Data Imputation with Limited Data Redundancy Using Data Lakes

Chenyu Yang
HKUST(GZ)
Guangzhou, China
cyang662@connect.hkust-gz.edu.cn

Yuyu Luo*
HKUST(GZ)/HKUST
Guangzhou, China
yuyuluo@hkust-gz.edu.cn

Chuanxuan Cui
Renmin University of China
Beijing, China
2022104187@ruc.edu.cn

Ju Fan
Renmin University of China
Beijing, China
fanj@ruc.edu.cn

Chengliang Chai
Beijing Institute of Technology
Beijing, China
ccl@bit.edu.cn

Nan Tang
HKUST(GZ)/HKUST
Guangzhou, China
nantang@hkust-gz.edu.cn

ABSTRACT

Data imputation is essential for many data science applications. Existing methods rely heavily on sufficient data redundancy from within-table values. However, many real-world datasets often lack such data redundancy, necessitating external data sources. In this paper, we introduce a retrieval-augmented imputation framework, **LakeFill**, which combines large language models (LLMs) and data lakes to address this challenge. Unlike existing "table-level" retrieval methods designed for question answering, which retrieve data in the granularity of tables, **LakeFill** performs fine-grained "tuple-level" retrieval, optimized specifically for data imputation at the tuple level. It encodes (possibly incomplete) tuples to capture nuanced similarities and differences, enabling effective identification of candidate tuples. A novel reranking method that integrates checklist-based training data annotation with stratified training group construction further refines the retrieved tuples. Finally, a reasoner with a novel two-stage confidence-aware imputation ensures reliable imputation results. Extensive experiments show that **LakeFill** significantly outperforms state-of-the-art methods for data imputation when there is limited data redundancy.

PVLDB Reference Format:
Chenyu Yang, Yuyu Luo, Chuanxuan Cui, Ju Fan, Chengliang Chai, and Nan Tang. Data Imputation with Limited Data Redundancy Using Data Lakes. PVLDB, 18(10): 3354 - 3367, 2025.
doi:10.14778/3748191.3748200

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HKUSTDial/Retrieval_Augmented_Imputation.

1 INTRODUCTION

Data quality is a critical aspect of effective data analysis. Missing values are a common challenge [9, 17, 18, 44] for data quality management. Having a lot of missing values can drastically affect the quality of downstream applications, such as SQL queries [36, 37, 42], web form analytics [75, 76], data visualization [45, 47, 77], and other analytical processes [80, 87], compromising their accuracy, reliability, and overall effectiveness. We broadly categorize existing missing value imputation solutions into two types: (1) with enough data redundancy from within-table values [10, 31, 52, 83], or (2) limited data redundancy from within-table values, for which external knowledge must be leveraged [20, 21, 25, 38].

(1) With enough data redundancy. The presence of similar or repeating data values or patterns enables these methods to extract patterns, dependencies, and relationships, facilitating the imputation of missing values, using (a) integrity constraints [10, 15, 16], (b) statistical methods [12, 22, 31], (c) machine learning [52, 63, 65, 67], or (d) deep learning models [14, 54, 79].

(2) Limited data redundancy. In scenarios with limited data redundancy, there have been traditional rule-based methods using reference tables or knowledge graphs, as well as recent advances in employing large language models (LLMs).

(a) *User-provided external sources and rules:* Certain fix [21] uses user-defined rules on the reference table to repair the data. KATARA [19] proposes table patterns, which map a table to a knowledge graph based on the table semantics through crowdsourcing, and uses these patterns as rules for cleaning. However, both studies have high human cost, from rule definition to data repair.

(b) *Directly prompting LLMs.* LLMs have been used to impute missing values, either through fine-tuning (*e.g.,* TURL [20], TableGPT [38]) or in-context learning (*e.g.,* GPT [57]). However, for data imputation that requires precise answers, LLMs may suffer from hallucination and lack interpretability.

(c) *Retrieval-Augmented Generation (RAG) with LLMs.* RAG not only improves the accuracy of LLMs but also enables users to trace the origins of the imputed values, enhancing transparency. However, current RAG methods for data imputation, such as RATA [25], often operate at a coarse-grained "table-level", rather than more fine-grained "tuple-level" retrieval, which can limit their effectiveness for missing value imputation.

Missing value imputation using data lakes: what are needed? In practice, imputing missing values for one tuple often requires information from one or a few relevant tuples.

Example 1. [*Incomplete tuple.*] *As shown in Figure 1, BoB Riley has missing values in District, Party, and Birth Date attributes.*

[Impute with data lake.] We need to find T_1 to impute his missing District and Party values, and T_2 to impute his Birth Date value.

*Yuyu Luo is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748200

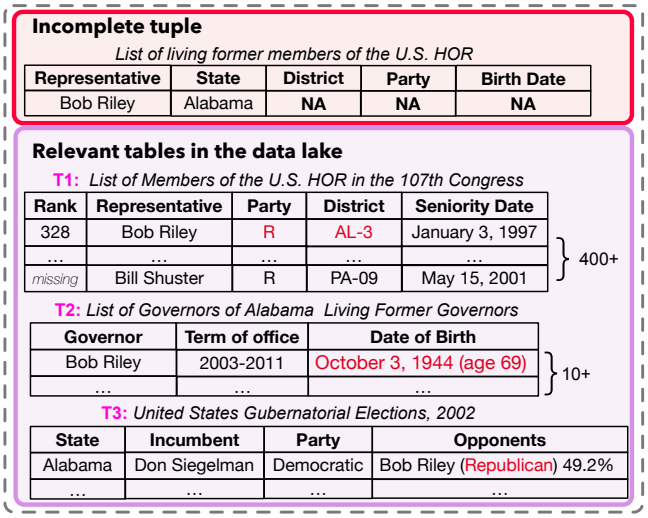

Figure 1: Imputing a tuple's missing values may require tuples scattered across multiple tables.

Note that, T_1 contains 400+ *tuples, only the "Bob Riley" tuple provides the necessary information for the given incomplete tuple.* □

Given a target table with many incomplete tuples, each requires tuple(s) from possibly different tables. Therefore, it is more natural to index data lakes at the tuple level. This could provide more precise and context-specific information, thereby enhancing the accuracy of the imputation compared to table-level retrieval.

Challenges. There are three main challenges with tuple-level RAG over a data lake for data imputation.

(C1) ***Retrieval: Encoding complete/incomplete tuples with heterogeneous schemas.*** Different from indexing tuples in one huge table, indexing tuples in data lakes face challenges like varying schemas, diverse textual representations, and missing values.

(C2) ***Reranking: computing tuple relevance in different granularity.*** Relying solely on tuple embedding similarity for retrieval may fall short, as embeddings often compress data that overlooks the fine-grained contextual relevance needed for imputation.

(C3) ***Ensuring the accuracy of LLMs with RAG.*** Even after reranking, the reranked tuples may also contain noisy tuples, *i.e.,* those tuples that are irrelevant.

Contributions. Our main contributions are summarized below.

(1) **The `LakeFill` framework.** We introduce **`LakeFill`**, a Retrieve-Rerank-Reason framework optimized for imputing missing values using data lakes. The **`Retriever`** designs tuple-level encoding for data lake tuples, effectively handling heterogeneous data and capturing imputation intent (Challenge 1). The **`Reranker`** is optimized with token-level methods (Challenge 2). To improve the accuracy, **`Reasoner`** devises a novel confidence-aware approach for further performance improvement (Challenge 3). (Section 2)

(2) **`Retriever`: Tuple imputation-aware representation learning.** We employ contrastive learning and carefully designed synthesized training data to learn tuple embeddings that capture both the imputation intent and the nuanced similarities and dissimilarities among heterogeneous data. (Section 3)

(3) **`Reranker`: Checklist-based training data annotation for token-level reranker.** We propose a novel approach that generates high-quality reranking data by integrating checklist-based annotation with stratified training group construction, enabling more accurate identification of retrieved tuples. (Section 4)

(4) **`Reasoner`: Confidence-aware inference for imputation.** We devise a novel two-stage confidence-aware imputation approach for accurate and confidence-aware data imputation. (Section 5)

(5) **Experiments.** We conducted extensive experiments on six datasets, showing that **`LakeFill`** significantly outperforms existing methods with limited data redundancy using data lakes. (Section 6)

2 AN OVERVIEW OF LAKEFILL

Data imputation with data lakes. A data lake $\mathbf{L} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_k\}$ is a collection of k tables, each of which may have a distinct schema. A relational table $\mathbf{D}$ consists of a schema, which is a set of attributes $R(\mathbf{D}) = \{A_1, A_2, \ldots, A_n\}$, defining the columns of the table, a set of tuples $\{t_1, t_2, \ldots, t_m\}$, and a textual caption. An incomplete tuple t is a tuple that contains one or more missing values. Given an incomplete tuple t and a data lake $\mathbf{L}$, the problem of *data imputation using data lakes* involves repairing t by retrieving relevant tuples from $\mathbf{L}$. The goal is to ensure that the repaired tuple t' matches its ground truth counterpart $t_\mathbf{g}$.

`LakeFill` overview. We propose **`LakeFill`**, which employs a classical Retrieve-Rerank-Reason framework for data imputation, as shown in Figure 2(a). Given an incomplete tuple t with missing values and a data lake L as input, **`LakeFill`** outputs a completed tuple t' by identifying and reasoning over retrieved tuples from L.

`Retriever`. This step retrieves relevant tuples for t from L. Using a pre-trained tuple encoder $\mathbf{enc}(\cdot)$, all tuples in L are encoded into vectors and stored in a vector database $\mathbf{V}$. Given t, **`LakeFill`** encodes it as $\mathbf{enc}(t)$ and performs a similarity search to retrieve the top-K most similar tuples from $\mathbf{enc}(t)$.

`Reranker`. This step refines the top-K retrieved tuples by recalculating their relevance to t using a finer-grained comparison, ensuring that the selected top-k tuples ($k \ll K$) are the most relevant.

`Reasoner`. This step leverages LLMs (*e.g.,* GPTs) to effectively impute missing values in t by considering the top-k retrieved tuples.

Existing approaches have the following limitations that prevent their direct application to our problem:

(1) Lack of an effective retriever for tuple-level imputation. Previous tuple embedding methods [39, 70, 73], fail to capture the complex connections between an incomplete tuple and relevant tuples in a data lake. Thus, we learn **tuple-level imputation-aware representation** (**`Retriever`**), pretrained using synthesized training data, as shown in Figure 2(b).

(2) Lack of fine-grained identification of relevant tuples. Existing methods often use simple rules without considering domain knowledge or contextual nuances [25, 46]. This approach overlooks the fine-grained understanding necessary for accurate matching. To enable the reranker to accurately identify relevant tuples, precise training data is required. However, manual annotation

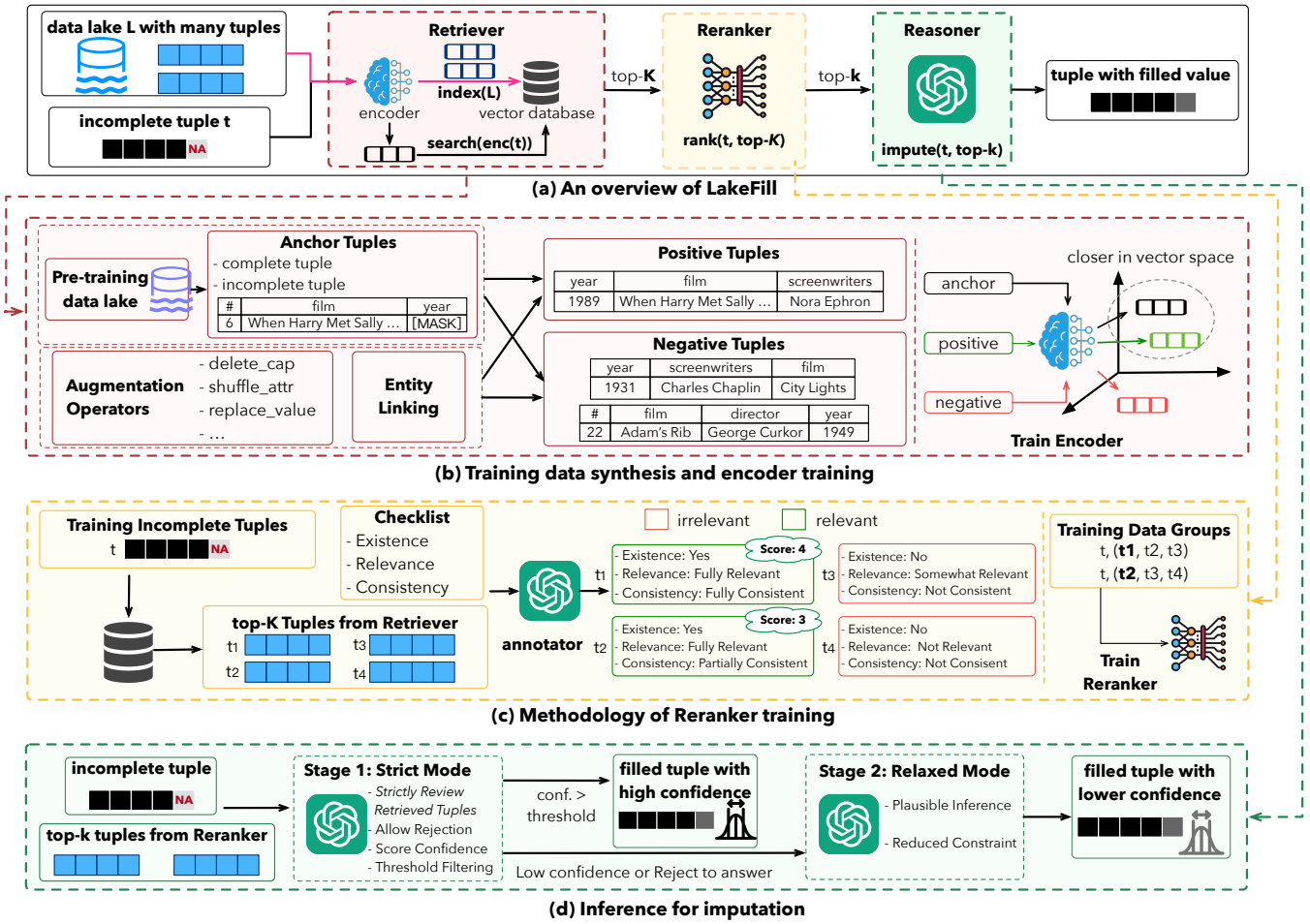

Figure 2: An overview of LakeFill.

is time-consuming. We propose **using LLM annotation with an evaluation checklist to construct training data for token-level Reranker**, as shown in Figure 2(c).

(3) Lack of reliability in inference for imputation. **Reasoner** must accurately derive missing values from top-k retrieved tuples, but retrieved tuples may lack the correct answer. To address this, we propose a **two-stage confidence-aware imputation** for **Reasoner** to ensure the reliability of imputed values while enhancing overall imputation accuracy, as shown in Figure 2(d).

3 RETRIEVER: LEARNING IMPUTATION-AWARE TUPLE REPRESENTATION

3.1 Contrastive Learning for Tuple Encoding

We employ contrastive learning within a Siamese network, leveraging its dual-encoder structure and shared weights for training. Each training sample is either an (Anchor, Positive) pair for closer tuple embeddings or an (Anchor, Negative) pair for farther embeddings.

Contrastive Loss. We optimize the **contrastive loss function** to maximize the similarity between (Anchor, Positive) pairs while minimizing the similarity between (Anchor, Negative) pairs.

We construct a batch $\mathcal{B}$ of training examples for N anchors, x_i ($i = [1, N]$). For each anchor x_i, we construct one positive pair (x_i, y_i^+) and M negative pairs $(x_i, y_{i,j}^-)$ ($j = [1, M]$). In total, there are $N \times (M+1)$ positive and negative samples in a batch $\mathcal{B}$, we denote the collection of them as Y, *i.e.*, $Y = \sum_{i=1}^{N}(y_i^+ + \sum_{j=1}^{M} y_{i,j}^-)$.

In the training process, we use the in-batch negative strategy [33], *i.e.*, for an anchor x_i, we consider all the other y in a batch except y_i^+ as its negatives, including both the original negatives of x_i, the negative and positive samples of other anchors in the batch. This strategy effectively trains the encoder by providing a larger pool of negatives without additional computational cost, resulting in $N \times (M+1) - 1$ negatives for each anchor x_i. We optimize the contrastive loss $\mathcal{L}$ as the negative log-likelihood of positive pairs:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(sim(x_i, y_i^+))}{\underbrace{\exp(sim(x_i, y_i^+))}_{\text{positive pair similarity}} + \underbrace{\sum_{y_k \in Y, y_k \neq y_i^+} \exp(sim(x_i, y_k))}_{\text{negative pairs similarity}}}$$

Table 1: Tuple augmentation operators.

Consider a sample tuple with a caption and attribute/value pairs: **Caption:** "Harrisburg, Pennsylvania, Sports" (**club:** Harrisburg . . ., **league:** USL Soccer, **venue:** Skyline . . .)		
Type	**Operator**	**Example**
Caption	delete_cap	Harrisburg, NA, Sports
	replace_cap	Harrisburg, Pennsylvania, Athletic
	shuffle_cap	Pennsylvania, Harrisburg, Sports
Attribute	shuffle_attr	new attribute order: **(league, venue, club)**
	delete_attr	new attribute set: **(club, venue)**
Value	replace_val	(Harrisburg . . ., United Soccer League, Skyline . . .)
	empty_val	(Harrisburg . . ., NA, Skyline . . .)

where $sim(x, y) = \textbf{enc}(x) \cdot \textbf{enc}(y)$, which calculates the similarity between the embeddings of the anchor tuple x and either the positive or negative y. We use dot product as the similarity function. Minimizing this loss increases the similarity between anchors and their corresponding positives, while simultaneously reducing their similarity to negatives.

3.2 Training Data Synthesis

Effectively training a tuple encoder with contrastive learning requires extensive training data. We propose to employ tuple augmentation operators and a three-step process for synthesizing training data. Our approach employs nuanced rules to ensure data diversity and heterogeneity while capturing a wide range of relevant cases to simulate real-world scenarios.

Tuple Augmentation Operators. Tuple augmentation operators transform a tuple into its "equivalent" form. These systematically designed operators will be used to construct training data that *simulates the diverse and heterogeneous nature* of data lakes.

Our data augmentation operators in Table 1 are designed to augment a tuple from three aspects: caption, attribute, or value.

Synthesizing Anchor Tuples. We construct two types of anchors:

(1) *Without missing values.* We sample a complete tuple from the data lake that has no missing entries, enabling the encoder to learn complete semantic representations.

(2) *With missing values.* To help the model learn the intent of imputation, we sample tuples and deliberately mask 30% of the cells containing significant information (*e.g.,* country names like "USA") with [MASK] symbol, while avoiding masking meaningless cells (*e.g.,* "-" or "?"). Using [MASK] helps differentiate these tuples from those with naturally occurring missing data in the data lake.

We synthesize anchor tuples in a 70/30 ratio of complete to masked tuples, which our experiments show yields superior results.

Synthesizing Positive Tuples. To create positive samples for an anchor tuple x_i, we employ two distinct strategies.

(1) *With tuple augmentation.* We augment the anchor tuple x_i using our designed tuple augmentation operations to generate positive tuples while maximizing the diversity.

(2) *With entity linking.* We identify tuples from other tables withe the same subject entity as x_i through entity linking, then augmenting them. They often differ significantly in schema and values.

Synthesizing Negative Tuples. We generate negative samples for each anchor tuple, categorized into two types:

(1) *Easy negatives.* Easy negatives are randomly selected tuples from other tables, due to the in-batch negative strategies, we do not need to construct them deliberately.

(2) *Hard negatives.* Hard negatives are tuples within the same table as the anchor tuple but represent distinct entities. Although hard negative tuples are quite similar to the anchor tuple since they are in the same table, they crucially represent different entities. This distinction is essential as it challenges the encoder to learn more discriminative and meaningful representations, enabling it to distinguish between similar but different tuples [48]. All negative samples are augmented as well to increase diversity.

Training Data Summary. For our training and development sets, we randomly select a subset of tables from the corresponding set of the WikiTables-TURL dataset [20] to construct them. In summary, our training set includes **282,862** anchor tuples from **41,260** tables, and the development set contains **9,460** anchor tuples from **775** tables. Each anchor tuple pairs with **1** positive and **7** negatives. Notably, our encoder, trained on this dataset, has shown strong generalization capabilities.

4 RERANKER: NOISE-ROBUST RERANKING VIA CHECKLIST-BASED DATA ANNOTATION

The **Retriever** often retrieves relevant tuples in top-K set only when K is sufficiently large (*e.g.,*, $K = 100$), which complicates subsequent reasoning. Therefore, a **Reranker** is required to prioritize relevant tuples within a smaller candidate subset. However, building such a model faces the challenge of obtaining accurately annotated training data. Unlike the heuristics strategies used in retrieval, training data of reranking requires precise annotations using domain knowledge.

We propose a noise-robust reranking approach that automatically constructs training data by combining: **(1)** a multi-dimensional evaluation checklist for fine-grained annotation, and **(2)** stratified training groups with contrastive learning to mitigate labeling errors. Instead of binary judgments (relevant or irrelevant), LLMs assess candidates across existence, relevance, and logical consistency. Based on these nuanced assessments, we assign labels and scores, construct training groups, and train the reranker to distinguish subtle differences within groups, rather than relying solely on absolute labels. We now detail this checklist-based data annotation process and our constructed training groups.

Checklist-based Training Data Annotation. Given an incomplete tuple in the training set, we aim to label at least $n-1$ negative candidates and at least one positive candidate to form a training group of size n. We retrieve the top-K ($K \geq n$) candidates using the **Retriever** and sequentially evaluate each (incomplete tuple, candidate tuple) pair with GPT-4o as annotator. The process terminates early once required samples are labeled, reducing computational cost and time. If no positive candidate is found after evaluating K candidates, the incomplete tuple is discarded. Leveraging the high retrieval performance of our **Retriever**, when $K = 30$ and $n = 16$, our method ensures that over 90% of the incomplete tuples in the original training set obtain sufficient training data and are retained.

For each pair, the LLM evaluates it using a checklist with three dimensions, as shown in Figure 3. Unlike binary relevance labels,

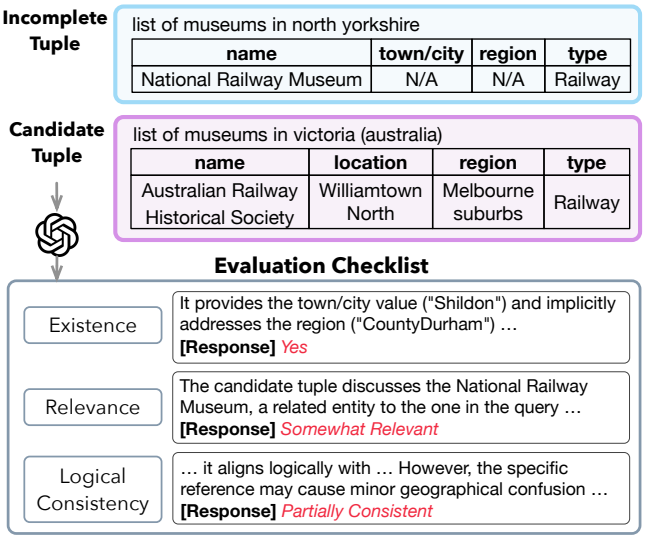

Figure 3: Evaluation checklist for reranking training data.

the checklist guides LLMs to provide fine-grained assessments, leading to a ***more accurate and nuanced judgment***:

Existence: Check whether the candidate tuple contains at least one of the missing attributes that need to be filled in the incomplete tuple. Options: **Yes** or **No**.

Relevance: Evaluate the degree of relevance between the candidate tuple and the incomplete tuple. Options: **Highly Relevant**, **Somewhat Relevant**, **Not Relevant**.

Logical Consistency: Evaluate whether the missing values in the query tuple can be logically inferred from the candidate tuple without contradictions. Options: **Fully Consistent**, **Partially Consistent**, **Not Consistent**.

After the LLM completes the annotation, we assign a label for each candidate tuple based on the following criteria:

(1) Existence: it is crucial for imputation since a candidate tuple that does not include any of the required missing attributes cannot contribute to filling the incomplete tuple. Thus, if the candidate is labeled *No*, it is directly classified as a **negative**. If labeled *Yes*, it proceeds to the next evaluation steps as a **potential positive**.

(2) Relevance and Logical Consistency: Each dimension is scored as *0* (lowest), *1* (moderate), or *2* (highest). For candidates marked as *Yes* in Existence, we calculate the combined score of these two dimensions: (a) If the combined score is 2 or higher, the candidate is labeled as a **positive**; (b) otherwise, it is labeled as a **negative**.

Theoretical Justification of Checklist-based Annotation. To support our design, we provide a theoretical justification that our checklist-based annotation can achieve lower error rates than binary annotation.

Theorem 4.1. *Consider a binary classification task where a candidate tuple is labeled relevant or irrelevant. In checklist-based annotation, each candidate is evaluated on m dimensions. Each dimension s_i provides a score, and The total score, $S = \sum_{i=1}^{m} s_i$, is compared against a threshold τ to determine the label.*

For any given annotation task, there exists a checklist with m evaluation dimensions such that: (1) Each dimension s_i is ***discriminative****: $E[s_i|y = 1] > E[s_i|y = 0], \quad \forall i \in \{1, \ldots, m\}$; (2) The dimensions are* ***conditionally independent*** *given the label y: $P(s_1, \ldots, s_m|y) = \prod_{i=1}^{m} P(s_i|y)$. Furthermore, there always exists a threshold τ that simultaneously reduces the false-positive rate (FPR) and false-negative rate (FNR) compared to binary relevance annotation:*

$$\epsilon'_{FP} < \epsilon_{FP}, \quad \epsilon'_{FN} < \epsilon_{FN},$$

where $\epsilon_{FP} = P(\hat{y} = 1|y = 0)$, $\epsilon_{FN} = P(\hat{y} = 0|y = 1)$ are the FPR and FNR for binary relevance annotation, and $\epsilon'_{FP} = P(S \geq \tau|y = 0)$, $\epsilon'_{FN} = P(S < \tau|y = 1)$ are the rates for checklist-based annotation.

Proof. Let $S_1 \sim P(S \mid y = 1)$ and $S_0 \sim P(S \mid y = 0)$ denote the total score distributions for relevant and irrelevant candidates under checklist-based annotation. To ensure checklist-based annotation reduces the false-positive and false-negative rates, it suffices to find a threshold τ that:

$$\epsilon'_{FP} = P(S \geq \tau \mid y = 0) < \epsilon_{FP}, \epsilon'_{FN} = P(S < \tau \mid y = 1) < \epsilon_{FN}. \quad (1)$$

This is equivalent to:

$$Q_0(1 - \epsilon_{FP}) < \tau < Q_1(\epsilon_{FN}) \quad (2)$$

where $Q_0(\cdot)$ and $Q_1(\cdot)$ are the quantile functions of S_0 and S_1, respectively. A sufficient condition for such a τ to exist is: $Q_0(1 - \epsilon_{FP}) < Q_1(\epsilon_{FN})$. Under the assumptions of the theorem, *i.e.,* the discriminative property and conditional independence of dimensions, the score distributions satisfy:

$$\mu_1 = \sum_{i=1}^{m} E[s_i \mid y = 1], \quad \mu_0 = \sum_{i=1}^{m} E[s_i \mid y = 0], \quad (3)$$

$$\sigma_1^2 = \sum_{i=1}^{m} \mathrm{Var}(s_i \mid y = 1), \quad \sigma_0^2 = \sum_{i=1}^{m} \mathrm{Var}(s_i \mid y = 0). \quad (4)$$

By the Central Limit Theorem, when m is sufficient to large, the total scores approximately follow normal distributions: $S_1 \sim N(\mu_1, \sigma_1^2)$, $S_0 \sim N(\mu_0, \sigma_0^2)$ Thus, the quantiles in Equation (2) are approximated as:

$$Q_0(1 - \epsilon_{FP}) \approx \mu_0 + z_{1-\epsilon_{FP}}\sigma_0, \quad Q_1(\epsilon_{FN}) \approx \mu_1 - z_{1-\epsilon_{FN}}\sigma_1 \quad (5)$$

where z_p denotes the p-th quantile of the standard normal distribution. Thus, inequality (2) is satisfied if: $\mu_1 - \mu_0 > z_{1-\epsilon_{FP}}\sigma_0 + z_{1-\epsilon_{FN}}\sigma_1$. Note that this equation is guaranteed to hold under either of the following sufficient conditions: (i) when the number of checklist dimensions m is sufficiently large, the score gap $\mu_1 - \mu_0 = \Theta(m)$ grows faster than the standard deviation terms $\sigma_y = O(\sqrt{m})$; or (ii) when each dimension is strongly discriminative and has low variance, even for small m.

Therefore, there exists a checklist with m dimensions that satisfies the conditions in the theorem, and a corresponding threshold τ such that $\epsilon'_{FP} < \epsilon_{FP}, \quad \epsilon'_{FN} < \epsilon_{FN}$, which completes the proof. □

Construct Stratified Training Groups for Contrastive Training. After annotation, each incomplete tuple is paired with x positives and y negatives ($x \geq 1$, $y \geq n - 1$). While many methods [23, 60] directly model reranking as a binary classification task (positive or negative), we propose stratified training groups with contrastive learning to better handle annotation noise. Even with

the checklist, LLM judgments are not always precise. For example, as shown in Figure 3, an irrelevant candidate may be labeled as "positive" with a score of 2. Directly training on these noisy labels may propagate annotation errors. Our **key insight** is that while individual labels may be noisy, the relative ranking among candidates by score are more reliable: higher-scored candidates tend to be more relevant. We therefore organize candidates into stratified groups that preserve score-based orderings, allowing the model to learn relative differences more effectively.

Specifically, for an incomplete tuple t, we organize its annotated candidates into groups. For a positive candidate P_i with score $s(P_i)$, we construct the corresponding training group as: $G_i = \{(t, P_i), \{(t, P_j) \mid s(P_j) < s(P_i)\}, \{(t, N_k)\}$, where P_j are positives with scores lower than $s(P_i)$, and N_k are sampled negatives. The total size of each group is n. To construct each group, we begin by including P_i and its lower-scored positives, then sample enough negative candidates to reach the required group size n. As a result, each incomplete tuple will have x training groups.

We use a BERT-based reranker model. The incomplete tuple and candidate tuple are serialized and concatenated before being fed into the BERT model. The output of the first token [CLS] is then passed through a linear layer to obtain a score. For each training group, we apply contrastive loss, where the goal is to maximize the positive candidate score P_i while minimizing all the other negative candidates in the group. This stratified grouping approach allows the reranker to learn subtle differences between tuples with varying scores, preferring tuples that are more logically consistent and relevant to the incomplete tuple.

5 REASONER: CONFIDENCE-AWARE INFERENCE

After obtaining the top-k reranked results, the process advances to the pivotal stage of data imputation using LLMs as **Reasoner**. Unlike LLMs used without external knowledge [49, 71, 74, 84, 85], the RAG paradigm adopted by **LakeFill** grounds the generation on retrieved content and has been shown to reduce hallucinations [24, 72]. However, even with RAG, hallucinations can still occur [59, 86], especially when the top-k retrieved results include noisy or irrelevant tuples despite reranking, or when the model overrelies on its parametric knowledge, overriding the retrieved context and introducing contradictions [69]. To mitigate the negative impact of irrelevant tuples, we propose a **Two-Stage Confidence-Aware Imputation framework**, as shown in Figure 4. It is based on the hypothesis that encouraging LLMs to strictly review retrieved evidence and minimize the reliance on internal parametric knowledge can substantially improve precision, which we empirically validate in Section 6.3.

Our core idea is to **prioritize precision over overall imputation accuracy**. In the first stage, the LLM is required to base its decision solely on the retrieved tuples, strictly reviewing the evidence to produce a confident imputed value or to reject imputation if the evidence is insufficient. However, relying only on this strict, evidence-driven mode inevitably reduces overall accuracy, as many imputations may not meet the confidence threshold. To address this, we introduce a second stage: a relaxed mode that allows the LLM to make plausible but less rigorously grounded inferences, thereby improving overall imputation accuracy at the cost of reduced precision. This two-stage design allows users to make informed trade-offs: leverage the strict mode for high-precision, evidence-backed values, and then fall to the relaxed mode for higher imputation when needed. Specifically, given top-k retrieved tuples R and incomplete tuple t, the imputation function $\mathcal{I}$ is defined as:

$$\mathcal{I}(R, t) = \begin{cases} (v, c_s) & \text{if } c_s \geq \tau \quad \text{(Stage 1)} \\ (v_r, c_r) & \text{otherwise} \quad \text{(Stage 2)} \end{cases}$$

where τ is the confidence threshold, v and v_r denote imputed values from the first and second stage respectively, with confidence scores $c_s = \text{LLM}_{\text{strict}}(R, t)$ and $c_r = \min(\text{LLM}_{\text{relaxed}}(R, t), \tau)$. Notably, we adopt verbalized confidence scores [78], where the LLM is explicitly prompted to estimate its own confidence in the form of output tokens, *i.e.,* a score between 0 and 1. This simple yet effective approach allows us to measure the model's self-assessed certainty. We further discuss the rationale and empirically demonstrate the reliability of these scores later.

Stage 1: Evidence-Driven Strict Mode. First, LLMs review the retrieved tuples from the three dimensions used in reranking (existence, relevance, logical consistency) and operates under the following constraints: (1) Rejection mechanism: Automatically refuses imputation when retrieval evidence is insufficient for confident reasoning. (2) Confidence score: Assigns confidence scores [0,1] based on evidence completeness. (3) Threshold filtering: Allows users to set strict thresholds to accept only high-confidence imputations.

Stage 2: Relaxed Completion Mode. For cases rejected in Stage 1 (insufficient evidence or low confidence), it activates: (1) Plausible inference: Allows LLM to fill missing values through plausible inference. (2) Reduced constraints: Accepts weaker logical consistency requirements. (3) Differentiated marking: Labels results with "[Relaxed]" tags for traceability.

As the confidence threshold τ increases, the precision of the first stage improves due to stricter filtering, but overall accuracy tends to decline as more values are filled in the relaxed mode. However, thresholds that are too low may sometimes introduce unreliable results that are less accurate than the model's internal knowledge. Our empirical observations show that LLMs typically assign scores ≥ 0.9 to reliable imputations, so we set $\tau = 0.9$ by default.

Reliability of Confidence Score. A key requirement for our imputation framework is that the LLM's verbalized confidence score reflects the likelihood of a correct imputation. In our setting, this score is not purely derived from the model's internal knowledge. Instead, it also reflects the quality of retrieved evidence, making it more grounded and less prone to overconfidence.

To assess the reliability of these scores, we evaluate their **calibration**—that is, whether a confidence score c corresponds to a c-level probability of being correct. We introduce the **Expected Calibration Error (ECE)** [78] to measure this alignment, defined as the expected gap between predicted confidence and actual accuracy: $\text{ECE} = \mathbb{E}_c\left[|\Pr[\text{correct} \mid C = c] - c|\right]$. In practice, we divide samples into M bins based on their confidence scores and compute: $\text{ECE} \approx \sum_{m=1}^{M} \frac{|B_m|}{n} \cdot |\text{acc}(B_m) - \text{conf}(B_m)|$, where $\text{acc}(B_m)$ and $\text{conf}(B_m)$ are the average imputation accuracy and confidence score in bin B_m.

Table 2: Statistics of mvBench (Tab. : Tables; Tup. : Tuples; Attrs: Attributes).

Datasets	Incomplete Tuples		Data Lake		Avg. Relevant Tup.	Part of Missing Attrs	#-Training Tup.
	#-Tab.	#-Tup.	#-Tab.	#-Tup.	#-Relevant Tup./#-Tup.		
WikiTuples (WT)	665	6,887	207,912	2,674,164	3.98	Party, Director, Team, ...	100
Show Movie (SM)	1	30	3	19,586	1	Age Rating	6
Cricket Players (CP)	1	213	2	94,164	1.38	Nationality, Batting Style	20
Education (ED)	2	654	17	11,132	4	Address, Zipcode, Phone	30
Zomato (ZM)	1	529	16	468,252	3.48	Location	30
FIFA World Cup (FF)	1	696	14	118,251	1.44	Home / Away Team Goals	30

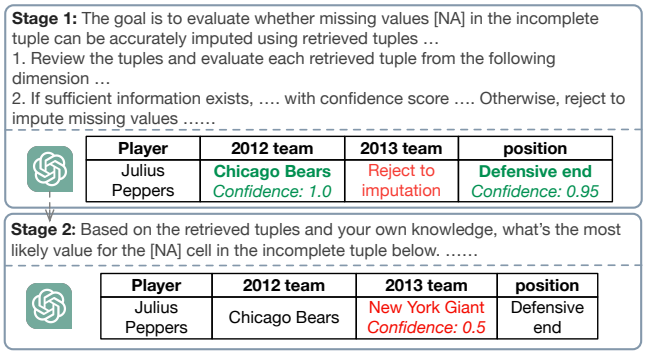

Figure 4: Two-Stage Confidence-Aware Imputation.

To demonstrate the reliability of our verbalized confidence score, we evaluate its ECE across multiple datasets. Additionally, we compare it against two widely adopted probability-based baselines, following [13]: Length-Normalized Score and Entropy-Based Score, both of which are mapped into the $[0, 1]$ using exponential transformation to serve as confidence scores. The details are shown in Section 6.3.

6 EXPERIMENTS

6.1 Datasets

Although existing datasets [25, 52, 55] for data imputation provide missing data and their corresponding ground truth, they mostly lack: (1) large data lakes containing massive tables that can assist in filling missing values, or (2) labeled relevant tuples or tables that facilitate the imputation of missing values. To overcome these limitations, we introduce **mvBench**, a large-scale benchmark containing **9,009** incomplete tuples and **3.39** million tuples from the data lake for missing value imputation with data lakes. To enable a fine-grained evaluation of the retrieval module, we further provide relevant tuples annotated by human experts for incomplete tuples.

Dataset Statistics. **mvBench** comprises six datasets collected from real-world scenarios, varying in scales, domains, and sources. Table 2 summarizes the statistics. The WikiTuples dataset is constructed based on WikiTables-TURL [20], with incomplete tuples from the test set and the data lake from the train set. Show Movie and Cricket Player datasets are adapted from RetClean [11]. While the incomplete tables and data lakes are from RetClean, since relevant tuples were not provided, we manually annotated them. The Education dataset, constructed from the Chicago Data Portal [5], focuses on Chicago schools. For Zomato and FIFA, their incomplete tuples are sourced from previous works [41, 55, 66]. We construct their data lakes by collecting domain-related tables from Kaggle, and manually annotate the tuples that are relevant for imputation. For each dataset, a subset of incomplete tuples is randomly sampled to form the training set for the reranker (see "#Training Tup." column of Table 2), while the remaining serves as the test set.

Relevant Tuple Annotation. The annotation of relevant tuples involves two steps: Candidate Tuples Construction to quickly filter out potential relevant tuples from data lakes, and Expert Annotation to ensure that the relevant tuples contain information to fill missing values, ensuring label accuracy.

Candidate Tuples Construction. For an incomplete tuple, we first create a candidate set by selecting tuples that could potentially help in imputing its missing values. We use explicit information, such as similar cell values or cells linking to the same *entity*, to establish effective filtering rules.

Expert Annotation. We present each incomplete tuple with its candidates to a human expert to judge whether the candidate can fill at least one missing value in the incomplete tuple, *i.e.,* be identified as a relevant tuple. This manual process is time-consuming and requires specific domain knowledge, thus we hire 10 PhD students as our "human experts" to annotate labels. To reduce the cost, we primarily focus on cases where the candidate set comprises 10 or fewer tuples. In total, over 200 **human hours** and approximately $1,000 were spent on curating relevant tuple labels for each incomplete tuple.

Remark. We do not use human-annotated relevant tuples for training **LakeFill**, they are mainly used to evaluate annotation quality and retrieval performance Our **Retriever** is trained on synthetic data rather than **mvBench**. For **Reranker**, we use checklist-based annotation to synthesize training data for incomplete tuples in the training set, eliminating the need for manual labeling.

6.2 Experimental Setting

6.2.1 Baseline Methods. We evaluate **LakeFill** from three aspects: performance on data imputation, retrieval, and reranking. For each aspect, we compare **LakeFill** against different SOTA methods.

Baselines for Reasoner (Data Imputation). **LakeFill** focuses on data imputation in scenarios with limited data redundancy, thus, we compare **LakeFill** with methods leveraging external knowledge. Since rule-based methods [19, 21] rely on human-defined rules, they are unsuitable for using data lakes for imputation.

(1) LLM without Retrieval [56]: We directly employ GPT-4o mini [2] and GPT-4o [1] for data imputation without using a retriever.

(2) LLM with BM25 retriever: To investigate the impact of a weaker retrieval method in our framework, we adopt BM25 [64] to retrieve tuples for LLMs.

(3) LM with fine-tuning: TURL [20] is one of the state-of-the-art pre-trained tabular models on table-related tasks. It can perform data imputation when the missing cell content can be linked to an entity seen during pre-training.

(4) Table-based retrieval: RATA [25] focuses on table-level retrieval to impute missing values, aiming to retrieve entire tables from the data lake rather than individual tuples.

For baselines (1) and (2), we compare baselines (1) and (2) with **LakeFill** in Exp-1. Due to differences in settings, baselines (3) and (4) cannot be directly applied to all datasets; hence, their comparisons are discussed separately in Exp-2.

Baselines for Retriever. We compare our **Retriever** against several baselines. The first two methods have zero-shot capabilities and are thus applied directly. For the third method, we retrain it using the same synthesized data in Section 3.2 for a fair evaluation.

(1) BM25 [64] is the most commonly-used sparse retrieval method.

(2) Contriever [29] is an unsupervised retriever that excels in few-shot and zero-shot passage retrieval.

(3) DPR-scale [61] is a dense retriever pre-trained on 65 million questions for passage retrieval.

(4) BERT with masked language modeling (MLM) task (*i.e.,* model trained on individual tuples). To assess the effectiveness of contrastive learning for tuple representation in retrieval, we train a BERT-based tuple encoder using individual tuples, following the language modeling task [58, 70]. Specifically, 30% of cells are randomly masked, and the model predicts the missing values.

(5) Sudowoodo [73] is a state-of-the-art entity matching method. For each dataset, we pretrain the model using the training set and 10,000 randomly selected tuples from the data lake and incomplete tuples. Pretraining is conducted for 3 epochs, followed by 40 epochs of fine-tuning.

Baselines for Reranker. Existing reranking methods broadly fall into two types: fine-tuned and prompt-based methods. We adopt this categorization for our baseline design.

Fine-tuned models require supervised training. In our setting, we use the positive/negative labels from the annotation phase for their training. We include three representative approaches:

(1) monoBERT [60] is a BERT-based model which concatenates two tuples and use the [CLS] token representation to compute reranking scores. We initialize it with the provided parameters [6].

(2) monoT5 [23] is a generative model that outputs "true" or "false" tokens to indicate relevance between an incomplete tuple and a retrieved one. It is initialized with pre-trained parameters from [8].

(3) RankLLaMA [50] is a fine-tuned LLaMA-based model for pointwise reranking. We fine-tuned both LLaMA2-7B [7] and LLaMA3-8B [3], resulting in two variants: RankLLaMA2 and RankLLaMA3.

Prompt-based methods leverage large language models (LLMs) without fine-tuning. We explore two common paradigms:

Table 3: Exact Match (EM) Accuracy of Data Imputation across Datasets. "Tup." denotes Tuples.

Reasoner	Retriever	WT	SM	ED	CP	ZM	FF
GPT-4o mini	w/o	0.63	0.75	0.119	0.904	0.0	0.475
	w/ tup. (BM25)	0.679	0.75	0.908	0.905	0.754	0.543
	w/ tup. (**LakeFill**)	**0.881**	**0.875**	**0.977**	**0.921**	**0.914**	**0.927**
GPT-4o	w/o	0.809	0.75	0.112	0.938	0.0	0.484
	w/ tup. (BM25)	0.815	0.833	0.928	0.932	0.758	0.551
	w/ tup. (**LakeFill**)	**0.907**	**0.917**	**0.978**	**0.97**	**0.924**	**0.948**

(1) Pairwise [62]: The LLM compares pairs of tuples to determine which one is more relevant and aggregates preferences via a bubble-sort-like process.

(2) Listwise [51, 68]: Given an incomplete tuple t and a list of candidates, we ask LLM to generate a reranked list based on relevance to t. Due to input limitations, we adopt a sliding window strategy with a window size of 30 and a step size of 14 [68].

For each prompting paradigm, we evaluate three LLMs: GPT-4o mini, GPT-4o, and LLaMA3-70B, yielding 6 prompt-based configurations in total.

6.2.2 Evaluation Metrics. We evaluate the end-to-end performance of **LakeFill** for data imputation using Exact Match (EM) Accuracy [30], which considers a generated value correct if it matches any acceptable answer after normalization. The performance of the **Retriever** is assessed using ***recall@K (R@K)***, the proportion of relevant tuples within the top-K results. For the **Reranker**, following [34], we report ***success@k (S@k)***, the percentage of incomplete tuples for which top-k retrieved tuples contains at least one relevant tuple.

6.2.3 Environment and hyper-parameters. Both **Retriever** and **Reranker** use BERT-base-uncased model (110M) [4] as the base framework. We set the batch size to 16 and train it for 2 epochs using AdamW optimizer [43]. Training takes approximately 7 hours on 4 RTX 4090 GPUs. Models are saved every 10,000 steps, and the one with the lowest development set loss is selected. **Reranker** is initialized with parameters pre-trained on the MS MARCO passage dataset [23]. For **Reasoner**, we utilize GPT-4o mini and GPT-4o with a temperature setting of 0.3, the default confidence threshold is 0.9. All experiments are conducted on an Ubuntu 22.04 server equipped with 8 RTX 4090 GPUs.

6.3 Evaluation for Data Imputation

Exp-1: How does LakeFill compare with LLMs without retrieval and with weaker retrieval methods? We evaluate the effectiveness of **LakeFill** in end-to-end data imputation by comparing it to two baselines: (1) LLMs without retrieval, which rely solely on internal parametric knowledge; and (2) LLMs augmented with a weaker retrieval method, specifically BM25. In all cases, both **LakeFill** and the baseline methods use GPT-4o mini or GPT-4o as the reasoner. For settings involving retrieval, the top-5 retrieved tuples, along with the incomplete tuple, are sent to the LLM.

As shown in Table 3, **LakeFill** significantly improves imputation accuracy compared to LLMs in isolation. When relying solely on their internal knowledge, LLMs often produce suboptimal results, even on relatively familiar domains such as Wikipedia. The

challenge is further amplified in specialized domains such as Education, which includes fine-grained information about public schools, and Zomato, where the Location attribute requires detailed addresses, not just province data. In such cases, GPT-4o mini and GPT-4o yield accuracies as low as 0.119 and 0.112, or even 0. In contrast, **LakeFill** effectively integrates the reasoning capabilities of LLMs with knowledge from retrieved tuples, achieving substantially higher performance. Excluding the outlier cases of Education and Zomato, **LakeFill** achieves average accuracy gains of 21.1% over GPT-4o-mini and 19% over GPT-4o.

The results underscore the importance of an effective retriever. As shown in Table 5, BM25 consistently underperforms compared to our **Retriever** across all datasets. While BM25 shows notable result in the Education dataset, where LLMs lack prior knowledge, its improvements elsewhere are limited. In the Cricket Player dataset, it even harms performance, likely due to introducing irrelevant or noisy tuples that increase the LLMs' reasoning burden. Additionally, GPT-4o generally outperforms GPT-4o mini due to its advanced reasoning capabilities, except on the Cricket Player dataset without retrieved tuples, where both models lack domain knowledge. Interestingly, when retrieval-augmented imputation is applied, the performance gap between GPT-4o and GPT-4o mini narrows. These findings show that effective retrieval can enable smaller models to bridge the knowledge gap and achieve performance closer to that of larger models, highlighting the strength of our framework in enhancing imputation accuracy.

Finding 1. **LakeFill** *significantly* ***outperforms*** *baseline LLMs and those with weaker retrieval methods, enhancing data imputation accuracy across various domains and compensating for the knowledge gaps of less advanced language models on data imputation.*

Exp-2: How does LakeFill compare with fine-tuned language models and table-based RAG methods? We further discuss the differences and advantages of **LakeFill** compared to the other two types of data imputation methods: a fine-tuned language model without retrieval and table-based RAG. Given their different settings, we discuss them separately:

LakeFill *v.s. TURL (fine-tuned LM).* We compare **LakeFill** with TURL [20] on the WikiTuples dataset, which is derived from TURL's test set. TURL first retrieves candidate entities based on co-occurrence in training tables and then reranks them using a fine-tuned LM, while **LakeFill** directly generates an imputed value. To ensure a fairer comparison, we report Precision@k (*i.e.,* whether the ground truth appears in the top-k candidates) for both models. Since **LakeFill** generates a single imputed value per incomplete tuple, it achieves P@1 = P@5 = 0.907, whereas TURL achieves P@1 = 0.788 and P@5 = 0.967.

TURL achieves a higher P@5 due to its candidate list obtained from training data, where each missing value in WikiTuples appears at least three times, and TURL is explicitly trained to capture entity relationships within this dataset. These favorable conditions give TURL a strong advantage, likely making its performance an upper bound. However, TURL performs worse in P@1, which better reflects scenarios requiring a single imputed value. This highlights **LakeFill** 's practical effectiveness, especially considering it uses only 7% of TURL's training data. Moreover, TURL is restricted

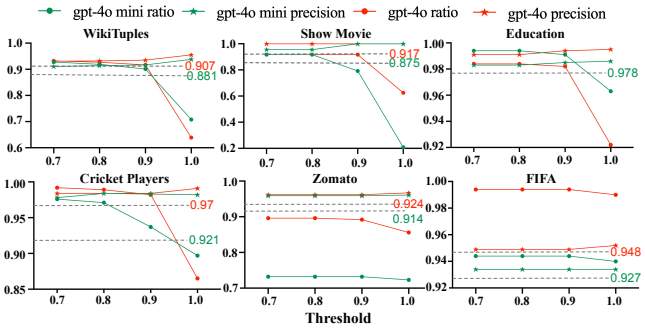

Figure 5: Impact of confidence thresholds on imputation precision and ratio.

to only imputing values that could be linked to an entity in its pre-constructed entity vocabulary, making it inapplicable to our other datasets, whereas **LakeFill** generalizes easily across datasets without such constraints.

LakeFill *(tuple retrieval) v.s. RATA (table retrieval).* RATA [25] is the state-of-the-art retrieval-augmented imputation model. While it focuses on table-level retrieval, our task requires more fine-grained retrieval at the tuple level. Additionally, its definition of the relevant table is different from ours. Thus, it's hard to adopt RATA on our datasets. To provide a comparison, following RATA's setting, we evaluate **LakeFill** on RATA's EntiTables dataset using MRR (Mean Reciprocal Rank) metric. For **LakeFill**, we define a special form of MRR@1: MRR is 1 if the imputed value is correct, and 0 if incorrect.

We apply **LakeFill**'s **Retriever** without additional training, achieving an MRR@10 of 0.552, a 17.7% improvement over RATA's 0.375. We then feed the top-10 retrieved tuples to GPT-4o mini, yielding an MRR@1 of 0.415, significantly exceeding RATA's reported results (MRR@10: 0.343). This is attributed to **LakeFill**'s tuple-based retrieval, which provides more relevant information for imputation tasks. Despite requiring more storage space than RATA (30G *v.s.* 14G on EntiTables), **LakeFill** justifies its larger footprint by providing much better results, demonstrating its efficiency and effectiveness in handling complex data imputation challenges.

The improvement in imputation accuracy is not as significant as the retrieval performance because RATA's definition of relevant tuples is simplistic: a table is considered relevant merely if containing the missing value, regardless of inferential logic. However, our **Reasoner** applies logical inference to filter out such superficially relevant tuples that do not contribute to reliable imputation.

Finding 2. **LakeFill** ***demonstrates significant advantages*** *over table-based retrieval methods and fine-tuned language models in adaptability and imputation performance. While its storage requirements are higher,* **LakeFill** ***strikes an acceptable balance*** *between storage and performance, making it highly effective when high imputation performance is prioritized.*

Exp-3: Can confidence thresholds enhance imputation reliability? This experiment evaluates whether confidence scores from LLMs can enhance imputation reliability by filtering out low-confidence predictions. Since reliable imputation demands not just

Table 4: Calibration of different methods. Metric: ECE. ⇓: Calibration error decreased compared to the 'relaxed' mode.

Dataset	Mode	Model	Verbalized	Length-Norm	Entropy
WT	strict	gpt-4o-mini	**0.0592** ⇓	0.0947	0.083
		gpt-4o	**0.0198** ⇓	0.1034	0.1783
	relaxed	gpt-4o-mini	**0.0922**	0.1632	0.1435
		gpt-4o	**0.0348**	0.1472	0.2312
CP	strict	gpt-4o-mini	**0.015** ⇓	0.0945	0.1442
		gpt-4o	**0.0112** ⇓	0.1345	0.2087
	relaxed	gpt-4o-mini	0.08	**0.014**	0.032
		gpt-4o	**0.0338**	0.1206	0.1777
ED	strict	gpt-4o-mini	**0.0017** ⇓	0.057	0.0638
		gpt-4o	**0.0036** ⇓	0.1593	0.14
	relaxed	gpt-4o-mini	0.0546	**0.0275**	0.0384
		gpt-4o	**0.0317**	0.1271	0.1676
ZM	strict	gpt-4o-mini	**0.0333**	0.0413	0.0828
		gpt-4o	0.053 ⇓	0.1264	0.1653
	relaxed	gpt-4o-mini	0.03	0.0858	0.0712
		gpt-4o	0.09	0.1096	0.1937
FF	strict	gpt-4o-mini	*0.035* ⇓	**0.0192**	**0.0045**
		gpt-4o	**0.0283** ⇓	0.0888	0.1601
	relaxed	gpt-4o-mini	**0.0433**	0.1074	0.0872
		gpt-4o	**0.125**	0.161	0.207

high accuracy but also trustworthy individual values, we focus on precision (correctness of filled values) and imputation ratio (proportion of missing values imputed).

We apply strict mode with varying confidence thresholds (0.7, 0.8, 0.9, 1.0), using the top-5 retrieved tuples as input to GPT-4o mini and GPT-4o. Values below the threshold are discarded, and we measure the resulting precision and imputation ratio (Figure 5). We also include the overall precision after performing two stages with 0.9 threshold as the baseline. From Figure 5, we observe that the precision in strict mode consistently exceeds the baseline, improving as the threshold increases, while the imputation ratio decreases. At thresholds of 0.7 to 0.9, the differences in precision and ratio are minimal, as many confidence scores from GPT-4o and GPT-4o mini are ≥ 0.9, otherwise they are likely to be rejected, supporting 0.9 as a reasonable default. At threshold 1.0, precision generally improves significantly, though this results in a decline in imputation ratio. To ensure higher precision, selecting only imputations with 1.0 confidence in strict mode yields an average precision improvement of 5.1% and 3.6% for GPT-4o mini and GPT-4o over baseline, respectively, at the cost of reduced imputation ratio.

*Finding 3. Our confidence thresholding successfully **identifies high-reliability imputations**, with GPT-4o mini and GPT-4o achieving precision improvements of 5.1% and 3.6% over the baseline.*

Exp-4: Are Verbalized Confidence Scores Well-Calibrated? Threshold filtering is only effective if confidence scores are well-calibrated, *i.e.,* accurately reflecting the likelihood of a correct imputation. Thus, to more precisely quantify the reliability of verbalized confidence scores and further justify our decision to use them, we measure the Expected Calibration Error (ECE) of three types of confidence scores: our verbalized score, length-normalized and entropy-based score. A lower ECE indicates stronger alignment between predicted confidence and actual correctness.

We use each dataset's training set as the calibration set, sending incomplete tuples and their top-5 retrieved tuples into the LLM for imputation under strict and relaxed modes respectively. The Show Movie dataset is excluded due to its small size. As shown in Table 4, verbalized scores consistently achieve the lowest ECE, confirming their superior calibration and suitability for threshold-based filtering. This may stem from the fact that, in the RAG setting, verbalized scores are more grounded in retrieved evidence than parametric knowledge, which is the common source of hallucination. Furthermore, we observe that ECE is lower under the strict mode compared to the relaxed mode, suggesting that rigorous assessment of retrieved tuples leads to more accurate confidence estimation.

*Finding 4. Our verbalized confidence scores are **better calibrated** than other probability-based methods.*

Table 5: Performance of retriever. Metric: Recall@100.

	WT	SM	ED	CP	ZM	FF
BM25	0.295	0.792	0.743	0.902	0.756	0.612
Contriever	0.449	0.042	0.628	0.074	0.085	0.704
DPR-scale	0.468	0.458	0.143	0.048	0.153	0.366
BERT w/ MLM	0.243	0.0	0.0	0.0	0.041	0.0
Sudowoodo	0.567	0.417	0.974	0.987	0.249	0.433
LakeFill's Retriever	**0.943**	**1.0**	**0.992**	**1.0**	**0.972**	**0.902**

Table 6: Recall@100 of retriever *v.s.* training datasets.

Training Datasets		WT	SM	ED	CP	ZM	FF
LakeFill		0.943	**1.0**	**0.992**	**1.0**	**0.981**	0.902
Anchor	w/ complete tuples	0.94	0.958	0.985	1.0	0.954	0.835
	w/ missing values	**0.949**	1.0	0.977	0.966	**0.981**	**0.908**
Positives	w/o tuples from other tables	0.924	1.0	0.962	0.967	0.973	0.749
Negatives	w/ augmented anchor without masked cells	0.851	0.833	0.93	1.0	0.949	0.881

6.4 Evaluation for Retriever

Exp-5: How does our Retriever compare with baselines? Table 5 shows that our **Retriever** achieves the highest recall and success rates on all datasets. The results indicate that: (1) Retrievers designed for passage retrieval tasks, *i.e.,* Contriever and DPR-scale, are not very effective for our task. (2) Contrastive learning is crucial for an effective tuple encoder for retrieval. **LakeFill**'s **Retriever** and BERT with MLM task share the same base model and training corpus, but the latter performs worst across all datasets, indicating its unsuitability for our scenario. (3) Our task differs from entity matching, as tuples describing the same entity are not necessarily relevant tuples.

*Finding 5. Our **Retriever**, leveraging contrastive learning, **outperforms** baselines in both recall and success rate across all datasets, demonstrating its superior effectiveness and generalization.*

Exp-6: What is the impact of synthesizing training data on Retriever? We evaluate the effectiveness of our synthesized training data and identify key factors in its construction, focusing on the anchor, positives, and negatives. The retrieval results, measured by recall@100, are presented in Table 6.

Table 7: Performance of reranker. Metric: S@5.

	WT	SM	ED	CP	ZM	FF
(1) Comparision with fine-tuned models (Full Data)						
monoBERT	0.781	0.875	0.982	**1**	0.99	0.979
monoT5	0.903	0.875	**0.987**	**1**	0.972	0.979
Rankllama2-7b	0.905	0.75	**0.987**	0.974	0.974	0.965
Rankllama3-8b	0.888	0.333	0.962	0.964	0.99	0.773
LakeFill's Reranker	**0.907**	**0.917**	0.986	0.995	**0.998**	**0.979**
(2) Comparision with prompting-based methods (Partial Data)						
GPT-4o-mini (pair)	0.82	0.875	0.995	0.994	0.995	0.83
GPT-4o-mini (list)	0.875	0.917	0.995	0.989	0.995	0.775
GPT-4o (pair)	0.86	0.917	0.995	**1**	0.99	0.81
GPT-4o (list)	0.88	0.917	0.995	**1**	0.99	0.78
LLaMA3-70B (pair)	0.86	0.917	0.995	**1**	0.985	0.845
LLaMA3-70B (list)	0.88	0.917	0.995	**1**	0.995	0.775
LakeFill's **Reranker**	**0.905**	**0.917**	**0.995**	0.994	**1.0**	**0.97**

Table 8: Performance of reranker that trained on the human-annotated training data. Metric: S@5. underline: Within 1% compared to LLM-annotated data.

Reranker	WT	SM	ED	CP	ZM	FF
monoBERT	0.771	0.875	0.984	1	0.98	0.979
monoT5	**0.918**	0.875	**0.989**	1	0.974	0.979
RankLLaMA2-7b	0.898	0.75	0.986	0.974	0.98	0.97
RankLLaMA3-8b	0.852	0.375	0.986	*0.964*	0.972	0.941
LakeFill's **Reranker**	0.895	**0.917**	0.981	**1**	**0.99**	**0.979**

(1) Anchor Tuples: We use two types of anchor tuples: complete tuples and those with masked cells (*i.e.,* simulating missing values to be filled). To show that combining these two types of anchor tuples yields better results, we construct datasets using only one type of them. From row 1 to 3, it is evident that combining both types improves retriever's performance on most datasets.

(2) Positive Tuples: Unlike traditional methods [73] that only consider augmented anchors as positives, we include relevant tuples from other tables in our synthesized data. Removing such positives (row 4) leads to a significant performance decline, highlighting the importance of diverse positive samples with various heterogeneous attributes, which aligns with real-world scenarios.

(3) Negative Tuples: We further explore using hard negatives, specifically, anchor variants with the key attribute removed, *i.e.,* tuples that are nearly identical to the anchor but omit the key information needed for imputation. We add this type of hard negative to the training data and report the corresponding result in row 5. We observe that this causes the sharpest drop in performance, probably because such cell-level variations are hard to distinguish very accurately in the embedding space.

Finding 6. The data synthesizing method for our **Retriever** *is very* ***effective****, with the combination of anchor tuples, positives and negatives proving crucial. Notably, synthesizing positive and negative samples has a more pronounced impact on performance than variations in anchor tuple construction.*

Table 9: Performance of reranker trained on the annotated data with only the highest score as positive (Metric: S@5).

	WT	SM	ED	CP	ZM	FF
monoBERT	0.612	0.875	0.984	1	0.992	0.953
monoT5	0.909	0.875	0.989	1	0.872	0.979
RankLLaMA2-7b	0.842	0.708	0.902	0.974	0.946	0.952
RankLLaMA3-8b	0.835	0.417	0.899	0.606	0.459	0.919
LakeFill	0.847	0.875	0.897	1.0	0.996	0.974

6.5 Evaluation for Reranker

Exp-7: How does the Reranker compare with baselines? We analyze fine-tuned and prompt-based reranking methods, evaluating fine-tuned models on the full dataset and prompt-based methods on a sampled subset due to cost constraints. Table 7 shows the results and we make the following observations.

(1) Compared to fine-tuned baselines, **Reranker** consistently achieves the best or comparable performance across most datasets when trained on our annotation data. While RankLLaMA models perform well on the Education dataset, they underperform on others, likely due to limited training data, which makes it difficult for large models to be fully well trained. As previously noted [32], their advantage as rerankers is not always evident. RankLLaMA2 shows more stable performance than RankLLaMA3, likely because reranking requires discriminative scoring rather than general instruction-following, the latter being LLaMA3's strength.

(2) Compared to prompt-based methods, **Reranker** performs comparably on several datasets and significantly outperforms them on FIFA, which contains many candidates from the same domain with similar schemas, requiring fine-grained distinction. Listwise prompting struggles in such settings due to the need to rank many similar tuples at once, while pairwise prompting, though slightly better, still lags behind **Reranker**. Moreover, prompt-based rerankers suffer from high computational cost. As shown in Table 10, listwise prompting takes about 30s per tuple and pairwise over 160s for top-100 reranking, over 80× slower than our lightweight, locally deployed **Reranker** (0.17s), making prompt-based approaches impractical for real-time use.

Finding 7. Our **Reranker** *achieves* ***better or comparable*** *performance to baselines and* ***offers substantial runtime advantages*** *over prompt-based methods.*

Exp-8: Can LLM-annotated data via evaluation checklist serve as an effective substitute for human annotations? We compare LLM-annotated data (via our checklist) with human-annotated data. As shown in Table 8, both monoT5 and **LakeFill** achieve strong performance. Furthermore, in most datasets, performance differences between models trained on LLM- and human-annotated data remain within 1%, indicating the high quality and reliability of our automatic annotations. On WikiTuples, monoT5 shows a larger drop under LLM-annotated data, possibly due to its reliance on binary classification and lack of robustness to label noise. In contrast, **LakeFill** benefits from our contrastive training strategy, which better captures fine-grained relevance distinctions and even leads to slight improvements over human annotations.

To further assess robustness of our construction method to label noise, we experiment with a simplified strategy by labeling only the highest-scoring tuple as positive and treating the rest as negatives. As shown in Table 9, this leads to noticeable performance drops across models, likely because some top-scoring tuples are still irrelevant. This confirms that our training group construction method provides more informative supervision by preserving the relative ranking among tuples.

*Finding 8. Our LLM-annotated data with stratified construction strategy achieves **comparable performance** with human-annotated data.*

Exp-9: Is Using an Evaluation Checklist Essential for Accurate Data Labeling? To validate the theoretical advantage of checklist-based annotation in Section 4, we empirically compare its labeling quality with binary relevance annotation. For the latter, we directly provide incomplete tuples and candidate tuples to GPT-4o. GPT-4o then analyzes whether the candidate can fill in the missing values in the incomplete tuple and labels it as relevant or irrelevant with an explanation. We asked LLM to label 100 training tuples of WikiTuples with 1 positive and 7 negatives. Note that this setup is artificial, as in real scenarios we wouldn't know the positives, but we aim to analyze the labeling performance here. The agreement rate between these labels and human annotations was 80.6%, while using the evaluation checklist annotations achieved over 95% agreement. This confirms that our proposed checklist leads to substantially more accurate labels, consistent with our theoretical analysis.

*Finding 9. Using the **evaluation checklist** for labeling **ensures much higher accuracy** in identifying relevant candidates, achieving an agreement rate of over 95%, compared to 80.6% directly using GPT-4o on WikiTuples dataset.*

6.6 Efficiency and Resource Analysis

We evaluate the run-time performance of **LakeFill** and some baselines in retrieval and reranking stages on WikiTuples datasets. The results are shown in Table 10.

Table 10 (a) summarizes the retriever performance. All methods use the *IndexFlatL2* type, which performs an exact search without compression. BM25 is fast to index since it avoids dense encoding, but suffers from high query-time latency. Sudowoodo, which performs blocking without indexing, is the slowest. RATA retrieves at the table level, reducing index size and retrieval time to about half of **LakeFill**, but at the expense of precision. **LakeFill** offers a good trade-off between accuracy and efficiency. Notably, if we switch **LakeFill** to use *HNSW* [53] index, retrieval time can be further reduced to *0.017 seconds per query* with only a slight drop (~5%) in recall@100. Table 10 (b) reports reranking performance. **LakeFill** achieves the highest efficiency due to its small model size. Among prompt-based methods, we report GPT-4o as representative, as other LLMs exhibit similar API latency. These methods are over 80× slower than **LakeFill**, making them impractical. In the imputation stage, **LakeFill** takes 7.49 seconds per incomplete tuple when querying GPT-4o with top-5 retrieved tuples.

Table 10: Retriever/reranker runtime on WikiTuples.

Method	Index Time (Index Size)	Retrieve Time (GPU Usage)
BM25	130s (12.74 GB)	29.96 s/q (cpu only)
Sudowoodo	–	214.9 s/q (2060MiB)
DPR_scale	7127.24 s (7.65 GB)	0.88 s/q (1432MiB)
RATA	1306.57s (5.83 GB)	0.415 s/q (934MiB)
LakeFill	8047.1 s (9.15GB)	0.808 s/q (936MiB)

(a) Retriever Performance.

Model	Rerank Time (GPU Usage)
GPT-4o (list)	29.6 s/q
GPT-4o (pair)	165 s/q
monoT5	0.53 s/q (2860MiB)
Rankllama2	4.23 s/q (14616MiB)
Rankllama3	3.99 s/q (16592MiB)
LakeFill	0.37 s/q (2268MiB)

(b) Reranker Performance.

7 RELATED WORK

Data imputation with limited data redundancy has gained attention with the advancement of LLMs. While some studies have applied LLMs directly to data imputation through in-context learning [57] or training models [38, 82], ensuring high accuracy and reliability remains a challenge. Retrieval-Augmented Generation (RAG), introduced by [35], involves retrieving relevant documents from external sources to generate answers. RAG has been adopted for many table-related tasks [25, 27], but the utilization of RAG in data imputation remains relatively unexplored. Previous imputation works incorporating retrieval ideas have limitations in addressing the challenges of imputation with limited redundancy. Some approaches retrieve information from the same table [40] or use simple matching [81], still relying on the table's inherent redundancy and failing to handle heterogeneous data. Others utilize external sources like master data [21, 28] or knowledge bases [19, 26], but require expert involvement, making them unsuitable for large-scale data lakes. RATA [25] employs a table-level retrieval for data imputation, but its coarse-grained retrieval is insufficient.

8 CONCLUSION

We introduce **LakeFill** for addressing data imputation in data lakes. **LakeFill** integrates a pre-trained retriever capable of identifying relevant tuples, a fine-tuned reranker to calculate fine-grained relevance, and a reasoner that applies in-context learning for the reliable imputation process. Our experiments demonstrate the exceptional effectiveness of **LakeFill**, surpassing various baselines and markedly improving upon methods that depend solely on LLMs.

ACKNOWLEDGMENTS

This work is partly supported by Guangdong provincial project 2023CX10X008, the NSF of China (62402409), Guangdong Basic and Applied Basic Research Foundation (2023A1515110545), Guangzhou Basic and Applied Basic Research Foundation (2025A04J3935), and Guangzhou-HKUST(GZ) Joint Funding Program (2025A03J3714). Ju Fan is supported by the NSF of China (62436010 and 62441230). Chengliang Chai is supported by the NSF of China (62472031), the National Key Research and Development Program of China (2024YFC3308200), Beijing Nova Program, CCF-Baidu Open Fund (CCF-Baidu202402), and Huawei.

REFERENCES

[1] Aug 06, 2024. GPT-4o. https://platform.openai.com/docs/models#gpt-4o.

[2] Jul 18, 2024. GPT-4o mini. https://platform.openai.com/docs/models#gpt-4o-mini.

[3] July 23, 2024. Llama-3-8b. https://huggingface.co/meta-llama/Llama-3.1-8B.

[4] Jun 30, 2023. bert-base-uncased. https://huggingface.co/bert-base-uncased.

[5] Jun 30, 2023. Chicago Data Portal. https://data.cityofchicago.org/.

[6] May 29, 2020. monobert-large-msmarco. https://huggingface.co/castorini/monobert-large-msmarco.

[7] Nov 14, 2023. Llama-2-7b. https://huggingface.co/meta-llama/Llama-2-7b.

[8] October 17, 2021. monot5-base-msmarco-10k. https://huggingface.co/castorini/monot5-base-msmarco-10k.

[9] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proc. VLDB Endow.* 9, 12 (2016), 993–1004.

[10] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley. http://webdam.inria.fr/Alice/

[11] 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).

[12] Naomi S. Altman. 1992. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician* 46 (1992), 175–185. https://api.semanticscholar.org/CorpusID:17002880

[13] Yavuz Faruk Bakman, Duygu Nur Yaldiz, Baturalp Buyukates, Chenyang Tao, Dimitrios Dimitriadis, and Salman Avestimehr. 2024. MARS: Meaning-Aware Response Scoring for Uncertainty Estimation in Generative LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 7752–7767. https://doi.org/10.18653/v1/2024.acl-long.419

[14] Felix Biessmann, Tammo Rukat, Phillipp Schmidt, Prathik Naidu, Sebastian Schelter, Andrey Taptunov, Dustin Lange, and David Salinas. 2019. DataWig: Missing value imputation for tables. *Journal of Machine Learning Research* 20, 175 (2019), 1–6.

[15] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional Functional Dependencies for Data Cleaning. In *ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*. 746–755. https://doi.org/10.1109/ICDE.2007.367920

[16] Bernardo Breve, Loredana Caruccio, Vincenzo Deufemia, Giuseppe Polese, et al. 2022. RENUVER: A Missing Value Imputation Algorithm based on Relaxed Functional Dependencies.. In *EDBT*. 1–52.

[17] Chengliang Chai, Nan Tang, Ju Fan, and Yuyu Luo. 2023. Demystifying Artificial Intelligence for Data Preparation. In *SIGMOD Conference Companion*. ACM, 13–20.

[18] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. 2023. Data Management for Machine Learning: A Survey. *IEEE Trans. Knowl. Data Eng.* 35, 5 (2023), 4646–4667.

[19] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1247–1261.

[20] 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.

[21] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *The VLDB journal* 21 (2012), 213–238.

[22] Alireza Farhangfar, Lukasz A. Kurgan, and Witold Pedrycz. 2007. A Novel Framework for Imputation of Missing Values in Databases. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 37, 5 (2007), 692–709. https://doi.org/10.1109/TSMCA.2007.902631

[23] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. Rethink training of BERT rerankers in multi-stage retrieval pipeline. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28–April 1, 2021, Proceedings, Part II 43*. Springer, 280–286.

[24] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[25] 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*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 5635–5648.

[26] Shuang Hao, Nan Tang, Guoliang Li, and Jian Li. 2017. Cleaning relations using knowledge bases. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 933–944.

[27] Jonathan Herzig, Thomas Mueller, Syrine Krichene, and Julian Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 512–519.

[28] Matteo Interlandi and Nan Tang. 2015. Proof positive and negative in data cleaning. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 18–29.

[29] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv:2112.09118* (2021).

[30] Gautier Izacard and Edouard Grave. 2020. Leveraging passage retrieval with generative models for open domain question answering. *arXiv:2007.01282* (2020).

[31] José M Jerez, Ignacio Molina, Pedro J García-Laencina, Emilio Alba, Nuria Ribelles, Miguel Martín, and Leonardo Franco. 2010. Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial intelligence in medicine* 50, 2 (2010), 105–115.

[32] Yuelyu Ji, Zhuochun Li, Rui Meng, and Daqing He. 2024. ReasoningRank: Teaching Student Models to Rank through Reasoning-Based Knowledge Distillation. arXiv:2410.05168 [cs.CL] https://arxiv.org/abs/2410.05168

[33] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[34] Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided supervision for openqa with colbert. *Transactions of the association for computational linguistics* 9 (2021), 929–944.

[35] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[36] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proc. VLDB Endow.* 17, 11 (2024), 3318–3331.

[37] Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025. Alpha-SQL: Zero-Shot Text-to-SQL using Monte Carlo Tree Search. In *Forty-Second International Conference on Machine Learning, ICML 2025, Vancouver, Canada, July 13-19, 2025*. OpenReview.net.

[38] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).

[39] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *Proc. VLDB Endow.* 14, 1 (2020). https://doi.org/10.14778/3421424.3421431

[40] Zhixu Li, Lu Qin, Hong Cheng, Xiangliang Zhang, and Xiaofang Zhou. 2015. TRIP: An interactive retrieving-inferring data imputation approach. *IEEE Transactions on Knowledge and Data Engineering* 27, 9 (2015), 2550–2563.

[41] Lei Liu, So Hasegawa, Shailaja Keyur Sampat, Maria Xenochristou, Wei-Peng Chen, Takashi Kato, Taisei Kakibuchi, and Tatsuya Asai. 2024. AutoDW: Automatic Data Wrangling Leveraging Large Language Models. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2041–2052. https://doi.org/10.1145/3691620.3695267

[42] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025. A Survey of Text-to-SQL in the Era of LLMs: Where are we, and where are we going? arXiv:2408.05109 [cs.DB] https://arxiv.org/abs/2408.05109

[43] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[44] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. Interactive Cleaning for Progressive Visualization through Composite Questions. In *ICDE*. IEEE, 733–744.

[45] Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li. 2022. Steerable Self-Driving Data Visualization. *IEEE Trans. Knowl. Data Eng.* 34, 1 (2022), 475–490.

[46] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *ICDE*. IEEE Computer Society, 101–112.

[47] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Trans. Vis. Comput. Graph.* 28, 1 (2022), 217–226.

[48] Yuyu Luo, Yihui Zhou, Nan Tang, Guoliang Li, Chengliang Chai, and Leixian Shen. 2023. Learned Data-aware Image Representations of Line Charts for Similarity Search. *Proc. ACM Manag. Data* 1, 1 (2023), 88:1–88:29.

[49] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting in Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 5303–5315. https://doi.org/10.18653/v1/2023.emnlp-

main.322

[50] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2023. Fine-Tuning LLaMA for Multi-Stage Text Retrieval. arXiv:2310.08319 [cs.IR] https://arxiv.org/abs/2310.08319

[51] Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023. Zero-Shot Listwise Document Reranking with a Large Language Model. *arXiv preprint arXiv:2305.02156* (2023).

[52] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.

[53] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (April 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[54] John T McCoy, Steve Kroon, and Lidia Auret. 2018. Variational autoencoders for missing data imputation with application to a simulated milling circuit. *IFAC-PapersOnLine* 51, 21 (2018), 141–146.

[55] 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.

[56] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? 16, 4 (Dec. 2022), 738–746. https://doi.org/10.14778/3574245.3574258

[57] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746. https://doi.org/10.14778/3574245.3574258

[58] Mona Nashaat, Aindrila Ghosh, James Miller, and Shaikh Quader. 2021. TabReformer: Unsupervised Representation Learning for Erroneous Data Detection. *ACM/IMS Transactions on Data Science* 2, 3 (2021), 1–29.

[59] Cheng Niu, Yuanhao Wu, Juno Zhu, Siliang Xu, KaShun Shum, Randy Zhong, Juntong Song, and Tong Zhang. 2024. RAGTruth: A Hallucination Corpus for Developing Trustworthy Retrieval-Augmented Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10862–10878. https://doi.org/10.18653/v1/2024.acl-long.585

[60] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[61] Barlas Oguz, Kushal Lakhotia, Anchit Gupta, Patrick Lewis, Vladimir Karpukhin, Aleksandra Piktus, Xilun Chen, Sebastian Riedel, Scott Yih, Sonal Gupta, and Yashar Mehdad. 2022. Domain-matched Pre-training Tasks for Dense Retrieval. In *Findings of the Association for Computational Linguistics: NAACL 2022*. Association for Computational Linguistics, Seattle, United States. https://doi.org/10.18653/v1/2022.findings-naacl.114

[62] Zhen Qin, Rolf Jagerman, Hui, et al. 2023. Large language models are effective text rankers with pairwise ranking prompting. *arXiv:2306.17563* (2023).

[63] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201.

[64] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

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

[66] Vraj Shah and Arun Kumar. 2019. The ML Data Prep Zoo: Towards Semi-Automatic Data Preparation for ML. In *Proceedings of the 3rd International Workshop on Data Management for End-to-End Machine Learning* (Amsterdam, Netherlands) *(DEEM'19)*. Association for Computing Machinery, New York, NY, USA, Article 11, 4 pages. https://doi.org/10.1145/3329486.3329499

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

[68] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. arXiv:2304.09542 [cs.CL]

[69] ZhongXiang Sun, Xiaoxue Zang, Kai Zheng, Jun Xu, Xiao Zhang, Weijie Yu, Yang Song, and Han Li. 2025. ReDeEP: Detecting Hallucination in Retrieval-Augmented Generation via Mechanistic Interpretability. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=ztzZDzgfrh

[70] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *Proc. VLDB Endow.* 14, 8 (2021), 1254–1261.

[71] Nan Tang, Chenyu Yang, Ju Fan, Lei Cao, Yuyu Luo, and Alon Y. Halevy. 2024. VerifAI: Verified Generative AI. In *CIDR*. www.cidrdb.org.

[72] Nan Tang, Chenyu Yang, Zhengxuan Zhang, Yuyu Luo, Ju Fan, Lei Cao, Sam Madden, and Alon Y. Halevy. 2024. Symphony: Towards Trustworthy Question Answering and Verification using RAG over Multimodal Data Lakes. *IEEE Data Eng. Bull.* 48, 4 (2024), 135–146.

[73] Runhui Wang, Yuliang Li, and Jin Wang. 2023. Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1502–1515.

[74] Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. 2023. KnowledGPT: Enhancing Large Language Models with Retrieval and Storage Access on Knowledge Bases. arXiv:2308.11761 [cs.CL] https://arxiv.org/abs/2308.11761

[75] Yifan Wu, Lutao Yan, Leixian Shen, Yinan Mei, Jiannan Wang, and Yuyu Luo. 2025. ChartCards: A Chart-Metadata Generation Framework for Multi-Task Chart Understanding. *CoRR* abs/2505.15046 (2025).

[76] Yifan Wu, Lutao Yan, Yizhang Zhu, Yinan Mei, Jiannan Wang, Nan Tang, and Yuyu Luo. 2025. Boosting Text-to-Chart Retrieval through Training with Synthesized Semantic Insights. *CoRR* abs/2505.10043 (2025).

[77] Yupeng Xie, Yuyu Luo, Guoliang Li, and Nan Tang. 2024. HAIChart: Human and AI Paired Visualization System. *Proc. VLDB Endow.* 17, 11 (2024), 3178–3191.

[78] Daniel Yang, Yao-Hung Hubert Tsai, and Makoto Yamada. 2024. On Verbalized Confidence Scores for LLMs. arXiv:2412.14737 [cs.CL] https://arxiv.org/abs/2412.14737

[79] 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.

[80] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025. AFlow: Automating Agentic Workflow Generation. In *ICLR*. OpenReview.net.

[81] 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.

[82] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023. TableLlama: Towards Open Large Generalist Models for Tables. *arXiv preprint arXiv:2311.09206* (2023).

[83] Yunfan Zhang, Changlun Li, Yuyu Luo, and Nan Tang. 2024. SketchFill: Sketch-Guided Code Generation for Imputing Derived Missing Values. arXiv:2412.19113 [cs.CL] https://arxiv.org/abs/2412.19113

[84] Zhengxuan Zhang, Zhuowen Liang, Yin Wu, Teng Lin, Yuyu Luo, and Nan Tang. 2025. DataMosaic: Explainable and Verifiable Multi-Modal Data Analytics through Extract-Reason-Verify. *CoRR* abs/2504.10036 (2025).

[85] Zhengxuan Zhang, Yin Wu, Yuyu Luo, and Nan Tang. 2024. MAR: Matching-Augmented Reasoning for Enhancing Visual-based Entity Question Answering. In *EMNLP*. Association for Computational Linguistics, 1520–1530.

[86] Zhengxuan Zhang, Yin Wu, Yuyu Luo, and Nan Tang. 2025. Fine-Grained Retrieval-Augmented Generation for Visual Question Answering. *CoRR* abs/2502.20964 (2025).

[87] Yizhang Zhu, Shiyin Du, Boyan Li, Yuyu Luo, and Nan Tang. 2024. Are Large Language Models Good Statisticians?. In *NeurIPS*.