i.e., relationships between rows and columns) To address this, GRIMP [13] uses a graph neural network that combines a graph representation of the table and multi-task learning.

However, these estimation-based methods have the following limitations. First, they simply leverage the data from itself, which ignores the potential benefits of utilizing external information. This limits their effectiveness for incorporating two scenarios S1 and S2, reducing their effectiveness in managing the diverse sources of imputed values (**A1**). Second, they may struggle in providing sufficient training data to develop a robust parametric model, leading to suboptimal performance when imputing small *HTML Tables* and an inability to deal with the variety of table types (**A2**). Last, these methods may underperform in imputing categorical values, as they require a large proportion of redundant categorical data to construct the training set. This limitation reduces their effectiveness in handling the variety of data types of missing value (**A3**). Thus, how to leverage the external information in the data lake to prepare the training data and further enhance estimation performance, remains an open question.

Inspired by advancements in large language models (LLMs), some studies [14, 16, 42, 56] use the LLMs to generate values for imputation. Typically, they first pretrain the LLMs, following the "pre-training, fine-tuning" and "pre-training prompts" pipelines, and then generate a value from the pretrained model's vocabulary. However, such methods may be ineffective due to the large classification vocabulary and out-of-vocabulary issue [6], and they require substantial computational power.

2.2.2 Other Related Work. There are other studies loosely related to ours where they address missing values in different domains, such as time series data [45, 57, 60] and spatio-temporal data [46, 66]. However, the structure and characteristics of these data significantly differ from tabular data, making their solutions unsuitable for ours.

3 THE CONTEXTUAL SEARCH MODULE

Key Issues. Existing search-based imputation methods [6, 22] do not work well in a data lake setting due ineffective embedding generated for the tables containing a large number of columns, and typically containing many numerical values. To overcome these issue, three sub-problems must be resolved:

P1 - How to find the relevant columns to effectively and efficiently identify a candidate value, especially for tables that have large column sizes.

P2 - After identifying the relevant columns, how to find the relevant tables that can provide the potential value, based on the relevant columns.

P3 - How to effectively identify the candidate value for numerical and categorical data types using these relevant tables.

Solution Overview. We propose a novel contextual search module to solve the three above issues. We first construct a *column-relevance*

graph for an incomplete table by leveraging the contextual information in the table and identify the most relevant columns to serve as contextual relevance, for the missing column in the same table, i.e., solving **P1** (§3.1). Note that we use "missing column" to refer to a column containing the missing value to be imputed, throughout this paper. Second, we introduce a relevant table identification process to retrieve the relevant tables based on relevant columns, i.e., solving **P2** (§3.2). Last, we define a relevance score for each candidate value and identify the one with the highest relevance score, i.e., solving **P3** (§3.3) .

3.1 Identifying Relevant Columns

Given a missing value $r_i[c_j]$ (the cell at the i-th row and the j-th column) from the table T, a search-based method needs contextual relevance, i.e., relevant columns, to determine from which table/row/value the imputed value might be sourced. However, not all of the other columns in the incomplete table are relevant for a missing column under a general data lake setting, especially for tables containing a large number of columns. To find columns relevant to column c_j, we introduce two forms of column relevance based on the contextual information in table: (1) relevant columns to the same entity, and (2) relevant columns with a dependency.

3.1.1 Relevant Columns To The Same Entity. One common table type in a data lake is an entity-attribute table, e.g., *HTML Tables*, where each row represents an entity, with one column designates an entity and the remaining columns serve as attributes. Given an entity with missing values, we can identify matching entities (referring to the same entity) from other tables and fill in the missing values based on them . Thus, finding columns that represent the same entity as the column containing missing values can improve the search process. Given a table T, a naive approach is to identify the columns included in the same entity and perform manual labeling. However, this is time-consuming, especially, in a data lake setting which can have thousands of tables.

To address this problem, we propose an automatic method that leverages existing knowledge bases (KBs). A KB is a structured database that stores knowledge in the form of triples, including attribute triples and relation triples [71] . The attribute triples, we leverage in this paper, are represented as the triple (entity, attribute, attribute value). For example, the attribute triple (USA, Capital, Washington) encodes the knowledge that the Capital (attribute) of USA (entity) is Washington (attribute value).

Formally, given a column pair (c_j, c_k) from the table T and a knowledge base KB, c_j and c_k are relevant if there exists an attribute p in KB with a co-occurrence score $RS(c_j, c_k, p)$ that exceeds a threshold τ_p. Here, $RS(c_j, c_k, p)$ is computed as $\frac{|\pi_{c_j,c_k}(p) \cap \pi_{c_j,c_k}|}{|\pi_{c_j,c_k}|}$, where π_{c_j,c_k} denotes the value pairs of columns c_j and c_k in the table T, and $\pi_{c_j,c_k}(p)$ represents these value pairs that overlap with attribute triples having attribute p .

3.1.2 Relevant Columns with a Dependency. Relying solely on external knowledge bases can result in low coverage [50] since some attributes of the value pairs in the table do not exist in the KB. To resolve this issue, we introduce a dependency-based approach to identify other relevant columns, inspired by functional dependencies [48, 61].

Algorithm 1: Graph Construction

Input: A table T with columns $\{c_1, c_2, \ldots, c_m\}$; KB; Threshold τ_p;
Output: A column relation graph $\mathcal{G}$

1 $\mathcal{V} \leftarrow$ InitializeNodes($\{c_1, c_2, \ldots, c_m\}$);
2 Column pair set $\mathcal{CP} \leftarrow \emptyset$;
3 $T \leftarrow$ SampleByRows(T);
4 **for** *each column pair* (c_j, c_k) *from* T **do**
  /* Relevant in the same entity */
5 $\mathcal{P} \leftarrow$ FindAttributes(KB, ExtractValuePairs(c_j, c_k))
6 **foreach** *relation* $p \in \mathcal{P}$ **do**
7 $RS(c_j, c_k, p) \leftarrow$ CalculateScore(c_j, c_k, p);
8 **if** $RS(c_j, c_k, p) > \tau_p$ **then**
9 Add (c_j, c_k) to $\mathcal{CP}$;
10 **break**;
  /* Relevant with dependency */
11 **if** *ExistsDependency*(c_j, c_k) **then**
12 Add (c_j, c_k) to $\mathcal{CP}$;
13 $\mathcal{G} \leftarrow$ ConnectEdges($\mathcal{V}$, $\mathcal{CP}$);
14 $\mathcal{G} \leftarrow$ ConnectIndependentNodes($\mathcal{V}, \mathcal{G}$);
15 **return** $\mathcal{G}$

Formally, given a column pair (c_j, c_k) from the table T, c_j and c_k are relevant if there is a dependency between c_j and c_k. The dependency holds if and only if for every pair of rows (r_i, r_l) in table T, whenever $r_i[c_j] = r_l[c_j]$, it follows that $r_i[c_k] = r_l[c_k]$.

Intuitively, the dependency between two columns represents a binary mapping relationship in which one determinant column (e.g., "country") functionally determines another dependent column (e.g., "country-code"). We can view "country" and "country-code" as relevant columns to describe the entity "country". Even though not all columns from such a dependency describes an entity, such as "begin-year" and "end-year", they are still meaningful for imputing missing values. This is because we can use the values from the determinant column (e.g., begin-year") to identify the missing values in the dependent column (e.g., end-year").

3.1.3 Implementation. We organize the relevant columns for a table in a graph, named *column-relevance graph*. Formally, given a set of columns $\{c_1, c_2, \ldots, c_m\}$ from table T, the *column-relevance graph* for T is $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ corresponds to a column c, and each edge $e \in \mathcal{E}$ indicates the columns that e connects with are relevant, either in the same entity or with dependency. The graph construction process proceeds as follows (cf. Algorithm 1):

Step 1: Sample table. We initialize the node set $\mathcal{V}$ and an empty column pair set $\mathcal{CP}$, and randomly sample 10% of rows from each table T. This sampling is designed to enhance the efficiency of our algorithm when processing *CSV Tables*, as the time complexity scales linearly with the row size of table. (Lines 1-3)

Step 2: Identify relevant columns for the same entity. We use a large public KB – Yago [55]. For each column pair (c_j, c_k), we extract the value pairs of c_j and c_k, find all attributes in KB that can describe these value pairs, and store them in $\mathcal{CP}$. Then, we compute

the score $RS(c_j, c_k, p)$ for each attribute p in KB. If $RS(c_j, c_k, p)$ exceeds the predefined threshold τ_p, c_j and c_k are relevant and we add (c_j, c_k) in $\mathcal{CP}$. (Lines 5-10)

Step 3: Identify relevant columns containing a dependency. We use the dependency discovery algorithm FDEP [21] to extract column pairs from T that satisfy such a dependency and add them in $\mathcal{CP}$. (Lines 11-12)

Step 4: Build a graph using the relevant columns. We construct an edge between each relevant column pair in $\mathcal{CP}$. Note that there are still some independent nodes with no neighbors in the graph. This may affect our ability to identify all relevant columns for the missing column when the missing column is one of the independent nodes. To resolve this, for each independent column in $\mathcal{G}$, we connect it to the column that has identified relevant columns that is *closest* to it, ensuring that relevant columns can be found in the graph. The *closeness* between columns c_j and c_k is defined as $\frac{|set(\pi_{c_j,c_k})|}{|\pi_{c_j,c_k}|}$, where π_{c_j,c_k} represents all value pairs of c_j and c_k, and $set(\pi_{c_j,c_k})$ denotes the set of distinct value pairs between c_j and c_k. (Lines 13-15)

Using the *column-relevance graph* $\mathcal{G}$, all columns in the same connected component [1] as c_j in $\mathcal{G}$ are the relevant columns of c_j.

3.2 Relevant Table Identification

Let $\mathcal{SC}$ denote the relevant columns of column c_j. We extract a sub-table $T_{\mathcal{SC}}$ from T, which is the projection of the columns $\mathcal{SC}$. Then, we identify the table candidates based on $T_{\mathcal{SC}}$. We assume that a table candidate T' should meet two criteria: (1) T' should contain a column with values that overlap with values in c_j, which ensures we are likely to find a value candidate. (2) The table T' should be unionable with the subtable $T_{\mathcal{SC}}$, ensuring that the two tables are relevant.

Based on the first criterion, we first identify a set of columns having overlapping values with c_j and add their corresponding tables to a table candidate pool (§3.2.1). For the second criterion, we identify a set of unionable tables given the sub-table $T_{\mathcal{SC}}$ based on the schemas and columns (§3.2.2). Finally, we propose an algorithm to select the table candidates from the table candidate pool and unionable table set (§3.2.3).

3.2.1 Identifying overlapping columns. We use *Jaccard similarity* [11] to evaluate the overlap between two columns and identify a set of columns whose *Jaccard similarity* to the missing column is greater than a predefined threshold τ_o. The *Jaccard similarity* is computed as the size of the intersection of distinct values from both columns, divided by the size of the union of these distinct values. To boost the efficiency, we build an LSH (Locality-Sensitive Hashing) index [72] for the columns in the data lake to quickly estimate *Jaccard similarity* during the search process for online imputation. After obtaining all of the columns containing overlapping values to the missing column, we add the corresponding tables into the table candidate pool $\mathcal{T}_{op}$.

3.2.2 Finding unionable tables. To identify unionable tables for a subtable, we consider two common types of unionable tables: tables with overlapping schemas and tables that can be unioned based on columns [19, 29].

Tables with overlapping schema. To efficiently find unionable tables based on the schema: (1) *Build a dictionary based on the schema* – We iterate over each table in the data lake, extracting the schema as keys and storing the corresponding table IDs as values; (2) *Select tables from the dictionary* – For a given subtable $T_{\mathcal{SC}}$ with columns $\mathcal{SC}$, iterate over each key (i.e., schema) in the dictionary to check if the key contains all the column names in $\mathcal{SC}$. If it does, we store the table corresponding to the value (table ID) into tables $\mathcal{T}_{su}$.

Tables that can be unioned based on the columns. To identify tables that can be unioned based on the columns, we first iterate over each column c in $\mathcal{SC}$ and find a set of unionable columns for each c. Then we collect all the tables that contain at least one column that is unionable with a column in $\mathcal{SC}$ into tables $\mathcal{T}_{cu}$.

There are two common approaches for identifying unionable columns in a data lake [17]. The first is based on set similarity search, which assumes that unionable columns have many overlapping values. The other is vector similarity search, which generates vectors for the columns and then indexes these vectors. Considering that the real-world columns in the data lake are likely to be noisy, with cell values that might not exactly match but have a similar meaning (e.g., Apple and Apple Inc.) [24], we choose and implement the vector similarity search approach using the following steps:

Step 1: Generating vectors for columns in the data lake. We adopt methods to embed numerical data types and other methods for categorical data types. For numerical columns, we generate a vector using 10 statistical features from the column, including the minimum, maximum, mean, sum, skew, kurtosis, variance, fraction of unique values, and the mean and standard deviation of the number of numerical characters based on the column values, to capture the distribution, value range, and data format information, as described in prior work [26]. For categorical columns, we create a vector to capture the semantic meaning of the column by identifying a specific theme shared by the cell values. For example, the first column with values [Japan, Australia] and the second column with values [China, USA] are both describing the same theme "country". We treat each column as a "paragraph" and its unique values as "words". We then apply a Distributed Bag of Words (DBOW) algorithm [35] to convert the column into a 400-dimensional paragraph vector.

Step 2: Building the index. After processing all of the columns in the data lake into vectors, we build the Hierarchical Navigable Small World (HNSW) index offline separately for numerical and categorical columns. This index is built on the vectors of columns [39], to support efficient searching for similar vectors based on *Euclidean Distance*.

Step 3: Searching for unionable columns based on the index. Given the column c, we convert it to a vector and identify the top-50 columns through the built HNSW index as the unionable columns. We select the top-50 to ensure the coverage for accurately identifying the true unionable columns, as verified through manual labeling and checks.

3.2.3 Table selection. After obtaining the table candidate pool $\mathcal{T}_{op}$, schema-based unionable tables $\mathcal{T}_{su}$, and column-based unionable

tables $\mathcal{T}_{cu}$, we perform the table selection process to select at least k table candidates. We choose the value of k by testing the non-missing values from the missing column, to ensure that as many of them as possible can be found in the retrieved top-k tables. This process consists of the following steps (pseudocode can be found in our technical report [5]):

Step 1: Initializing table candidates. We first select the tables that are in both $\mathcal{T}_{op}$ and $\mathcal{T}_{su}$ as our initial table candidates $\mathcal{T}_{su}^{'}$. This is based on the assumption that the schema-based unionable tables $\mathcal{T}_{su}$ are relevant to our sub-table.

Step 2: Selecting and ranking table candidates. Then, if the number of these tables exceeds k, we directly use them as the table candidates. Otherwise, we consider more tables from the column-based unionable tables $\mathcal{T}_{cu}$. We only keep the tables both in $\mathcal{T}_{op}$ and $\mathcal{T}_{cu}$ and store them in $\mathcal{T}_{cu}^{'}$. For each table T' in $\mathcal{T}_{cu}^{'}$, we compute a table unionability score $U(T_{\mathcal{SC}}, T^{'})$ between T' and $T_{\mathcal{SC}}$, select top-$(k - len(\mathcal{T}_{su}^{'}))$ tables with highest unionability scores, and outputs these tables along with $\mathcal{T}_{cu}^{'}$. $U(T_{\mathcal{SC}}, T^{'})$ is measured by the number of columns in $T^{'}$ that can be mapped to a unionable column in $\mathcal{SC}$.

3.3 Value Identification

Here, we introduce our approach to identifying which value will be used to replace the missing value. Given the missing value $r_i[c_j]$, all values in the overlapping columns of c_j from the fetched table candidates are the value candidates, which could be used to impute $r_i[c_j]$. To select the most promising value from the candidate values, we define a relevance score for each candidate values and select the candidate value with the highest relevance score for the imputed value. Before providing the definition of the relevance score of a candidate value, here we first define *row similarity*, and the relevance score is defined using the row similarity.

Row Similarity. Given the row with missing value r_i from $T_{\mathcal{SC}}$ and another row r from one table candidate T', the row similarity $rs(r_i, r)$ between r_i and r is defined as:

$$rs(r_i, r) = \frac{1}{|M(T_{\mathcal{SC}}, T')|} \sum_{(c,c') \in M(T_{\mathcal{SC}}, T')} vs(r_i[c], r[c'])$$

where $M(T_{\mathcal{SC}}, T')$ stores the column mapping information, i.e., one column in $T_{\mathcal{SC}}$ is mapped which column in T'. $vs(r_i[c], r[c'])$ is the value similarity: (1) if $r_i[c]$ or $r[c']$ is a null value, $vs(r_i[c], r[c']) = 0$; (2) when the value type is numerical, $vs(r_i[c], r[c']) = 1$ if $r_i[c] = r[c']$, otherwise $vs(r_i[c], r[c']) = 0$; (3) when the value type is non-numerical, we treat them as words and calculate the value similarity using n-gram similarity [32]. This can handle real-world noise [24] such as typos, e.g., "Shooting Guard" becoming "Shootung Guard".

Relevance Score. Given a value candidate v for $r_i[c_j]$, the relevance score of v to $r_i[c_j]$ is defined as:

$$\sum_{T' \in \mathcal{T}_s} \sum_{r \in T' \wedge r[c]=v \wedge (c_j,c) \in M(T_{\mathcal{SC}}, T')} rs(r_i, r)$$

where $\mathcal{T}_s$ is the set of the table candidates.

4 THE ACQUISITION-GUIDED ESTIMATION MODULE

Key Issues to Overcome. Existing estimation-based methods [8, 51, 54, 59, 65] perform poorly when filling missing values in small tables, e.g., *HTML Tables*, due to the lack of training data. *Can we use data from other tables to improve imputation performance?* Since similar data exists in the same data lake, we believe it is possible. However, there are still two key limitations: **P1**- Given a large collection of tables in the data lake, how can we effectively discover the most relevant tables to the incomplete table; **P2**- After obtaining a set of candidate tables, how do we efficiently select the most meaningful rows from these tables to improve the imputation accuracy.

Solution Overview. To enhance existing estimation-based methods in the data lake setting, we propose an *acquisitor* in the acquisition-guided estimation module. First, the acquisitor performs table acquisition to identify the most relevant tables, i.e., solving **P1** (§4.1). Second, during the row acquisition process by the acquisitor, we introduce an influence score to evaluate the likelihood that each row will improve the imputation performance, and further exploit the sampling-based strategy to efficiently select a set of rows, i.e., solving **P2** (§4.2).

4.1 Table Acquisition

Given an incomplete table T, our primary goal is to identify a set of relevant tables to enhance the imputation accuracy of the estimation-based methods. Unlike the previous relevant table identification process (§3.2), which focuses on discovering relevant tables based on a subset of columns, we aim to find relevant tables by considering all of the columns. This is because most estimation-based methods [13, 51, 54, 59] use all of the columns to train a model to predict missing values based on column correlations [49]. We leverage our previous method (§3.2.2) to identify schema-based and column-based unionable tables to table T. Following the discussion in Section 3.2.3, if the number of schema-based unionable tables is less than k, we complement the set with the top column-based unionable tables ranked by their unionability scores. For each relevant table $T^{'}$, we align it with T based on the column mappings $M(T, T^{'})$. If a column in T does not have a column mapped in $T^{'}$, we use a NULL value for this column. All aligned relevant tables are then stored in $\mathcal{T}_e$.

4.2 Row Acquisition

The rows from the relevant tables may not have a similar distribution to our incomplete table, raising the question of whether the accuracy of the imputation model will be improved. So, we design a row acquisition process to determine which rows from the candidate tables to use: (1) We introduce an influence score, which represents the potential benefit for each row to enhance the imputation accuracy (§4.2.1); (2) A sampling-based exploration strategy (§4.2.2) is proposed to efficiently select the best rows from a large set of row candidates.

4.2.1 Influence score. To evaluate the likelihood of each row on improving the imputation accuracy, a naive solution is to select a row with the highest accuracy improvement each time when a

row is added to the training data, and retrain the model. However, the process is both time-consuming and resource-intensive. To address this limitation, we introduce an influence score to estimate the impact of each row without retraining the model, based on an influence function [30, 37, 43]. An influence function is a classic technique from statistics that has been effective in obtaining high-quality training data without requiring a model to be retrained [37].

Suppose we have a parametric model $\mathcal{M}$ with parameters θ and a loss function L. The influence score of each training candidate (x_i, y_i) is defined as:

$$I(x_i, y_i) = -H_\theta^{-1} \nabla_\theta L(x_i, y_i, \theta) \quad (1)$$

where $H_\theta = \frac{1}{|\mathcal{D}_{test}|} \sum_{(x,y)\in\mathcal{D}_{test}} \nabla_\theta^2 L(x, y, \theta)$ is the Hessian matrix [58] and $\mathcal{D}_{test}$ is the testing set, which consists of 20% of the training data. The data candidate (x_i, y_i) with a positive influence score $I(x_i, y_i)$ indicates that including it in the training process will reduce the model's loss, thereby improving the accuracy.

4.2.2 An Efficient Sampling-based exploration strategy. When processing a large number of row candidates, it is time-consuming to iterate over each row to calculate the influence score. To improve the efficiency, we propose a sampling-based exploration strategy to select rows. The core idea is to reduce the search space by dividing the rows into a set of small clusters and selecting additional rows from the meaningful clusters. Rows in the same cluster follow a similar data distribution, while rows across different clusters do not. This is because rows with similar distributions are likely to have similar influence scores [31]. Let $\mathcal{R}$ denote all candidate rows, we apply Mini batch K-Means [62] to divide $\mathcal{R}$ into z clusters, denoted as $\{\mathcal{R}_1, \ldots, \mathcal{R}_z\}$, after converting categorical values to numerical values using ordinal encoding [15]. We choose z as the number of clusters which provides the best clustering performance, following [62]. We find that, in certain cases, a large proportion of rows (e.g., 80%) may be assigned to a single cluster, reducing efficiency gains. For such cases, we evenly distribute the rows across 10 clusters to ensure the efficiency improves.

To identify the clusters that provide meaningful rows, we sample a subset of up to 1000 rows from the i-th cluster $\mathcal{R}_i$ (if $|\mathcal{R}_i| \leq 1000$, we use all rows in $\mathcal{R}_i$) and calculate the reliability $Pr(\mathcal{R}_i)$ based on the proportion of rows that have positive influence scores $I(x_j, y_j)$, which is constructed from a row in $\mathcal{R}_i$:

$$Pr(\mathcal{R}_i) = \frac{\sum_{(x_j,y_j)\in\mathcal{R}_i} \mathbb{I}[I(x_j, y_j) > 0]}{|\mathcal{R}_i|} \quad (1)$$

where $\mathbb{I}[\cdot]$ is an indicator function. Next, we select a cluster with the highest reliability as our target cluster and choose the rows with a positive influence score from that cluster. Using this strategy, we can effectively reduce our search space from $|\mathcal{R}|$ to $\frac{|\mathcal{R}|}{z}$.

5 THE CLASSIFIER MODULE

We have introduced two imputation strategies, namely search-based and estimation-based, utilized in our contextual search module (§3) and acquisition-guided estimation module (§4) respectively. However, a missing value can differ in imputation scenarios, table types, and missing value types. Therefore, the final challenge is to effectively utilize the available information to select the most suitable imputation strategy. We resolve this using a learning-based classifier. We construct a set of table-level and column-level statistics as features to help identify which imputation scenario applies (§5.1). Then, we train a classifier to predict the imputation strategy using these features (§5.2).

5.1 Feature Design

Features of the classifier input should effectively capture three key characteristics that will influence the imputation process, imputation scenarios, table types, and data types (§1). We categorize these features into three types: (1) table-level features (§5.1.1), which summarise the different table types and diverse imputation scenarios. (2) column-level features (§5.1.2), which consider different imputation scenarios based on the data type. (3) row-level features which estimates the likelihood of finding a redundant row in the data lake that can provide an exact replacement value, similar to our search module.

5.1.1 Table-level features. For a certain table T, table-level features can be classified into two types:

Features Related to Table Types. We count the number of rows (**F1**) and columns (**F2**) and calculate the percentage of numerical columns (**F3**).

Features Related to Imputation Scenarios. The imputation scenarios (S1- S3) dictate whether we can find relevant tables and similar values from other tables. To encode this knowledge, we count how many tables have the same schema as T (**F4**) and the overlap ratio between T and the other tables in the data lake is computed. Specially, the overlap ratio between two tables (T, T') is calculated using the Jaccard containment w.r.t. T [20]: $\frac{|V(T)\cap V(T')|}{|V(T)|}$, where $V(T)$ and $V(T')$ represent the set of distinct cell values in T and T', respectively. Then, we compute the maximum, mean, and minimum values for all of the overlap ratios as features (**F5 - F7**). Given the number of tables in the data lake, we use the containment min hash algorithm [34] to compute the overlap ratio between two tables efficiently, reducing the total time cost by 10% in our experiments.

5.1.2 Column-level feature. For the given missing column c_j, the columns-level features can also be classified into three categories:

Scenario-Specific Features. We count the number of columns (**F8**) that have a similar name to c_j. We can compute overlap ratios between other columns in the data lake with c_j using Jaccard containment and compute the maximum, mean, and minimum of these overlap ratios as additional features (**F9 - F11**).

Features Related to Data Types. We record the column value type (numerical or categorical) (**F12**), and the column domain size, the number of distinct values in the column, (**F13**).

Column Property Features. We include the number of missing values (**F14**), and the percentage of missing values (**F15**), which can be utilized to avoid the usage of the estimation-based strategy when it contains too many missing values.

We also introduce one additional feature, the maximum overlap ratio at the row-level for each missing value (**F16**). To calculate **F16**, we first identify a table that has the highest overlap ratio to the table that contains the missing value (cf. **F5**). Then, based on

the rows in this table, we gather a collection of sets, where each set consists of the cell values from a single row. We also create a set containing other cell values from the row containing the missing value. Based on this set, we then iterate each set in the collection, to obtain the maximum Jaccard containment between the two sets. This is an important feature because, given a missing value, the appropriate imputation strategy can also be influenced by finding a row that has similar cell values to the row containing the missing value.

5.2 Classifier Selection Strategy

When developing a learning-based classifier to determine the most appropriate imputation strategy, our goals are: (1) support class imbalances, as many training samples may require a single imputation strategy (e.g., 25% belong to estimation-based strategy in our training set), and (2) working effectively with limited training data, as the training data is derived from a few small tables (e.g., around 1000 training samples in our traing set), so we must support few-shot training. To achieve these goals, we apply a supervised KNN classifier [69], which can effectively handle class imbalances and adapt to small training set. Next, we introduce the details of our KNN classifier and describe how we train and use this classifier for inference.

5.2.1 KNN classifier. The KNN classifier is a non-parametric algorithm that classifies each test sample based on proximity to the labeled training samples. Given an input feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{D}_{train}| \times d}$ and a label vector $\mathbf{Y} \in \mathbb{R}^{|\mathcal{D}_{train}|}$ from the training set, where $|\mathcal{D}_{train}|$ represents the number of training samples and d is the dimension of each feature vector, it builds an index over $\mathbf{X}$. During the inference phase, given a test sample represented by a feature vector $\mathbf{x} \in \mathbb{R}^d$, it then performs the following steps:

(1) **Calculating Distances**: Compute the Euclidean distance [3] between the test sample $\mathbf{x}$ and the training samples $\mathbf{X}$. The Euclidean distance between the test sample $\mathbf{x}$ and the i-th training sample $\mathbf{x}_i \in \mathbf{X}$ is given by: $d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{d} (\mathbf{x}_j - \mathbf{x}_i^j)^2}$ where $\mathbf{x}_j$ is the j-th element of the test vector and $\mathbf{x}_i^j$ is the j-th element of the i-th training vector.
(2) **Selecting Neighbors**: After computing the distances between the test sample and all training samples, it then finds the K training samples with the smallest distances. These K samples are the nearest neighbors.
(3) **Majority Voting**: Finally, predicting the label for the test sample is accomplished using the majority vote from the labels of the nearest neighbors.

5.2.2 Training and Inference. To train our classifier, we first construct a training set $(\mathbf{X}, \mathbf{Y})$, by randomly sampling non-missing values for each missing column and removing them. For these newly introduced missing values, we generate the labels $\mathbf{Y}$ by comparing which imputed value from our search and estimation modules to the closest ground truth values. Here, "closest" means that the imputed value minimizes the absolute error relative to the ground truth value for numerical values, or uses exact matching for categorical values.

We construct a feature matrix $\mathbf{X}$ based on our proposed features. A straightforward approach is to represent each sample using a feature vector $[\mathbf{F1}, \cdots, \mathbf{F16}]$, concatenating all feature vectors into a matrix, and then normalising it. However, this approach can lead to poor classification results when using a KNN classifier, as it fails to consider the varying importance of each feature. To solve this problem, we align the features into the same scale by grouping values based on the mutual information [52] between the bins and the labels. Specifically, we convert each feature into discrete bins using a supervised discretization method [47]. For example, given the feature values [6, 8, 16] from **F9**, the discretized feature would be [1, 1, 2]. To make predictions using the trained classifier, we convert each missing value into a normalised feature vector $\mathbf{x}$, following the same process as in the training data, and input it into the classifier to get the predicted label using majority voting.

6 EXPERIMENTAL EVALUATION

We conduct extensive experiments to study four questions: (Q1) How does CESID perform on real-world data lakes in a variety of table types and data types of missing value? (Q2) How does CESID handle a variety of "source of imputed value"? (Q3) How do the three components designed in CESID contribute to effectiveness and efficiency? (Q4) Can CESID complete the imputation efficiently with acceptable storage usage?

6.1 Experimental Setup

6.1.1 Data Lakes. We conduct experiments on EntiTables [70], WebTables [17], and OpenData [17]. The first two are *HTML Tables* while the last is *CSV Tables*. For EntiTables, we use the same missing value test presented in previous studies [22, 70]. Since WebTables and OpenData do not contain any test missing values, we create test missing values by randomly injecting missing values for these two data lakes. Specifically, we uniformly sample 10,000 and 5,000 non-missing values with numerical and categorical values from OpenData and WebTable, respectively. The sample sizes are chosen based on the size of the rows and tables. These values are then masked as the test missing values. We show the properties of our data lakes in Table 1. It first provides key statistical information about the data lake, then highlights the variety in terms of table types, source of imputed value, and data types of missing value, and finally presents statistics about missing values.

Training set construction. For each data lake, we construct training data for both the acquisitor (§4.2) and the classifier (§5.2). For training examples in the acquisitor, which are used to train a parametric model, we randomly select a maximum of 500 rows that have no missing values in each missing column. Based on these rows, the values from the columns other than the missing one are converted to features, while the values from the missing column provide as the labels. For the training examples in the classifier, we first randomly select non-missing values from each missing column, with the same percentage of missing values in that column, to ensure equal missing probability. Then we find the appropriate imputation strategy for each non-missing value to be the labels, and compute the corresponding statistics as features, as detailed in §5.2.2.

Table 1: Properties of Data Lake and Missing values.

Data Lake	EntiTables	WebTable	OpenData
Avg rows	11.4	23.0	79,310.7
Avg columns	5.1	6.5	16.1
# Tables	1,579,117	2,772,082	7,690
Table type	HTML Tables	HTML Tables	CSV Tables
Scenario S1/S2/S3[1]	35%/73%/27%	69%/99%/1%	59%/55%/28%
Value type Num./Cat.	9%/91%	86%/14%	73%/27%
# Missing Num./Cat. vals.	90/882	4,252/748	6,759/3,241
# Incomplete tables	187	178	49

[1] A single missing value can belong to both S1 and S2 scenarios.

Table 2: Imputation error for numerical values. [1]

Cate.[2]	Method	EntiTables		WebTable		OpenData	
		MAE	RMSE	MAE	RMSE	MAE	RMSE
E	MICE	0.271	0.414	0.119	0.191	0.071	0.190
	GAIN	0.304	0.433	0.148	0.308	0.085	0.298
	NOMI	0.446	0.626	0.271	1.071	0.073	0.183
	MissForest	0.302	0.442	0.149	0.235	0.067	0.168
	Datawig	0.461	0.589	0.157	0.244	0.070	0.185
S	RetClean	6	23	78	1,227	19,878	796,035
	RATA	32	59	15	498	3,372	117,288
H	CESID	**0.253**	**0.389**	**0.095**	**0.187**	**0.065**	**0.167**

[1] The best results are in bold and the second-best are underlined.
[2] In the Cate. column, E denotes estimation-based approaches, S denotes search-based approaches, and H represents our hybrid approach.

6.1.2 Baselines. We choose the baselines from both estimation-based and search-based methods (§2.2). For estimation-based methods, we include the classical methods MICE [51] and MissForest [54], and the most recent models GAIN [65], Datawig [8], and NOMI [59]. For search-based methods, we adopt the latest approaches RetClean [6] and RATA [22]. Note that MICE, GAIN, and NOMI only support numerical values. We exclude GRIMP [13] due to its GPU memory usage exceeding the maximum memory size of our available devices; we exclude Auto-completion [70] since there is no publicly (or privately) available implementation.

6.1.3 Evaluation metrics. Following the methodology in previous work [44], we use RMSE [27] and MAE [27] to evaluate the imputation accuracy for numerical data types. For categorical data types, we use the accuracy as defined in [13], which computes the percentage of imputed values that match the ground truth value.

6.1.4 Implementation & Hardware. For MICE and MissForest, we use the implementation from HyperImpute[3], and use the default parameters. For GAIN, Datawig, RATA, and NOMI, we use the original source code and the suggested hyperparameters from the original papers. For RetClean, we use the most cost-effective version, which can be locally deployed using a RoBERTa-based foundation model [38]. To ensure a fair comparison with existing estimation-based methods, we implement the estimation method in our Acquisition-guided Estimation module by selecting the best-performing estimation-based method for each data lake and data type of missing value. We set our parameters τ_p (§3.1) to 0.1, and τ_o (§3.2) to 0.05.

All code is implemented in python and available at [4]. All experiments were conducted on a server running Red Hat 7.9, equipped with an Intel(R) Xeon(R) E5-2690 CPU, 512GB RAM, and an NVIDIA Tesla P100 GPU with 16GB of memory.

[3] https://github.com/vanderschaarlab/hyperimpute

Table 3: Imputation accuracy on categorical values.

Category	Method	EntiTables	WebTable	OpenData
E	MissForest	0.168	0.195	0.317
	Datawig	0.158	0.138	0.220
S	RetClean	0.168	0.017	0.037
	RetClean (SS)[1]	0.194	0.213	0.041
	RATA	0.113	0.080	0.020
	RATA (SS)[1]	0.133	0.244	0.024
H	CESID	**0.268**	**0.449**	**0.427**

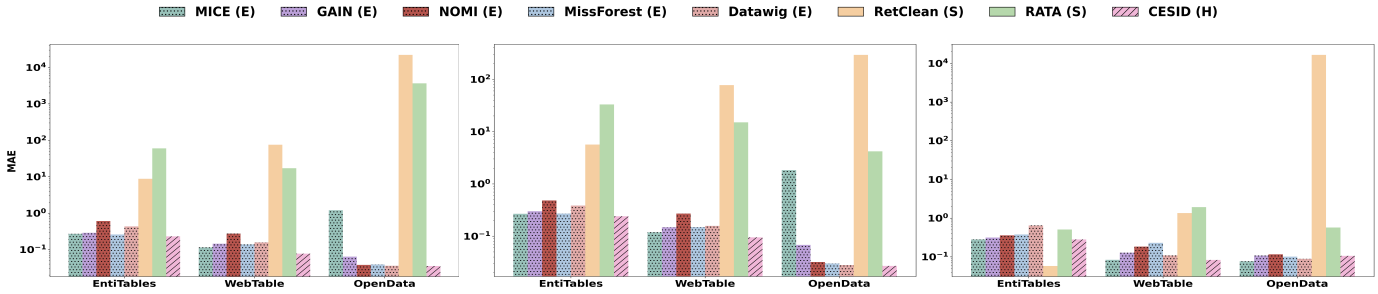
[1] Imputation accuracy evaluated on categorical values from incomplete tables that have tables with the same schema.

6.2 Evaluation on Table and Data Types of Missing Value (Q1)

In this subsection, we analyze the imputation accuracy across two different table types. For each data lake, we report results based on the data type of missing value, namely numerical (see Section 6.2.1) or categorical (see Section 6.2.2), respectively.

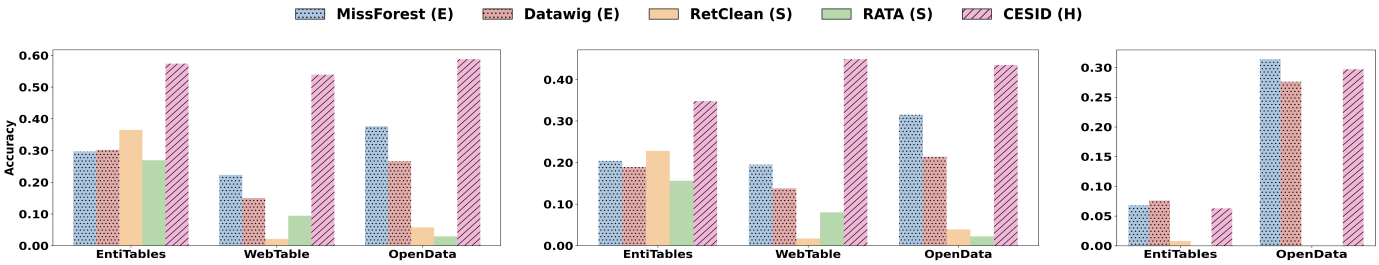
6.2.1 Imputation performance for numerical data. Table 2 shows the RMSE and MAE for all of the compared methods on three data lakes. Observe that: (1) Overall, CESID outperfoms all other imputation methods for RMSE and MAE on all data lakes. For example, in terms of MAE, CESID outperforms the second-best baseline on EntiTables, WebTable, and OpenData by 7%, 20%, and 3% respectively; (2) Estimation-based methods perform better on *CSV Tables* than on *HTML Tables*. The accuracy of estimation-based methods relies on sufficient training data to train an effective model. However, *HTML Tables* are typically too small to provide enough training data, resulting in higher error rates during imputation; (3) Search-based methods perform the worst on all data lakes. Since these approaches treat tables as text and identify potential replacement values using text-based embedding similarity, which can work for categorical values in *HTML Tables*. The error rate is even greater on *CSV Tables*, as seen in OpenData. This demonstrates that search-based methods rely on precise embedding similarity and struggle with filling tables with numerical columns.

6.2.2 Imputation performance for categorical data. Table 3 shows the accuracy of each method when imputing missing values of categorical data. Observe that: (1) CESID consistently outperforms all other baselines for all data lakes. For instance, CESID achieves an accuracy improvements of 59%, 130%, and 35% compared to the second-best method for EntiTables, WebTables, and OpenData, respectively. (2) Search-based methods perform worse, particularly on WebTable and OpenData. This low performance can be attributed to two reasons. First, search-based methods are effective at identifying potential missing values from related tables that have the same schema [70] However, only 18% of incomplete tables in WebTable



(a) Scenario S1 (b) Scenario S2 (c) Scenario S3

Figure 3: Imputation performance of numerical values over three imputation scenarios.



(a) Scenario S1 (b) Scenario S2 (c) Scenario S3

Figure 4: Imputation performance of categorical values for three imputation scenarios.

and 10% in OpenData return tables using the same schema. In contrast, it 65% of the tables in EntiTables have similar schemas. Second, search-based methods rely on embedding-based similarity and struggle to deal with tables with many unique values. This issue is evident when looking at the accuracy when imputing missing categorical values using incomplete tables that have tables with the same schema, i.e., RetClean (SS) and RATA (SS). Both methods achieved a 10% accuracy improvement on EntiTables and WebTable, but their accuracy dropped by 5% on OpenData. (3) Estimation-based methods perform significantly better on OpenData compared to EntiTables and WebTable. With sufficient training data for *CSV Tables*, these approaches can produce a more effective model. However, there are still challenges when imputing categorical values, even on OpenData. Based on a detailed failure analysis, we find that categorical values in OpenData are too sparsely distributed. Specifically, for the ground truth of the missing categorical values, on average only 22% of the training data have the same values. Estimation-based methods, which impute categorical values using classification, often struggle when imputing missing values that appear less frequently in the training data.

6.3 Evaluation on Source of Imputed Value (Q2)

Here, we evaluate the imputation performance based on the variety of "source of imputed value", by dividing our missing test values into three imputation scenarios: S1, S2, and S3 (§1). For categorization, we use the following rules. For S1, the ground truth of these missing values can be found from the candidate values retrieved using our proposed search method (§3). For S2, the estimated values for these missing values from our estimation method (§4), generally fall in the same range or value set for the corresponding missing columns. For S3, it includes the values that do not fall under S1 or S2.

6.3.1 Imputing numerical values in diverse scenarios. The imputation accuracy for numerical values across three imputation scenarios is shown in Figure 3. For the legend, the letter in parentheses indicates the method type: (E) denotes an estimation-based method, (S) denotes a search-based method, and (H) represents our proposed hybrid method. Observe that: (1) CESID outperforms existing imputation methods in S1 and S2. On average, there is a 13% improvement in S1 and an 11% improvement in S2 across the three data lakes compared to the second-best methods; (2) CESID is the best or second-best one for S3. It is interesting to see that RetClean achieves the highest accuracy when filling missing numerical values in S3, in the EntiTables. On EntiTables, RetClean extract numerical values from columns with similar names to the missing column. Even though the tables containing these columns are not relevant to the incomplete table, the extracted values have low error rates when compared with the ground truth values, as observed when analyzing these 24 missing numerical values in S3. (3) Search-based methods generally perform the worst in every scenario, even in the search-related scenario S1. They identify potential values from the row with the highest text-based similarity. However, by directly converting each row into a sentence and then using a pre-trained language model to create text embeddings, they ignore properties of numerical values in the row, resulting in the wrong rows being retrieved, and fail to correctly identify the missing values.

6.3.2 Imputing categorical values in diverse scenarios. Figure 4 illustrates the imputation accuracy for categorical data. Observe that:

Table 4: Ablation study for Contextual Search Module. [1]

Variant	EntiTables				WebTable				OpenData			
	Numerical		Categorical		Numerical		Categorical		Numerical		Categorical	
	MAE	Time (s)	Accuracy	Time (s)	MAE	Time (s)	Accuracy	Time (s)	MAE	Time (s)	Accuracy	Time (s)
Contextual Search	0.371	1.300	0.252	1.990	0.211	1.190	0.390	0.740	0.265	1.939	0.410	1.270
w/o Graph	1.109	1.309	0.229	1.995	0.548	1.192	0.373	0.803	3.792	2.102	0.384	1.277
Textual Similarity	7.189	1.440	0.178	2.070	0.288	1.294	0.345	0.803	1.095	4.603	0.204	7.050

[1] "Time (s)" shows the average online imputation time per missing value. The online imputation time is the total time spent on the search process for the evaluated missing values.

Table 5: Ablation study for Acquisition-guided Estimation Module. [1]

Method	EntiTables				WebTable				OpenData			
	Numerical		Categorical		Numerical		Categorical		Numerical		Categorical	
	MAE	Time (s)	Accuracy	Time (s)	MAE	Time (s)	Accuracy	Time (s)	MAE	Time (s)	Accuracy	Time (s)
Acquisition-guided Estimation	0.122	0.78 (0.77)	0.223	1.57 (0.77)	0.129	0.52 (0.51)	0.195	1.03 (0.22)	0.029	2.00 (1.37)	0.323	3.06 (1.27)
w/o Sampling	0.120	0.82 (0.81)	0.224	1.66 (0.86)	0.124	0.61 (0.60)	0.201	1.05 (0.24)	0.029	2.71 (2.06)	0.323	3.85 (2.06)
w/o Acquisitor	0.130	0.01	0.202	0.80	0.134	0.01	0.189	0.81	0.029	0.65	0.315	1.79

[1] "Time (s)" shows the average online imputation time per missing value, with the external time cost used by the acquisitor in parentheses. The online imputation time is the total time spent on the estimation process for the evaluated missing values.

(1) In scenario S1, CESID significantly improves accuracy compared to existing search-based methods. It even achieves up to 10 times higher accuracy than RetClean on OpenData. This highlights the ability of CESID to address the limitations of search-based methods in imputing categorical values in CSV Tables. (2) In scenario S2, CESID achieves higher imputation accuracy than that of estimation-based methods. For example, by acquiring more training data on EntiTables and WebTable, i.e., *HTML Tables*, CESID shows an average improvement of up to 100%. (3) In scenario S3, estimation-based methods, MissForest and Datawig, achieve the better accuracy while CESID remains competitive. However, search-based methods perform worse because they cannot find an exact match or a good approximate match in the other tables. With our classifier, CESID can effectively select the best strategy when doing the imputation. (4) For methods utilizing search-based imputation strategies, i.e., RetClean, RATA, and CESID, they perform better on scenario S1 compared to S2 and S3, except for RetClean and RATA on the WebTable. By observing the missing categorical values in WebTable, we find that most false imputations occur in incomplete tables whose schema cannot be found from other tables. This demonstrates that the effectiveness of existing search-based methods may depend on the schema overlap.

6.4 Ablation Study (Q3)

We conduct an ablation study to evaluate the impact of key designs in CESID, including the *column-relevance graph* (§3.1) and row similarity (§3.3) in the search module, the acquisitor (§4) in the Acquisition-guided Estimation module, and KNN classifier (§5.2) in the Classifier module.

6.4.1 Contextual search module. We compare three different search variants: (1) *Contextual Search*, which is our proposed search module; (2) *W/O Graph*, which retrieves relevant tables based on all columns of one table instead of only relevant to the missing value; (3) *Text Similarity*, which replaces our proposed row similarity with the textual similarity. We convert each row into text as in RetClean [6], embed the text using a sentence-transformer[4], and use cosine similarity [2] as the textual similarity. We compare these variants based on the imputed values that can be identified by our proposed search module (§3).

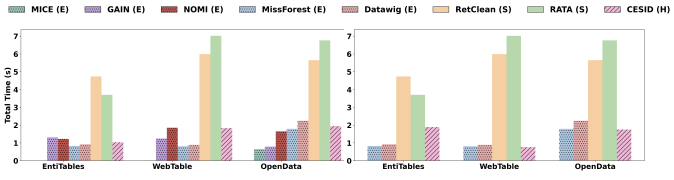
From Table 4, observe that: (1) *column-relevance graph* improves both imputation accuracy and online imputation efficiency. For example, for OpenData, substantial improvements in filling numerical values can be seen, with a 93% reduction in MAE and an 8% decrease in imputation time. This is because incomplete tables with these numerical missing values have large column sizes – 28 columns on average. For *column-relevance graph*, CESID can efficiently identify relevant column subsets and filter out the columns that may have a negative impact. (2) Our proposed row similarity method is more efficient than text-based similarity, especially for OpenData. This is because converting rows into embeddings is time-consuming as the number of rows increases. Despite the time savings, our row similarity also improves imputation accuracy. The similarity is computed using the cell values and data types, which is more reliable for structured tables. In contrast, text-based similarity treats each row as a text sequence, ignoring the inherent structure of the table and the unordered cell values.

6.4.2 Acquisition-guided estimation module. We compare three different estimation variants: (1) *Estimation*, which is our proposed estimation method; (2) *W/O Sampling*, which explores all the rows from other tables without sampling; (3) *W/O Acquisitor*, which directly adopts existing estimation-based methods. We test these variants on the imputed values that are estimated by incorporating

[4]See https://huggingface.co/sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2. In contrast to the text embedding model used by RetClean and RATA, we implement a model that is more commonly used in semantic search tasks.

Table 6: Ablation study for Classifier module.

Method	EntiTables		WebTable		OpenData	
	MAE	Accuracy	MAE	Accuracy	MAE	Accuracy
CESID	0.253	0.268	0.095	0.449	0.065	0.427
Oracle	0.233	0.304	0.068	0.509	0.051	0.480



(a) Numerical value. **(b) Categorical value.**

Figure 5: Time required to fill a missing value (in seconds).[5]

meaningful rows selected from other tables (§4).. From Table 5, observe that: (1) The proposed sampling-based exploration strategy improves efficiency by 10% on average for online imputation of numerical and categorical values across the three data lakes, with only a 1% reduction in performance. ; (2) The acquisitor in our Acquisition-guided Estimation module enhances imputation accuracy and efficiently obtains external rows from other tables, requiring at most 1.37 seconds per missing value.

6.4.3 Classifier module. This study focuses on evaluating the effectiveness of our classifier module. We compare CESID with an Oracle, which selects the more accurate value from the search and estimation result, i.e., the one closer to the ground truth for each missing value. The results are shown in Table 6. We can see that even our learned classifier can reliably predict the best method to use for imputation, achieving comparable accuracy to Oracle.

6.5 Efficiency and Storage (Q4)

6.5.1 Online imputation efficiency. Figure 5 shows the average time in imputing one missing numerical and categorical value. Observe that: (1) Overall, CESID is much faster than search-based methods and is comparable to estimation-based methods; (2) RetClean and RATA have the greatest imputation time. This is because both methods need to retrieve values from millions even billions of rows in the data lake, and the search time complexity remains logarithmic relative to the total number of rows in the data lake. However, these estimation-based methods only need to call the estimation model, leading to a short imputation time.

6.5.2 Offline preprocessing time. The results are shown in Table 7. We compared the time overhead of CESID with the SOTA search-based method, RATA, and the SOTA estimation-based method, Datawig. For CESID, we report the time for offline construction of LSH and HNSW indexes[6] (§3.2). For RATA, we include the time used to construct the embeddings and building indexes for the row embeddings under the same parallel setup. For Datawig, we record the offline training time. From the results, observe that: (1) Overall, Datawig has the smallest or second-smallest processing time, since it only trains a single model; (2) Both RATA and Datawig require more time when the size of the data lake increases, while CESID exhibits a different trend. This is because RATA and Datawig build indexes row-wise, with a time complexity of $O(\mathcal{A} \cdot \log \mathcal{A})$, while CESID builds indexes column-wise, with a time complexity of $O(\mathcal{B} \cdot \log \mathcal{B})$, where $\mathcal{A}$ and $\mathcal{B}$ represent the total number of rows and columns in the data lake, respectively. For example, on OpenData (~610M rows and ~123K columns), CESID is the most efficient approach.

[5]The time required to fill numerical values in the EntiTables and WebTable is approximately 0.01 seconds, so the bar may not be visible in the figure.

[6]Given the large number of tables in each data lake, which could increase our preprocessing time, we used 30 parallel processes to build the indexes simultaneously for RATA and CESID.

Table 7: Comparison of offline process time and storage with the representative imputation methods.

Method	Offline process time (Hour)			Storage (GB)		
	EntiTables	WebTable	OpenData	EntiTables	WebTable	OpenData
CESID	0.97	2.33	0.10	30	64	0.45
RATA [1]	0.92	2.50	31	26	84	1350
Datawig [2]	0.02	0.06	1.73	0.08	0.78	1.50

[1] RATA is selected as the SOTA search-based imputation method.

[2] Datawig is selected as the SOTA estimation-based method which can handle both numerical and categorical values, similar to CESID.

6.5.3 Storage usage. Table 7 compares of the storage usage of CESID, the SOTA search-based method RATA, and the SOTA estimation-based method Datawig. For CESID, storage usage of the constructed LSH and HNSW indexes is reported. For RATA, we report the storage of the ANN index [22]. For Datawig, we report the storage usage of the pretrained parametric model. Our results show that CESID is more storage-efficient than RATA, especially for OpenData. This is because the storage requirements for CESID are based on the number of columns in the data lake, whereas RATA's storage is based on the number of rows. Moreover, Datawig has the lowest storage requirements among all the methods.

7 CONCLUSION

In this paper, we propose a novel approach CESID, which uses a combination of estimation-based and search-based methods for missing value imputation in data lakes. To enhance the effectiveness, CESID proposes three core modules: the Contextual Search Module, the Acquisition-guided Estimation Module, and the Classifier Module. The Contextual Search Module constructs a graph to leverage contextual relevance for imputing missing values and efficiently identifies candidate values from other tables. The Acquisition-guided Estimation Module introduces an influence function and a sampling-based exploration strategy to efficiently identify rows from other tables, providing a more accurate estimate value. The Classifier Module employs a learned classifier that exploits table-level and column-level statistics to automatically select which method to use and then perform the imputation using the corresponding module. Extensive experiments using three data lakes demonstrate the effectiveness and efficiency of CESID in incorporating the three aspects of variety for imputation, when compared to several state-of-the-art imputation approaches.

REFERENCES

[1] [n.d.]. Connected Components Algorithm. https://graphstream-project.org/doc/Algorithms/Connected-Components/.

[2] [n.d.]. Cosine Similarity. https://en.wikipedia.org/wiki/Cosine_similarity.

[3] [n.d.]. Euclidean distance. https://en.wikipedia.org/wiki/Euclidean_distance.

[4] [n.d.]. Our Codes. https://anonymous.4open.science/r/mv_imputation_datalake-C712/.

[5] [n.d.]. Our Technical Report. https://anonymous.4open.science/r/mv_imputation_datalake-C712/technique_report.pdf.

[6] Mohammad Shahmeer Ahmad, Zan Ahmad Naeem, Mohamed Eltabakh, Mourad Ouzzani, and Nan Tang. 2023. RetClean: Retrieval-Based Data Cleaning Using Foundation Models and Data Lakes. *arXiv preprint arXiv:2303.16909* (2023).

[7] Ahmad Ahmadov, Maik Thiele, Julian Eberius, Wolfgang Lehner, and Robert Wrembel. 2015. Towards a hybrid imputation approach using web tables. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*. IEEE, 21–30.

[8] Felix Biessmann, Tammo Rukat, Philipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing Value Imputation for Tables. *J. Mach. Learn. Res.* 20, 175 (2019), 1–6.

[9] Felix Biessmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. 2018. " Deep" Learning for Missing Value Imputationin Tables with Non-numerical Data. In *Proceedings of the 27th ACM international conference on information and knowledge management*. 2017–2025.

[10] Bernardo Breve, Loredana Caruccio, Vincenzo Deufemia, Giuseppe Polese, et al. 2022. Imputation of Missing Values through Profiling Metadata. In *CEUR WORKSHOP PROCEEDINGS*, Vol. 3194. CEUR-WS, 141–148.

[11] Andrei Z Broder. 1997. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. IEEE, 21–29.

[12] Svetlana Bryzgalova, Sven Lerner, Martin Lettau, and Markus Pelger. 2022. Missing financial data. *Available at SSRN* 4106794 (2022).

[13] Riccardo Cappuzzo, Saravanan Thirumuruganathan, and Paolo Papotti. 2024. Relational Data Imputation with Graph Neural Networks. In *EDBT/ICDT 2024, 27th International Conference on Extending Database Technology*. 221–233.

[14] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. Seed: Domain-specific data curation with large language models. *arXiv e-prints* (2023), arXiv–2310.

[15] Mwamba Kasongo Dahouda and Inwhee Joe. 2021. A deep-learned embedding technique for categorical features encoding. *IEEE Access* 9 (2021), 114381–114391.

[16] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record* 51, 1 (2022), 33–40.

[17] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, et al. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1925–1938.

[18] Tlamelo Emmanuel, Thabiso Maupong, Dimane Mpoeleng, Thabo Semong, Banyatsang Mphago, and Oteng Tabona. 2021. A survey on missing data in machine learning. *Journal of Big data* 8 (2021), 1–37.

[19] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée Miller. 2022. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *arXiv preprint arXiv:2210.01922* (2022).

[20] Raul Castro Fernandez, Jisoo Min, Demitri Nava, and Samuel Madden. 2019. Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1190–1201.

[21] Peter A Flach and Iztok Savnik. 1999. Database dependency discovery: a machine learning approach. *AI communications* 12, 3 (1999), 139–160.

[22] Michael Glass, Xueqing Wu, Ankita Rajaram Naik, Gaetano Rossiello, and Alfio Gliozzo. 2023. Retrieval-Based Transformer for Table Augmentation. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics, Toronto, Canada, 5635–5648. https://doi.org/10.18653/v1/2023.findings-acl.348

[23] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12571–12590.

[24] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and S Yu Philip. 2023. Automatic table union search with tabular representation learning. In *Findings of the Association for Computational Linguistics: ACL 2023*. 3786–3800.

[25] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. 2023. Gittables: A large-scale corpus of relational tables. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–17.

[26] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. 2019. Sherlock: A deep learning approach to semantic data type detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1500–1508.

[27] Shawn R Jeffery, Minos Garofalakis, and Michael J Franklin. 2006. Adaptive cleaning for RFID data streams. In *Vldb*, Vol. 6. Citeseer, 163–174.

[28] Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. 2020. Learning numeral embedding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2586–2599.

[29] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[30] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[31] Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. 2019. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems* 32 (2019).

[32] Grzegorz Kondrak. 2005. N-gram similarity and distance. In *International symposium on string processing and information retrieval*. Springer, 115–126.

[33] Stavros Konstantinopoulos, Genevieve M De Mijolla, Joe H Chow, Hanoch Lev-Ari, and Meng Wang. 2020. Synchrophasor missing data recovery via data-driven filtering. *IEEE Transactions on Smart Grid* 11, 5 (2020), 4321–4330.

[34] David Koslicki and Hooman Zabeti. 2019. Improving minhash via the containment index with applications to metagenomic analysis. *Appl. Math. Comput.* 354 (2019), 206–215.

[35] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *International conference on machine learning*. PMLR, 1188–1196.

[36] Jiang Li, Xiaowei S Yan, Durgesh Chaudhary, Venkatesh Avula, Satish Mudiganti, Hannah Husby, Shima Shahjouei, Ardavan Afshar, Walter F Stewart, Mohammed Yeasin, et al. 2021. Imputation of missing values for electronic health record laboratory data. *NPJ digital medicine* 4, 1 (2021), 147.

[37] Jiabin Liu, Fu Zhu, Chengliang Chai, Yuyu Luo, and Nan Tang. 2021. Automatic data acquisition for deep learning. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2739–2742.

[38] Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[39] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[40] Michael May, Christine Körner, Dirk Hecker, Martial Pasquier, Urs Hofmann, and Felix Mende. 2009. Handling missing values in gps surveys using survival analysis: a gps case study of outdoor advertising. In *Proceedings of the Third International Workshop on Data Mining and Audience Intelligence for Advertising*. 78–84.

[41] Patrick E McKnight, Katherine M McKnight, Souraya Sidani, and Aurelio Jose Figueredo. 2007. *Missing data: A gentle introduction*. Guilford Press.

[42] Yinan Mei, Shaoxu Song, Chenguang Fang, Haifeng Yang, Jingyun Fang, and Jiang Long. 2021. Capturing semantics for imputation with pre-trained language models. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 61–72.

[43] Xiaoye Miao, Yangyang Wu, Lu Chen, Yunjun Gao, Jun Wang, and Jianwei Yin. 2021. Efficient and effective data imputation with influence functions. *Proceedings of the VLDB Endowment* 15, 3 (2021), 624–632.

[44] Xiaoye Miao, Yangyang Wu, Lu Chen, Yunjun Gao, and Jianwei Yin. 2022. An experimental survey of missing data imputation algorithms. *IEEE Transactions on Knowledge and Data Engineering* (2022), 5737–5738.

[45] Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. 2021. Generative semi-supervised learning for multivariate time series imputation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 8983–8991.

[46] Mashaal Musleh and Mohamed F Mokbel. 2023. Kamel: A Scalable BERT-based System for Trajectory Imputation. *Proceedings of the VLDB Endowment* 17, 3 (2023), 525–538.

[47] Guillermo Navas-Palencia. 2020. Optimal binning: mathematical programming formulation. *arXiv preprint arXiv:2001.08025* (2020).

[48] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment* 8, 10 (2015), 1082–1093.

[49] Massimo Perini and Milos Nikolic. 2024. In-Database Data Imputation. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–27.

[50] Jingyi Qiu, Aibo Song, Jiahui Jin, Tianbo Zhang, Jingyi Ding, Xiaolin Fang, and Jianguo Qian. 2023. Dependency-Aware Core Column Discovery for Table Understanding. In *International Semantic Web Conference*. Springer, 159–178.

[51] Patrick Royston and Ian R White. 2011. Multiple imputation by chained equations (MICE): implementation in Stata. *Journal of statistical software* 45 (2011), 1–20.

[52] Ricardo Salazar-Díaz, Boris Glavic, and Tilmann Rabl. 2024. InferDB: In-Database Machine Learning Inference Using Indexes. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1830–1842.

[53] Pegdwendé Sawadogo and Jérôme Darmont. 2021. On data lake architectures and metadata management. *Journal of Intelligent Information Systems* 56, 1 (2021), 97–120.

[54] Daniel J Stekhoven and Peter Bühlmann. 2012. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics* 28, 1 (2012), 112–118.

[55] Fabian M Suchanek, Mehwish Alam, Thomas Bonald, Lihu Chen, Pierre-Henri Paris, and Jules Soria. 2024. Yago 4.5: A large and clean knowledge base with a rich taxonomy. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 131–140.

[56] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2021. RPT: relational pre-trained transformer is almost all you need towards democratizing data preparation. *Proceedings of the VLDB Endowment* 14, 8 (2021), 1254–1261.

[57] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems* 34 (2021), 24804–24816.

[58] William Carlisle Thacker. 1989. The role of the Hessian matrix in fitting models to measurements. *Journal of Geophysical Research: Oceans* 94, C5 (1989), 6177–6196.

[59] Jianwei Wang, Ying Zhang, Kai Wang, Xuemin Lin, and Wenjie Zhang. 2024. Missing Data Imputation with Uncertainty-Driven Network. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–25.

[60] Xu Wang, Hongbo Zhang, Pengkun Wang, Yudong Zhang, Binwu Wang, Zhengyang Zhou, and Yang Wang. 2023. An observed value consistent diffusion model for imputing missing values in multivariate time series. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2409–2418.

[61] Yue Wang and Yeye He. 2017. Synthesizing mapping relationships using table corpus. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1117–1132.

[62] Ziheng Wang, Bowei Duan, Qian Zhang, Chuan Li, and Xiaozhen Li. 2022. High-dimensional power enterprise employee clustering based on convolutional neural network and mini-batch K-means algorithm. In *Fifth International Conference on Mechatronics and Computer Technology Engineering (MCTE 2022)*, Vol. 12500. SPIE, 1410–1419.

[63] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.

[64] Li Yang, Qifan Wang, Zac Yu, Anand Kulkarni, Sumit Sanghai, Bin Shu, Jon Elsas, and Bhargav Kanagal. 2022. MAVE: A product dataset for multi-source attribute value extraction. In *Proceedings of the fifteenth ACM international conference on web search and data mining*. 1256–1265.

[65] Jinsung Yoon, James Jordon, and Mihaela Schaar. 2018. Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*. PMLR, 5689–5698.

[66] Haitao Yuan, Gao Cong, and Guoliang Li. 2024. Nuhuo: An Effective Estimation Model for Traffic Speed Histogram Imputation on A Road Network. *Proceedings of the VLDB Endowment* 17, 7 (2024), 1605–1617.

[67] Xiaocan Zeng, Pengfei Wang, Yuren Mao, Lu Chen, Xiaoze Liu, and Yunjun Gao. 2024. MultiEM: Efficient and Effective Unsupervised Multi-Table Entity Matching. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 3421–3434.

[68] Meihui Zhang and Kaushik Chakrabarti. 2013. Infogather+ semantic matching and annotation of numeric and time-varying attributes in web tables. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 145–156.

[69] Shichao Zhang. 2021. Challenges in KNN classification. *IEEE Transactions on Knowledge and Data Engineering* 34, 10 (2021), 4663–4675.

[70] Shuo Zhang and Krisztian Balog. 2019. Auto-completion for data cells in relational tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 761–770.

[71] Zhongwei Zhang, Lei Cao, Xiliang Chen, Wei Tang, Zhixiong Xu, and Yangyang Meng. 2020. Representation learning of knowledge graphs with entity attributes. *IEEE Access* 8 (2020), 7435–7441.

[72] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.

A TIME COMPLEXITY ANALYSIS

We analyze the time complexity of offline preprocessing and online imputation in CESID.

A.1 Offline Preprocessing

There are two key offline preprocessing steps:

Building Indexes. To discover relevant tables for the search and estimation modules, we need to pre-build the LSH and HNSW indexes. The time complexity for building the LSH indexes is $O(|\mathcal{B}| \cdot \overline{|C|})$, and for the HNSW index, it is $O(|\mathcal{B}| \cdot \log|\mathcal{B}|)$. Here, $\mathcal{B}$ represents the columns in the data lake, and $\overline{|C|}$ represents the average size of distinct values per column.

Constructing Training Datasets. We pre-build two training datasets, one for the estimation module and one for the classifier module. In the estimation module, rows are sampled to form a training dataset for each incomplete table T, with a time complexity of $O(n)$ per table, where n denotes the number of rows in T. In the classifier module, the training dataset $\mathcal{D}_{train}$ is constructed, with a time complexity of $O(|\mathcal{D}_{train}| \cdot (|\mathcal{T}| + |\mathcal{B}|))$, where $\mathcal{T}$ represents the tables in the data lake.

A.2 Online Imputation

Given CESID first selects the appropriate imputation strategy using the classifier module, and then utilizes either the search or estimation module to obtain the imputed value, the overall time cost is the sum of the classifier module's time cost and the maximum time cost between the search and estimation modules. Next, we analyze the time cost of each module.

Classifier Module. In the classifier module, we first convert each missing value to a feature vector by referencing the tables and columns from the data lake, with a time complexity of $O(|\mathcal{T}|+|\mathcal{B}|)$. Next, we predict the label by retrieving the most similar sample from the training dataset $\mathcal{D}_{train}$, with the time complexity of $O(|\mathcal{D}_{train}|)$.

Search Module. In the search module, we first construct a *column-relevance graph*, with a time complexity of $O(m^2)$, where m denotes the columns of table T. Next, we identify the overlapping columns for the missing column using the LSH index, with a time complexity of $O(|\mathcal{B}|)$. After that, we discover the candidate tables $\mathcal{T}_s$ using the HNSW index, which has a time complexity of $O(|\mathcal{SC}| \cdot \log|\mathcal{B}|)$, where $\mathcal{SC}$ refers to the relevant columns. Finally, we scan all rows from the candidate tables to compute their similarities, with a time complexity of $O(\overline{|R|} \cdot |\mathcal{T}_s|)$, where $\overline{|R|}$ is the average number of rows per table.

Estimation Module. In the estimation module, we first identify the relevant tables using the HNSW index, with a time complexity of $O(|C| \cdot \log|\mathcal{B}|)$. Then, we apply the sampling-based exploration strategy to select rows from these tables to improve imputation accuracy, with a time complexity of $\frac{|\mathcal{R}|}{z}$ (cf. Section 4.2.2). Finally, we apply estimation-based methods to produce an imputed value[7].

[7]The time complexity for this step depends on the specific method used, thus we don't discuss it here.

Algorithm 2: Table Selection

Input: A sub-table $T_{\mathcal{SC}}$; Table sets $\mathcal{T}_{op}$, $\mathcal{T}_{su}$, $\mathcal{T}_{cu}$; A threshold k
Output: Table candidates

$\mathcal{T}'_{su} \leftarrow \mathcal{T}_{op} \cap \mathcal{T}_{su}$;
if $len(\mathcal{T}'_{su}) \geq k$ **then**
  return $\mathcal{T}'_{su}$;
else
  $\mathcal{T}'_{cu} \leftarrow \mathcal{T}_{op} \cap \mathcal{T}_{cu}$;
  `/* Record the unionability score */`
  $score \leftarrow \{\}$;
  foreach $T' \in \mathcal{T}'_{cu}$ **do**
    $score[T'] \leftarrow$ CalculateTableUnionability($T_{\mathcal{SC}}, T'$);
  `/* Select top tables by unionability score */`
  $\mathcal{T}'_{cu} \leftarrow$ SortTables($score, k - len(\mathcal{T}'_{su})$);
  return $\mathcal{T}'_{su} \cup \mathcal{T}'_{cu}$;