BClean+: A Bayesian Data Cleaning System with Automated Prior Generation

Ziyan Han, Jing Zhu, Jinbin Huang, Sifan Huang, Yaoshu Wang, Rui Mao*, Jianbin Qin*

Abstract— **Probabilistic approaches, particularly Bayesian methods, are a cornerstone of data cleaning, yet they often depend on complex prior distributions that require costly and labor-intensive expert input. Our prior work, BClean, alleviated this burden by introducing automatic Bayesian network (BN) construction and lightweight user constraints (UCs), but it still fundamentally relies on manually provided prior knowledge. In this paper, we present BClean+, an enhanced Bayesian data cleaning system that extends BClean with a novel framework for automated prior generation. BClean+ leverages Large Language Models (LLMs) to identify attribute semantics and automatically synthesizes format patterns as UCs, while continuously maintaining a reusable template library. It also enhances BN construction through hierarchical structure discovery, improving interpretability and enabling more effective refinement for accurate inference. By integrating the automatically generated UCs into its Bayesian inference framework, BClean+ achieves more robust and accurate cleaning. Moreover, the framework generalizes to the synthesis of probabilistic programming language (PPL) code for systems such as PClean, thereby addressing a critical usability challenge in PPL-based data cleaning. Extensive experiments on real-world datasets demonstrate that BClean+ achieves an average F1-score of 0.89 (up to 0.98), outperforming state-of-the-art methods by 0.42 on average (up to 0.57), while reducing user configuration time from hours to under five minutes, with an average of 113.28× speedup in total runtime over BClean and other baselines. The source code of BClean+ is available at https://github.com/thethe-github/BCleanplus.**

Index Terms—**Data quality, data cleaning, Bayesian networks, probabilistic inference, probabilistic algorithms.**

I. Introduction

DATA cleaning is an essential step in preprocessing data for downstream applications such as data analysis and machine learning (ML) model development. Existing data cleaning solutions employ user-defined rules, outlier detection techniques, crowdsourcing, or knowledge bases to detect errors [1]–[3] and subsequently repair data [4]–[6]. The majority of research on data cleaning has focused on error correction using discriminative models based on integrity constraints [7]–[9], external data [10]–[14], statistical approaches [15], [16], ML techniques [17], or hybrid methods [6], [18]. Despite these advances, several challenges remain. Methods based on integrity constraints often require expert-defined rules, limiting their applicability in real-world scenarios. Approaches that rely on external data typically incur substantial costs for expert information collection. Moreover, ML methods face difficulties in learning reliable feature representations from noisy data.

To address these limitations, probabilistic methods [19], [20] have been proposed to perform data cleaning in a data-driven manner. These methods leverage probabilistic inference [21], for example through probabilistic graphical models (PGM). In addition, probabilistic programming languages (PPLs) [22]–[24] have emerged as powerful tools for specifying and executing probabilistic models. Another representative method is HoloClean [5], where probabilistic inference is used as a feature generator to evaluate the validity of denial constraints (DCs), one type of dependency rules. Compared with PPLs, rule configuration in HoloClean is relatively simpler. However, despite demonstrating robust cleaning performance in several empirical evaluations, HoloClean remains a semi-supervised approach that depends on manually annotated data. Other semi-supervised methods, such as Raha [3] and Baran [6], support automatic rule discovery and require labels for approximately 20 tuples each to perform error detection and correction. However, despite their ease of deployment, these methods are prone to error propagation from detection to correction. Currently, probabilistic inference-based data cleaning remains an active area of research, as PGMs offer natural advantages by treating dirty data as the observed outcomes of underlying generative processes [25].

In recent years, Bayesian methods, as a subset of probabilistic approaches, have been explored and shown promising results [4]. For instance, existing studies [25]–[27] employ Bayesian networks (BNs) to correct errors. These methods typically consist of two components: Bayesian network construction and inference. The construction phase involves structure learning and parameter estimation from the observed data, while the inference phase determines the most probable value for each data cell, conditioned on the other attributes of the same tuple and the learned data distribution. If the inferred value differs from the original, it is adopted as the corrected cell value.

Example 1: *Consider the Customer table in Table I, which contains 6 tuples with 9 attributes. Tuples 1–3 contain relatively simple errors that most existing methods can accurately identify and correct, as the context is clean and complete. For instance, functional dependencies (FDs) [28] can be leveraged to fix missing or inconsistent values: the FD InsuranceCode $\rightarrow$ InsuranceType imputes the missing value NULL with "Normal" in Tuple 1, while the FD ZipCode $\rightarrow$ State corrects the error "KT" to "CA" in Tuple 2. In addition, context-based methods (e.g., [6]) can repair typos such as correcting "25676x00" to "25676000" and "315 w hicky st" to "315 w hickory st".*

In contrast, Tuples 4–6 pose a greater challenge due to the

Ziyan Han, Jing Zhu, Jinbin Huang, and Sifan Huang are with Shenzhen University, Shenzhen 518060, China (e-mail: hanzy@szu.edu.cn; zhujing2023@email.szu.edu.cn; jbhuang@szu.edu.cn; huangsifan2020@email.szu.edu.cn).

Yaoshu Wang is with Shenzhen Institute of Computing Sciences, Shenzhen 518100, China (e-mail: yaoshuw@sics.ac.cn).

Rui Mao and Jianbin Qin are with SICS, Shenzhen University, Shenzhen 518060, China (e-mail: mao@szu.edu.cn; qinjianbin@szu.edu.cn).

*Corresponding author

TABLE I
CUSTOMER TABLE

Tid	Name	Department	Jobid	City	State	ZipCode	InsuranceCode	InsuranceType
1	Johnny.R	315 w hickory st	25676000	sylacauga	CA	35150	2567600035150	
2	Johnny.R	400 northwood dr	25676x00	sylacauga	KT	35150	2567600035150	Normal
3	Johnny.R	315 w hicky st	25676000	sylacauga	CA	35150	2567600035150	Normal
4	Henry.P	400 northwood dr	25600180	centre	KT		2560018035960	Low
5	Henry.P	400 nprthwood dr	25600180	centre	NY	3960	25600v5960	High
6	Henry.P		25600180	centre	KT	35960		Low

presence of numerous critical errors. The values of Department, InsuranceCode, InsuranceType, and State cannot be reliably inferred from limited evidence in Name, Jobid, and ZipCode. Due to the lack of observations, the errors "400 nprthwood dr", "NY", "3960", "25600v5960", and "High" may be misinterpreted as correct by probabilistic models. Such cases highlight the limitations of probabilistic inference under sparse observations and the need for richer priors to guide data repair.

To incorporate such priors, many existing data cleaning methods turn to external data [5], [11]. In Bayesian methods, domain knowledge is essential for encoding data distribution and BN structure. However, state-of-the-art Bayesian systems such as PClean [4] rely on hand-crafted network construction and priors, requiring users to author PPL programs that explicitly specify data types, compliant distributions, and possible noises (e.g., uncommon symbols or Gaussian noise). This demands specialized expertise and is both labor-intensive and error-prone, making it impractical for non-experts. For example, PClean requires users to precisely partition Table I into four parts: $P_1 = \{\mathsf{Name} \sim \mathsf{Dist}(\theta_1), \mathsf{Department} \sim \mathsf{Dist}(\theta_2), \mathsf{Jobid} \sim \mathsf{Dist}(\theta_3)\}$; $P_2 = \{\mathsf{City} \sim \mathsf{Dist}(\theta_4), \mathsf{State} \sim \mathsf{Dist}(\theta_5), \mathsf{ZipCode} \sim \mathsf{Dist}(\theta_6)\}$; $P_3 = \{\mathsf{InsuranceCode} \sim \mathsf{Dist}(\theta_7), \mathsf{InsuranceType} \sim \mathsf{Dist}(\theta_8)\}$; and $P_4 = \{P_1, P_2, P_3, \mathsf{error\_dist}(\theta_9)\}$, where $A \sim \mathsf{Dist}(\theta)$ in each part denotes that attribute A follows a distribution with parameter θ. The partition specification is inherently difficult and error-prone, and becomes increasingly infeasible for datasets with large schemas.

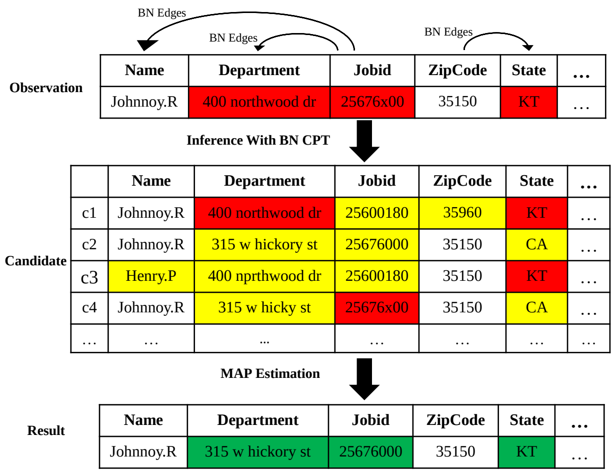
This example illustrates a primary challenge of existing Bayesian data cleaning systems: their heavy reliance on expert-provided prior knowledge. Our prior work, BClean [29], attempted to mitigate this barrier by introducing automatic BN construction and lightweight user constraints (UCs), such as regular expressions or value ranges. However, both hand-crafted PPL code and user-defined UCs remain fundamentally dependent on manual specification, posing a critical bottleneck for non-technical users and for datasets with large schemas. This highlights the need for a universal framework that can automatically generate rich, structured priors, eliminating the dependence on manual prior specification.

To bridge this gap, we propose BClean+, an enhanced Bayesian data cleaning system that emphasizes automation. BClean+ extends BClean by incorporating a novel framework for automated prior generation and restructuring the pipeline into two primary stages: (1) **Automated Knowledge Acquisition and Modeling**. This stage constructs a foundational Bayesian network to capture attribute dependencies and applies community detection to identify higher-level semantic modules. It then employs an LLM for zero-shot semantic labeling of attributes, which guides the synthesis of format patterns that serve as UCs; and (2) **Inference and Repair**. In this stage, BClean+ performs data cleaning by leveraging the core Bayesian inference engine, compensatory scoring model, and optimization strategies from BClean. By integrating automatically generated UCs, BClean+ not only enhances system usability but also leverages richer priors to achieve more robust and accurate inference than BClean.

The significant advancement of BClean+ lies in the automation of constraint generation. While BClean has already reduced user effort by allowing simple expressions for constraint definition, BClean+ goes further by generating these constraints automatically. Users no longer need to write rules from scratch; instead, their role shifts to optionally validating or fine-tuning the high-quality constraints synthesized by the system. Moreover, the framework can also map identified semantic types to corresponding distributions and syntactic patterns, thereby providing a foundational pathway for the automatic generation of more complex PPL code. This extension enables applicability to PPL-based systems such as PClean, addressing a key usability challenge. It dramatically lowers the barrier to entry for non-experts, while retaining the flexibility to support complex, user-defined functions for specific domain requirements.

The contributions of this paper are summarized as follows:

- We propose BClean+, an enhanced Bayesian-inference-based data cleaning system that extends BClean with automated prior generation to improve practicality and usability.
- We develop a novel framework for automated prior generation, which automatically identifies the semantic types of attributes and synthesizes high-quality format patterns as user constraints (UCs), while maintaining a reusable template library that grows with new datasets and serves as a shared resource.
- We enhance Bayesian network construction through hierarchical structure discovery, grouping attributes into higher-level semantic modules. This design improves interpretability, enables effective refinement, and supports automated PPL synthesis.
- We employ Large Language Models (LLMs) as a key component of our framework to enable *zero-shot* semantic type identification, bootstrapping the entire automation process.
- We also demonstrate the generalizability of our framework by integrating automatically learned structural dependencies (to construct the data relationship models in PPLs) with heuristic analysis of inherent data characteristics (to select and configure probabilistic distribution models). This enables the automated synthesis of complex PPL code and addresses a core usability challenge in existing PPL-based Bayesian cleaning systems.
- We conduct extensive experiments on real-world datasets to validate the effectiveness and efficiency of BClean+, demonstrating its state-of-the-art performance.



Name	Department	Jobid	ZipCode	State	…
Johnnoy.R	400 northwood dr	25676x00	35150	KT	…

	Name	Department	Jobid	ZipCode	State	…
c1	Johnnoy.R	400 northwood dr	25600180	35960	KT	…
c2	Johnnoy.R	315 w hickory st	25676000	35150	CA	…
c3	Henry.P	400 nprthwood dr	25600180	35150	KT	…
c4	Johnnoy.R	315 w hicky st	25676x00	35150	CA	…
…	…	...	…	…	…	…

Name	Department	Jobid	ZipCode	State	…
Johnnoy.R	315 w hickory st	25676000	35150	KT	…

Fig. 1. A running example of data cleaning. Red, yellow, and green represent erroneous, candidate, and clean cell values, respectively.

The remainder of this paper is organized as follows. Section II introduces preliminaries. Section III provides an overview of BClean+ methodology. Sections IV–VII present the core technical components of BClean+. Section VIII extends the framework to automated PPL code synthesis. Section IX reports the experimental results. Section X reviews related work. Section XI concludes the paper.

II. Preliminaries

The aim of BClean+ is to repair erroneous data in a structured dataset D, thereby producing a cleaner version D^*. The dataset D consists of n tuples $\{T_1, \ldots, T_n\}$ defined over m attributes $\mathcal{A} = \{A_1, \ldots, A_m\}$. The domain of attribute A_j is denoted by $dom(A_j)$, and $T_i[A_j]$ represents the observed value of the j-th attribute in the i-th tuple. To repair the dataset, BClean+ considers candidate values $c \in dom(A_j)$ and selects the most probable one c^* to replace $T_i[A_j]$ for all $i \in [1, n]$ and $j \in [1, m]$. This process is formulated as a maximum a posteriori (MAP) inference problem that leverages attribute dependencies to identify the most likely repair. In line with prior work on data cleaning systems [3], [5], [6], BClean+ targets the repair of both syntactic and semantic errors, including typos, missing values, inconsistencies, formatting issues, violations of functional dependencies, and swapping-value errors.

Bayesian networks (BNs) are among the most effective theoretical models for uncertain knowledge representation and inference [21]. In general, a BN is a directed acyclic graph (DAG) (N, E, θ), where N is a set of nodes representing random variables, E is a set of directed edges indicating the conditional dependencies between random variables, and θ is a set of conditional probability tables (CPTs) that quantify the strength of these dependencies. The probability of each random variable is determined by the probabilities of its parent nodes; if no parents exist, the distribution is specified by prior information.

In BClean+, each attribute is modeled as a node in the BN, and edges capture dependencies among attributes. This design allows the system to naturally incorporate attribute relationships into the MAP framework. For each cell, BClean+ generates candidate values and computes their probabilities using MAP estimation with the BN. Fig. 1 illustrates this process on a sample tuple, where dependencies guide the repair of erroneous values. From an observed dataset D, both the BN structure and its CPTs can be learned. Fig. 2 presents an example BN learned from the dataset in Table I.

Formally, given a tuple $T = (t_1, \ldots, t_m)$ from the dataset D, its joint probability can be defined as follows:

$$\mathbf{Pr}[t_1, ..., t_m] = \prod_{A_i \in \mathcal{A}} \mathbf{Pr}[T[A_i] = t_i | T[A_j] = t_j, j \in Ans(i)], \tag{1}$$

where $Ans(i)$ represents the set of ancestor nodes of A_i. For nodes without any ancestor nodes, $\mathbf{Pr}[T[A_i] = t_i | T[A_j] = t_j, j \in Ans(i)]$ is determined by the prior probability of A_i, which can be estimated directly from D.

While BNs capture statistical dependencies, additional user-specified constraints are required to guide the inference, ensuring that candidate repairs satisfy semantic or domain-specific requirements (e.g., a U.S. ZIP code must be five digits). We use the term user constraint (UC) to denote any binary function $UC(\cdot)$ that verifies whether an input (cell, tuple, or dataset) adheres to user-specified constraints, returning 1 if satisfied and 0 otherwise. UCs can take various forms, including rules (e.g., FDs and DCs), arithmetic expressions, regular expressions, or even deep neural networks. In BClean, UCs were restricted to a few straightforward forms: (1) attribute statistics such as minimum/maximum lengths or value range, (2) non-null constraints, and (3) simple format specifications via regular expressions for digits and dates. These lightweight UCs serve as priors without requiring database expertise, and even users unfamiliar with regular expressions can rely on online tools such as Regex Generator++ [30], [31] to generate them from examples. However, despite their simplicity, these UCs still fundamentally rely on manual specification, which poses a bottleneck for non-technical users and for large datasets.

BClean+ addresses this limitation by extending UCs beyond these restricted forms in BClean. It generalizes format constraints to cover all attribute types and, more importantly, introduces an automated prior generation framework that synthesizes richer, semantically meaningful patterns, thereby greatly reducing manual effort (see Section V). Consequently, users no longer need to be familiar with databases or advanced techniques used in systems like HoloClean, such as conditional FDs [32] and metric FDs [33], nor are they required to label a non-trivial number of tuples (e.g., approximately 40 in Raha and Baran). These improvements make BClean+ more broadly applicable.

III. Methodology of BClean+

Fig. 2 provides an overview of BClean+, an end-to-end data cleaning system with the following two primary phases.

Modeling and knowledge acquisition. The process begins with data relationship modeling, where a foundational BN is constructed by extending the FDX method [28] and utilizing the graphical lasso technique [34], [35] to capture attribute dependencies. The resulting BN skeleton is then analyzed using the Infomap algorithm [36] to identify higher-level semantic modules (e.g., a "location" module), yielding a richer, multi-level structural view of data. To further automate prior knowledge acquisition, it employs LLMs for zero-shot semantic

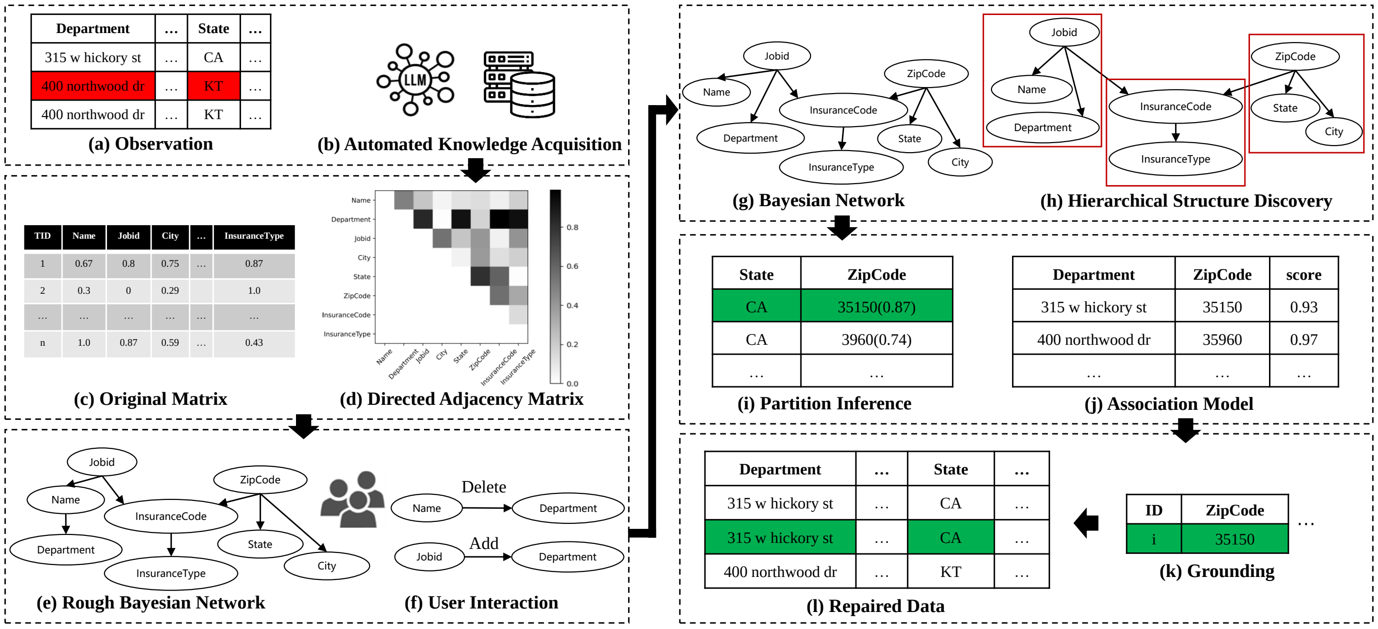


Fig. 2. An overview of the BClean+ framework.

$$\boxed{Pr[c]} \overset{\text{Bayes Rule}}{=} \boxed{\frac{Pr[t]Pr[c|t]}{Pr[t|c]}}$$

$$\text{Transform} \Downarrow$$

$$\boxed{\underbrace{\log Pr[c|t]}_{\text{BN Score}} + \underbrace{(\log Pr[t] - \log Pr[t|c])}_{\text{Compensative Score}}}$$

Fig. 3. The data-driven inference model of BClean+.

labeling of attributes and synthesizes high-quality format patterns as UCs, while maintaining a reusable template library that stores the inferred semantic types, patterns, and distribution models. Leveraging this library, BClean+ supports the automatic synthesis of PPL code for PPL-based systems such as PClean.

Inference grounding and pruning. In this stage, candidate values for data cleaning are generated by iterating over the domain values of each attribute, and their probabilities are subsequently computed, with the most probable value selected as the repair. Since the number of candidate values can be vast, BClean+ reduces unnecessary generation through three techniques. First, it partitions the BN by leveraging the Markov property [21]. Second, within each partition, it prunes the domain of each attribute by eliminating values that are inconsistent with domain semantics. Third, it applies the compensatory score to perform preliminary detection and inference for variables that are likely to be correct.

We formulate the data cleaning task as a MAP problem to determine the optimal data distribution. Given a dataset D, the goal is to derive a cleaned dataset D^* by inferring, for each tuple $T_i \in D$ and attribute $A_j \in \mathcal{A}$, the most probable candidate value $c^* \in dom(A_j)$. Conditioning on the other attributes of T_i, we apply Bayes' rule and compute c^* via MAP as follows:

$$c^* = \arg\max_{c \in dom(A_j)} \frac{\mathbf{Pr}[t]\mathbf{Pr}[c|t]}{\mathbf{Pr}[t|c]}, \text{ subject to } UC(c) = 1, \quad (2)$$

where t denotes the observed values of attributes other than A_j in T_i, i.e., $t = T_i[A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_m]$. By transforming it to logarithmic form, we obtain

$$c^* = \arg\max_{c \in dom(A_j)} (\log \mathbf{Pr}[c|t] + \log \mathbf{Pr}[t] - \log \mathbf{Pr}[t|c]),$$
$$\text{subject to } UC(c) = 1. \quad (3)$$

The terms in Eq. (3) can be divided into two parts: $\log \mathbf{Pr}[c|t]$ and $\log \mathbf{Pr}[t] - \log \mathbf{Pr}[t|c]$. The first term can be obtained from the BN using the parent nodes of A_j. The second term, which cannot be directly computed from the BN, is estimated through a compensatory scoring model, detailed in Section VI. Fig. 3 illustrates the data-driven inference model of BClean+.

Algorithm 1 outlines the high-level procedure for cell-level repair in BClean+. It iterates over all rows and columns of the observed dataset D and infers the most probable value for each cell. For each cell, c^* is initialized as the original value $T_i[A_j]$. Then, for each candidate satisfying UCs, its probability is computed using the BN and the compensatory scoring model (Eq. (3)), and c^* is updated if a higher probability is obtained. The collection of all c^* values constitutes the cleaned dataset D^*, which is returned by the algorithm.

User interaction and automation modes. BClean+ is designed to minimize manual effort while still allowing users to inject domain knowledge when necessary. In practice, BClean+ provides three *optional* points of user interaction: (i) *BN validation and modification*, where users can view and modify the structures of automatically constructed BN; (ii) *UC validation and refinement*, where users can add, delete, or edit the synthesized UCs before inference; (iii) *PPL code validation and refinement*, where users can refine and confirm the PPL code automatically synthesized from the template library when targeting PPL-based systems such as PClean. Users are free to skip any or all of these steps, in which case BClean+ runs as a fully automatic cleaning pipeline driven solely by the learned priors. As shown in Section IX, this automatic mode already yields strong performance, while

Algorithm 1: Cell-level Repair in BClean+

```
Input  : An observed dataset $D$ with $n$ rows and $m$
         columns, and a Bayesian network $BN$.
Output : The cleaned dataset $D^*$.
1  for $i \leftarrow 1$ to $n$ do
2      for $j \leftarrow 1$ to $m$ do
3          $c^* \leftarrow T_i[A_j]$;
4          $p^* \leftarrow \log(BN[A_j](c^*)) + \log(CS[A_j](c^*))$;
5          foreach $c \in dom(A_j)$ s.t. $UC(c) = 1$ do
6              $p \leftarrow \log(BN[A_j](c)) + \log(CS[A_j](c))$;
7              if $p > p^*$ then
8                  $c^* \leftarrow c$; $p^* \leftarrow p$;
9          $T_i^*[A_j] \leftarrow c^*$; // Clean ($i$-th row, $j$-th column)
10 return $D^* = \{T_1^*, \dots, T_n^*\}$;
```

lightweight interventions (typically within a few minutes per dataset) can further improve the overall cleaning quality.

IV. Data Relationship Modeling

In this section, we propose a two-stage framework to model data relationships at multiple levels: (i) Bayesian network construction (Section IV-A), which constructs a BN that captures dependencies among individual attributes; and (ii) hierarchical structure discovery (Section IV-B), which groups attributes into higher-level semantic modules from the BN, providing a more macroscopic and interpretable view to guide BN modification and subsequent automated PPL synthesis (see Section VIII).

A. Bayesian Network Construction

We first introduce our approach for constructing a BN; the BN partitioning technique will be discussed in Section VII.

The problem of constructing a BN from a given dataset is NP-hard [37]. Consequently, exhaustive search over all possible structures is infeasible for large datasets. Many existing methods therefore adopt heuristics to approximate a solution. For instance, hill-climbing-based approaches, such as MMHC [38] available in the Pgmpy toolkit [39], incrementally add edges and evaluate their scores to determine each edge and its direction. However, such approaches often converge to a local optimum, limiting their effectiveness. Other typical approaches include tree search, which requires a pre-specified root state, and the PC algorithm [40], which relies on user-provided conditional independence hypotheses.

In BClean+, we construct a BN from the observed dataset D, as illustrated in Fig. 2. Automatic BN construction is particularly challenging because it either requires accurate data or prior knowledge (e.g., as specified using PPL [4]), whereas D is often unclean and sparse. We aim to provide a supportive system that obviates the need for users to initiate BN construction from scratch. This makes the aforementioned heuristic approaches ill-suited to our setting. To address this, BClean+ leverages a structure learning algorithm that tolerates noise and incorporates domain knowledge, thereby constructing an approximate BN.

Specifically, we extend the FDX method [28] with error tolerance to generate an *inverse covariance matrix* using *graphical lasso* [34], from which a BN skeleton is derived. Our method is statistical in nature; it models the distribution underlying the observed data while capturing attribute relationships via FDs. An FD $X \rightarrow Y$ states that $T_i[Y] = T_j[Y]$, if $T_i[X] = T_j[X]$. Given the presence of errors in the dataset, the application of strict FDs lacks flexibility. Therefore, we propose to soften the FDs by introducing a similarity measure: for any pair of tuples $T_i, T_j \in D$, we denote the similarity between their two attribute values as $Sim(T_i[X], T_j[X]) \in [0, 1]$, which serves as a probability in our modeling. For numerical attributes, we use $\frac{|T_i[X]-T_j[X]|}{(|T_i[X]|+|T_j[X]|)/2}$; for string attributes, we use unit-cost edit distance normalized by string lengths as follows:

$$Sim(T_i[X], T_j[X]) = 1 - \frac{2 \cdot ED(T_i[X], T_j[X])}{len(T_i[X]) + len(T_j[X])}, \quad (4)$$

where ED denotes unit-cost edit distance (i.e., Levenshtein distance) and len denotes string length. For example, in the Department attribute of Tuples 1 and 3 in Table I, the derived feature equals 1 because they refer to the same entity. In contrast, strict FDs would yield a violation (0), whereas our softened measure reports a similarity of 0.86, thereby tolerating errors in the dataset.

We compute pairwise attribute similarities within each tuple and regard these similarity values as observations from a multivariate Gaussian distribution. The graphical lasso [34] is then utilized to calculate the covariance matrix Σ of the underlying distribution. Following the approach commonly adopted in structure learning for linear models [35], [41] and employed in FDX [28], we further decompose the inverse covariance matrix $\Theta = \Sigma^{-1}$ as:

$$\Theta = \Sigma^{-1} = (I - B)\Omega(I - B)^T, \quad (5)$$

where Θ denotes the inverse covariance matrix for pairwise attributes in $\mathcal{A}$, I is the identity matrix, and B is the autoregression matrix of the model, which is exactly the adjacency matrix that stores the weights of the edges of the BN skeleton. A weight threshold is then applied to retain only significant edges with weights exceeding the threshold.

Upon constructing the BN skeleton, we perform parameter learning to estimate the joint probability of attributes and derive the CPTs concerning the observations. It is important to note that, during the construction of the BN skeleton, observational errors are not explicitly addressed; instead, our BN construction models errors as part of the distribution. Since the automatically constructed BN skeleton may contain inaccuracies, we provide an interface that allows users to review and manually modify the structure, thereby generating the BN used for inference. Users can refine the BN by directly adding or deleting edges. To facilitate this process, the system employs automated hierarchical structure discovery to cluster semantically related attributes into modules, providing a more macroscopic and intuitive perspective to guide BN modification (see Section IV-B). Since edges are associated with CPTs, the CPTs need to be updated if users modify the BN. For efficiency, we recalculate only the CPTs of the attributes involved in the modification instead of all attributes.

Example 2: *Consider the dataset D in Table I, we first compute pairwise similarities, as shown in Fig. 2(c). We then compute the inverse covariance matrix and decompose it to obtain the adjacency matrix (Fig. 2(d)) of the BN skeleton, where*

(ZipCode, InsuranceCode) is represented as a directed edge because its weight exceeds the threshold (Fig. 2(e)). The BN skeleton can then be refined through user modifications, such as adding or deleting edges (Fig. 2(f)), yielding the final BN employed for inference (Fig. 2(g)).

Remarks. To use the FDX method, we first sort tuples by each attribute, and only compute similarities or check equality within two adjacent tuples. As such, we do not need to compute each tuple pair in D. Although the BN construction is not based on all tuple pairs, it is demonstrated to be effective in our experiments (see Table V). We construct the BN by using the FDX method and extend it to support fuzzy matching, e.g., edit similarity. The time complexity is $O(nm \log n + nm + r|B|)$, where r is the number of iterations of graphical lasso learning, and $|B|$ is the size of the autoregression matrix.

B. Hierarchical Structure Discovery

The constructed BN encodes local, direct dependencies between individual attributes. While accurate at a micro-level, this flat view fails to explicitly capture higher-level semantic concepts that naturally arise in real-world datasets, where attributes often group together to describe a single entity (e.g., City, State, and ZipCode collectively define a location). Moreover, for datasets with large schemas, such flat representations can become overly complex and difficult to interpret.

To address these limitations, we introduce hierarchical structure discovery, which automatically discovers higher-level attribute modules from the learned BN in an unsupervised manner. By clustering semantically related attributes (e.g., City, State, and ZipCode) into modules (e.g., a "location module"), it provides a more macroscopic and intuitive view of the data. This hierarchical perspective not only facilitates human interpretation and enables more effective BN modification, but also provides rich structured prior knowledge that lays the groundwork for subsequent automation, including PPL synthesis.

Building on the BN-derived attribute dependency graph, we aim to identify groups of attributes that are densely interconnected. To this end, we adopt Infomap [36], a community detection algorithm rooted in information theory and widely recognized for its effectiveness in discovering modular structures in complex networks. Infomap functions by optimizing the Map Equation [36], [42], which evaluates the quality of a partition by minimizing the description length of a random walker's movements. Intuitively, if a random walker tends to remain within a subset of nodes for an extended period, that subset constitutes a cohesive community.

Applying Infomap to the attribute dependency network derived from the BN yields a partition of the attribute set $\mathcal{A}$ into k distinct communities (or modules), denoted by $C = \{C_1, C_2, \dots, C_k\}$. Each community C_i represents a semantically coherent group of attributes that are strongly interdependent, while connections between different communities are comparatively weaker.

Example 3: *Continuing with Example 2, when the Infomap algorithm is applied to the BN (Fig. 2(e)), it automatically discovers the following partitions, shown in Fig. 2(h):*

- *Community C_1 (Location Module): {City, State, ZipCode}, which are geographically interdependent and collectively define a precise location.*
- *Community C_2 (Identity Module): {Name, Department, Jobid }, which jointly represent the personal and professional identity of a customer.*
- *Community C_3 (Insurance Module): {InsuranceCode, InsuranceType }, which are directly related and form a distinct module of business-related information.*

In the "Identity" module, for example, grouping Name, Department, and Jobid suggests that an edge from Jobid to Department is reasonable, while a spurious edge from Name to Department should be removed. Building on this modular view (Fig. 2(h)), user modifications (Fig. 2(f)) can be systematically guided to yield the final BN (Fig. 2(g)).

This automated partitioning not only validates the structural soundness of the learned BN from an intuitive perspective, but more importantly, it enriches the data with a macroscopic semantic structure that aligns with human cognition and extends beyond localized dependencies. This hierarchical discovery is not merely a theoretical exercise; it provides the direct and practical foundation for automated PPL synthesis. Automatically identified modules can be programmatically translated into PPL code segments (see Section VIII). This addresses the manual grouping challenge inherent in PPL-based systems such as PClean, thereby significantly reducing the entry barrier for non-technical users.

Remarks. We apply the Infomap algorithm to the constructed BN to automatically discover macro-level semantic modules. The time complexity of this stage is $O(|E|)$, reflecting the near-linear cost of the Infomap community detection on a network with $|E|$ edges. Together with BN construction, the overall complexity is $O(nm \log n + nm + r|B| + |E|)$. This integrated hierarchical modeling strategy yields a comprehensive, multi-level structural view of the data, laying a solid foundation for automated cleaning.

V. Automated Semantic Distribution Modeling

As previously discussed, a major limitation of existing Bayesian cleaning systems is their heavy reliance on expert-provided prior knowledge. To address this, BClean+ introduces a novel automated prior generation framework that transforms raw data columns into structured priors through two stages: (i) semantic type identification, which identifies the underlying semantics behind data (Section V-A); and (ii) semantic-aware modeling, which leverages the identified semantic types to guide the synthesis of the corresponding format pattern templates (Section V-B).

A. Semantic Type Identification via Large Language Models

The automated generation of high-quality priors fundamentally relies on an accurate understanding of data semantics. Without a clear identification of the semantic type of each attribute, the subsequent pattern synthesis would lack the necessary guidance and fail to produce meaningful constraints. Therefore, we begin by performing semantic type identification to assign a precise semantic label to each attribute. To this end, we adopt the ArcheType framework [43] for its powerful zero-shot classification capability and compatibility with diverse models,

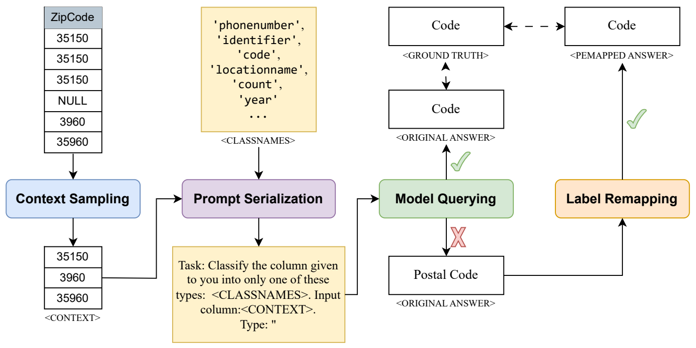


Fig. 4. The four-stage ArcheType framework for semantic type identification.

INSTRUCTION: Select the option which best describes the input.
INPUT: [10001, 10002, 10003, 10004, 10005, ...]
OPTIONS: Identifier, Code, PersonName, OrganizationName, LocationName, ObjectName, Address, DateTime, Age, Count, Measurement, Money, Percentage, Score, Category, Text, LanguageName, Email, PhoneNumber, URL, Boolean, Year

ANSWER: Identifier

Fig. 5. The example prompt of our zero-shot application.

making it well-suited for building an extensible pattern template library. As illustrated in Fig. 4, the identification process follows a fine-grained, four-step automated workflow.

(1) **Context Sampling**. Given the input token limits of most LLMs, the framework avoids processing an entire column of data; instead, it samples a representative subset of values using a weighted probability strategy, where an importance function can be customized for each attribute, e.g., assigning higher weights to longer strings (which carry more information), or to values containing keywords that indicate potential types.

(2) **Prompt Serialization**. The sampled values are integrated into a structured prompt that combines (i) the sampled context, (ii) a task-specific instruction (e.g., "Please select the best matching type for this column"), and (iii) a set of candidate semantic labels drawn from our template library. To effectively specify the task to the LLM, the framework supports multiple serialization strategies. An example prompt is shown in Fig. 5.

(3) **Model Querying**. The serialized prompt is submitted to an LLM. The framework is compatible with a wide range of models, such as the GPT series and T5. In our implementation, we use the accessible yet high-performing flan-t5-xxl model. The LLM leverages its pre-trained world knowledge to infer the most appropriate semantic description for the given data samples.

(4) **Label Remapping**. The final step normalizes the raw output of the model, which may not exactly match the canonical labels in our template library (e.g., "State Names" instead of "State"). This normalization uses techniques such as string containment checks or vector similarity to map the free-form prediction of the model to a valid label, thereby ensuring standardized outputs. Examples of predefined labels used in this paper are shown in Fig. 5.

This framework ensures both adaptability and robustness. Its zero-shot capability enables the system to handle new semantic types without costly retraining, while the state-of-the-art performance guarantees accuracy. Moreover, compatibility with open-source models ensures reproducibility for future research.

B. *Semantic-Aware Pattern Synthesis*

The accuracy of semantic type identification is crucial, as it directly guides the synthesis of high-quality format patterns. Our experiments (Section IX-C) confirm that these patterns, typically expressed as regular expressions, play a dominant role in data cleaning performance. When precise patterns are unavailable, both precision and recall degrade significantly. However, this heavy reliance on high-quality, user-provided patterns remains a major bottleneck, as manually writing regular expressions is labor-intensive, error-prone, and requires technical expertise.

To address this challenge, we develop a mechanism that automatically infers and synthesizes format patterns from data samples. Integrated with the semantic types identified in Section V-A, this mechanism ensures the inferred patterns remain targeted and accurate. The process consists of two main stages:

- **Sample Aggregation.** For each semantic type, the system first aggregates a representative subset of non-empty values from all attributes assigned to that semantic label.
- **Pattern Generation and Synthesis.** With the aggregated samples, it employs a dual strategy to generate candidate patterns.
  - Pattern Generation: We employ the rexpy module from the Test-Driven Data Analysis (tdda) library [44] to automatically infer regular expressions from sample values, thereby capturing latent, overarching formats across the dataset.
  - Pattern Synthesis: To accommodate diversity and potential noise in the samples, additional patterns are synthesized by combining elements from an internal library of common atomic patterns (e.g., `d\{3\}` for a three-digit number, and `[A-Z][a-z]+` for a capitalized word).

The collection of patterns generated and synthesized is stored as entries in a *reusable template library* and linked to their corresponding semantic types; each finalized after optional user validation. As shown in Table II, each record in the library contains (i) a semantic type label, (ii) candidate format patterns, and (iii) candidate distribution models that are used for synthesizing PPL code (Section VIII). This template library builds upon the patterns already accumulated in BClean and is continuously expanded and maintained in BClean+. Each newly validated pattern is added back into the library, enabling it to evolve into a growing repository of structured priors.

Template library reuse scenarios. In practice, the template library is reused across cleaning tasks in three typical ways.
(i) *Onboarding repeated cleaning tasks on the same dataset.* When BClean+ reruns on a dataset that has been cleaned before, semantic type identification reuses the stored type labels, and the system retrieves the best-matching patterns from the library, with only minimal user validation when necessary.
(ii) *New datasets with new schemas or distributions.* For a new dataset, BClean+ first infers the semantic type of each attribute. If the inferred type exists in the library, the corresponding patterns are selected and reused, with light parameter re-estimation of the associated distribution models on the new data; if these patterns no longer provide sufficient coverage, BClean+ additionally invokes automatic pattern generation to augment the pattern set. When the inferred type does not exist in the library, BClean+ directly invokes automatic prior generation and inserts newly validated patterns and distribution models into the library.

TABLE II
EXAMPLES OF ENTRIES IN THE REUSABLE TEMPLATE LIBRARY, INCLUDING SEMANTIC TYPES, FORMAT PATTERNS, AND DISTRIBUTION MODELS

Semantic type	Pattern 1	Pattern 2	Pattern 3	Distribution 1	Distribution 2
year	^([2][0][0-9][0-9])$	^([1-2][0-9][0-9][0-9])$	^[0-9]{4}$	ChooseUniformly	ChooseProportionally
code	^[0-9]{4,5}$	^\d+$	^[a-zA-Z0-9]+$	ChooseUniformly	ChooseProportionally
datetime	^[0-9]{1,2}:[0-9]{2} [a-z]\.m\.$	^\d{8}$		TimePrior	
percentage	^[0-9]{1,3}%$	^\d\d[%]$	^\d+$	ChooseProportionally	ChooseUniformly
organizationname	^[\u4e00-\u9fffa-zA-Z\s]+$	^[A-Za-z]+$	^[a-zA-Z0-9-]+$	StringPrior	ChooseProportionally
address	^[a-zA-Z0-9]+$	^(?=.*[a-zA-Z])(?=.*\d)(?=.*\s)[a-zA-Z0-9\s-]+$	^[A-Z\s.-]+$	StringPrior	ChooseUniformly
category	^[A-Z][2]$	^[\u4e00-\u9fffa-zA-Z\s]+$	^[a-zA-Z]+$	ChooseUniformly	ChooseProportionally
identifier	^[a-zA-Z]+$	^[A-Z]{2}[a-zA-Z0-9]+$	^[0-9]{4,5}$	ChooseUniformly	ChooseProportionally
phonenumber	^[0-9]{10}$	^\+?[\d\s()-]{7,20}$		StringPrior	ChooseUniformly
personname	^[A-Za-z]+_[A-Z][a-z]+$	^[a-zA-Z0-9-]+$	^[A-Za-z]+$	StringPrior	ChooseProportionally
boolean	^(yes|no)$	^(1|0)$			
text	^[\u4e00-\u9fffa-zA-Z\s]+$				

(iii) *Interoperating with other cleaning systems.* The library can also serve external systems; format patterns can be exported as pattern-based constraints for rule-driven cleaning systems, while distribution models can be used as priors in PPL-based systems such as PClean. In this way, the library grows with new datasets and becomes a shared repository of reusable priors.

Overall, when a new attribute is classified, the system retrieves candidate patterns from the library and suggests the most probable ones, shifting the user's role from pattern creator to validator. This interaction mode significantly reduces user effort and technical barriers while maintaining the quality of critical constraints, thereby enhancing overall data cleaning performance.

VI. BAYESIAN INFERENCE WITH COMPENSATORY SCORE

Upon constructing the BN and automatically generating priors (i.e., UCs), we design an inference method that incorporates a compensatory strategy to identify the most probable candidate value for each attribute of a tuple T. Traditional BN inference typically performs repair via MAP estimation in an iterative manner. However, such a method is often ineffective as most real-world datasets are unclean and the BNs constructed from them lack accuracy. Consequently, incorrect repairs may propagate errors to subsequent steps, a problem we refer to as error amplification.

To address this issue, recall that for each tuple $T \in D$ and attribute $A_j \in \mathcal{A}$, the logarithmic form of the probability of a candidate value $c \in dom(A_j)$, i.e., $\log \mathbf{Pr}[c]$, can be partitioned into two terms, $\log \mathbf{Pr}[c|t]$ and $\log \mathbf{Pr}[t] - \log \mathbf{Pr}[t|c]$, where t denotes the remaining attribute values of T excluding A_j (see Eq. (3)). The first term $\log \mathbf{Pr}[c|t]$ can be computed through BN inference via CPTs. To minimize the errors produced by $\log \mathbf{Pr}[c|t]$, we regard the second term as a compensatory score:

$$\mathsf{Score}_{\mathsf{comp}} = \log \mathbf{Pr}[t] - \log \mathbf{Pr}[t|c]. \tag{6}$$

Since $\mathbf{Pr}[t]$ is a constant once the BN is fixed, the discriminative power of $\mathsf{Score}_{\mathsf{comp}}$ is determined solely by $\mathbf{Pr}[t|c]$. Intuitively, if $\mathbf{Pr}[t|c]$ is very large, then c tends to be overfitted to the specific tuple t, often reflecting a spurious or isolated error. In this case, $\mathsf{Score}_{\mathsf{comp}}$ becomes smaller, thereby penalizing such candidates. Conversely, if $\mathbf{Pr}[t|c]$ is relatively small, c aligns more closely with the general data distribution rather than a single tuple, which results in a larger $\mathsf{Score}_{\mathsf{comp}}$ and favors the selection of c.

Example 4: *Consider the BN constructed from the relational data in Table I, where we aim to impute the missing value for the attribute* $\mathsf{Department}$ *of* t_6. *Owing to the unclean data, we derive* $\mathbf{Pr}[\text{"}\textit{400 nprthwood dr}\text{"}|t_6] = 0.54$ *and* $\mathbf{Pr}[\text{"}\textit{400 northwood dr}\text{"}|t_6] = 0.51$. *If only* $\log \mathbf{Pr}[c|t]$ *is considered, the incorrect value "400 nprthwood dr" would be imputed. Incorporating* $\mathsf{Score}_{\mathsf{comp}}$ *helps to avoid such cases.*

However, computing $\mathsf{Score}_{\mathsf{comp}}$ directly is intractable, as its discriminative power depends on the unknown true value c^*. Since c^* is precisely what we aim to infer, relying on it creates a circular dependency between c and t. Moreover, $\mathsf{Score}_{\mathsf{comp}}$ suffers from a critical limitation: when an erroneous value c' appears only once in the dataset, we obtain $\mathbf{Pr}[t|c'] = 1$, which yields $-\log \mathbf{Pr}[t|c'] = 0$. In this case, $\mathsf{Score}_{\mathsf{comp}}$ degenerates to the constant $\log \mathbf{Pr}[t]$, independent of c', and thus fails to penalize errors effectively. To address these issues, we approximate the compensatory score using the correlation between c and t:

$$\mathsf{Score}_{\mathsf{comp}}(c, t) \approx \mathsf{Score}_{\mathsf{corr}}(c, t). \tag{7}$$

This approximation is intuitive. If a candidate value $c \in \mathsf{dom}(A_i)$ of a tuple T is indeed the correct value c^*, it tends to co-occur frequently with the other attribute values of tuple T, i.e., t, thereby exhibiting strong correlation. Thus, modeling the correlation between c and t, i.e., $\mathsf{Score}_{\mathsf{corr}}$, aligns with the same objective as $\mathsf{Score}_{\mathsf{comp}}$.

Inspired by Bayeswipe [26], we adopt a straightforward yet effective idea to compute $\mathsf{Score}_{\mathsf{corr}}$ by counting the co-occurrences of (c, t), i.e., $\mathsf{Score}_{\mathsf{corr}} = \frac{\mathsf{count}(c,t)}{|D|}$. Since this scoring suffers from sparsity and tends to be inaccurate for infrequent observations, to mitigate this, we compute the correlation between c and each individual attribute value in t, and then accumulate them. If more values in t have a high correlation with c, then $\mathsf{Score}_{\mathsf{corr}}(c, t)$ is likely to be more significant. Since $c = T_i[A_j]$ and $t = T_i[A_1, \ldots, A_{j-1}, A_{j+1}, \ldots, A_m]$, we formally define $\mathsf{Score}_{\mathsf{corr}}$ as follows:

$$\mathsf{Score}_{\mathsf{corr}}(c, t, A_j) = \sum_{A_k \in \mathcal{A} \setminus \{A_j\} \wedge e = t[A_k]} \mathsf{corr}(c, e, A_j, A_k), \tag{8}$$

where $\mathsf{corr}(c, e, A_j, A_k)$ denotes the correlation between value c of attribute A_j and value e of attribute A_k. Due to the presence of noise, instead of computing $\mathsf{corr}(c, e, A_j, A_k)$ by directly counting the co-occurrence of (c, e), i.e., $\mathsf{corr}(c, e, A_j, A_k) = \frac{\mathsf{count}(c,e,A_j,A_k)}{|D|}$, we incorporate user-provided UCs and introduce the concept of confidence (conf). Specifically, we assign each tuple T_i a weight (i.e., confidence) $\mathsf{conf}(T_i)$, where a higher value indicates that T_i is more likely to be accurate, and thus the correlations among its values are more reliable. Formally, we define $\mathsf{conf}(T_i)$ as follows:

$$\mathsf{conf}(T_i) = \max\Big\{0, \frac{\sum\limits_{e\in T_i} \mathbf{1}_{\{UC(e)=1\}} - \lambda\cdot\sum\limits_{e\in T_i} \mathbf{1}_{\{UC(e)=0\}}}{|T_i|}\Big\}, \quad (9)$$

where $UC(\cdot)$ is a function that returns 1 if a candidate attribute value satisfies the user-specified constraints and 0 otherwise, $\lambda \geq 0$ controls the strength of the penalty for violations, and $\mathbf{1}$ is the indicator function. Based on this, we define $\mathsf{corr}(c, e, A_j, A_k)$ as

$$\mathsf{corr}(c, e, A_j, A_k) = \frac{\sum_{T\in\Omega} (\mathbf{1}_{\mathsf{conf}(T)\geq\tau} - \beta\cdot\mathbf{1}_{\mathsf{conf}(T)<\tau})}{|D|},$$
$$\Omega = \{T' | T' \in D \wedge (c, e) = (T'[A_j], T'[A_k])\}, \quad (10)$$

where $\tau \in [0, 1]$ is the confidence threshold and $\beta \geq 0$ penalizes unreliable tuples. Specifically, if the confidence of a tuple $T \in D$ satisfies $\mathsf{conf}(T) \geq \tau$, T is regarded as reliable and the combinations of its attribute values (i.e., $(T[A_j], T[A_k])$) contribute positively to corr. Otherwise, T is considered unreliable, and a penalty β is imposed to reduce its influence on corr.

Since UCs are incorporated in Eq. (9), more violations of UCs will reduce $\mathsf{Score}_{\mathsf{corr}}$. Note that the inference process is essentially a competition among candidate values in the domain, where only relative ranking matters rather than absolute scores. Therefore, $\mathsf{Score}_{\mathsf{corr}}$ need not equal $\mathsf{Score}_{\mathsf{comp}}$.

To efficiently compute $\mathsf{Score}_{\mathsf{corr}}$ for all attributes, we design a compensatory score computation algorithm with time complexity $O(nm^2)$ (Algorithm 2). It scans each tuple in D, for every attribute A_i, computes its correlation with other attribute $A_j \in \mathcal{A} \setminus \{A_i\}$ (lines 2–16). Specifically, for each tuple $T \in D$ (line 2), it computes $\mathsf{conf}(T)$ using Eq. (9) and updates correlations for each attribute pairs (A_i, A_j). If $\mathsf{conf}(T) \geq \tau$, it updates corr by accumulating the counts of $(T[A_i], T[A_j])$; otherwise, it imposes a penalty β to decrease corr. Finally, the correlation values of all attribute-value pairs in D are computed and stored, enabling $\mathsf{Score}_{\mathsf{corr}}$ to be readily computed using Eq. (8).

Example 5: *Continuing with Example 4, we further incorporate the compensatory score with parameters $\lambda = 0.25$, $\beta = 2$, and $\tau = 0.75$. Suppose $UC(\cdot)$ is a spell checker on* Department *that returns 1 if the spelling is correct and 0 otherwise. For the erroneous candidate c = "400 nprthwood dr" (from t_5), the UC is violated, yielding $\mathsf{conf}(t_5) = 0 < \tau$; consequently, t_5 contributes a penalty $-\beta$ to the $\mathsf{corr}(c, t_6[A_k], \mathsf{Department}, A_k)$ in Eq.* (10)*; via Eq.* (8)*, this lower $\mathsf{Score}_{\mathsf{corr}}(c, t_6, \mathsf{Department})$. In contrast, the candidate "400 northwood dr" (from t_4) satisfies the UC, with $\mathsf{conf}(t_4) = 1 \geq \tau$, providing reliable positive correlation evidence. Under this UC configuration, combining BN inference with the compensatory score updates the probabilities to* $\mathbf{Pr}$[*"400 northwood dr"*$|t_6] = 0.64$ *and* $\mathbf{Pr}$[*"400 nprthwood dr"*$|t_6] = 0.23$*. Unlike Example 4 where the BN alone favored the erroneous value, the proposed method correctly imputes the* Department *of t_6 as "400 northwood dr".*

Remarks. Our idea is inspired by BayesWipe [26], but the realization is fundamentally different. Instead of optimizing a global loss, we compute a correlation-based score $\mathsf{Score}_{\mathsf{corr}}$ that aggregates attribute-value co-occurrence evidence, weighted by tuple confidence. We implement this with a co-occurrence dictionary and accumulate evidence between the candidate value and other observed values in the tuple; the overall computation is $O(nm^2)$.

Algorithm 2: Compensatory Score Computation

```
Input  : An observed dataset $D$, user constraints $UC(\cdot)$, a
         confidence threshold $\tau$, and a penalty parameter $\beta$.
Output : A set of compensatory scores corr.
1  corr ← {};
2  foreach $T \in D$ do
3      $p \leftarrow \emptyset$;
4      Compute conf($T$) using Eq. (9);
5      foreach $A_i \in \mathcal{A}$ do
6          $c \leftarrow T[A_i]$;
7          foreach $A_j \in \mathcal{A} \setminus \{A_i\}$ do
8              $v \leftarrow T[A_j]$;
9              if conf($T$) $\geq \tau$ then
10                 corr[$\langle c, v, A_i, A_j\rangle$] ← corr[$\langle c, v, A_i, A_j\rangle$]+1;
11             else
12                 corr[$\langle c, v, A_i, A_j\rangle$] ← corr[$\langle c, v, A_i, A_j\rangle$]$-\beta$;
13 return corr;
```

VII. Bayesian Inference Optimization

When the dataset is large, the number of variables in the BN increases, making BN inference computationally prohibitive. To address this issue, we design a partition inference mechanism that performs inference locally, restricting interacting to nodes within one hop of the Markov blanket. Furthermore, we introduce two pruning strategies to reduce unnecessary computation: (i) a tuple pruning strategy, which identifies and repairs only the cells likely to be erroneous while skipping those that are largely clean, and (ii) a domain pruning strategy, which eliminates attribute values that are clearly invalid candidates for repair.

A. Partitioned Bayesian Inference

Bayesian inference methods can be categorized into exact inference (e.g., variable elimination and belief propagation), and approximate inference (e.g., Gibbs sampling). Unlike existing techniques that focus on the global distribution of BN, we leverage the Markov property to localize inference. Specifically, we treat the inferred node A_j as a state in BN while disregarding nodes that are not directly connected to A_j. This essentially transforms a BN into a set of sub-networks, with all other nodes regarded as observations. During inference on each node, only the nodes and edges within its sub-network are involved in the computation.

BN partitioning. The variable elimination strategy for exact inference necessitates the inclusion of all precursor states of A_j in the BN topology. However, the states of precursor nodes might have been altered in previous repairing steps, meaning the prior of A_j may no longer match its original distribution. Consequently, inferring from the first node can incur higher costs for every A_j, and if the modified states are erroneous, these errors may propagate into the inference of A_j. To avoid this erroneous propagation and accelerate inference, we partition BN into l sub-networks (here, we use a set of attributes to denote a sub-network): $\mathsf{BN_{sets}} = \{\mathcal{A}^{(1)}_{\text{joint}}, \mathcal{A}^{(2)}_{\text{joint}}, \ldots, \mathcal{A}^{(l)}_{\text{joint}}\}$, following the Markov blanket. Here, $\mathcal{A}^{(i)}_{\text{joint}}$ $(1 \le i \le l)$ denotes the i-th sub-network containing the inferred node A_j, alongside its one-hop parent nodes $\mathcal{A}^{(i)}_{\text{parent}}$ and child nodes $\mathcal{A}^{(i)}_{\text{child}}$. Formally,

$$\mathcal{A}^{(i)}_{\text{joint}} = \mathcal{A}^{(i)}_{\text{parent}} \cup \{A_j\} \cup \mathcal{A}^{(i)}_{\text{child}}. \tag{11}$$

Multiple sub-networks might intersect at a node A_k, but $A_k \in \mathcal{A}^{(i)}_{\text{joint}}$ does not affect other sub-networks.

Bayesian inference. In the partitioned BN, nodes can be categorized into two types: isolated nodes $\mathcal{A}_{\text{iso}}$, which are not connected to other nodes, and joint nodes $\mathcal{A}_{\text{joint}} \in \mathsf{BN_{sets}}$, which connect to other nodes. For each isolated node, the CPT is modeled as a uniform distribution, assuming candidate values are uniformly distributed in the domain. For each joint node, all observations are known; thus, the probability distribution of candidate values is directly obtained from the CPT. Specifically, the inferred node A_j interacts only with its connected nodes $\mathcal{A}_{\text{connected}}$, including parent nodes $\mathcal{A}_{\text{parent}}$ and child nodes $\mathcal{A}_{\text{child}}$, with CPTs given by $\mathbf{Pr}[\mathcal{A}_j|\mathcal{A}_{\text{parent}}]$ and $\mathbf{Pr}[\mathcal{A}_{\text{child}}|A_j]$. In accordance with the properties of BN, we treat A_j as known, denoting every value in the domain as a candidate. As such, $\mathcal{A}_{\text{parent}}$ and $\mathcal{A}_{\text{child}}$ are independent. Formally,

$$\mathbf{Pr}[A_j|\mathcal{A}_{\text{connected}}] = \mathbf{Pr}[A_j|\mathcal{A}_{\text{parent}}] \cdot \mathbf{Pr}[\mathcal{A}_{\text{child}}|A_j]. \tag{12}$$

B. Pruning Strategies

Data cleaning on D is performed with values drawn from the domain of their corresponding attributes, and the inference cost is bounded by both the number of cells and candidate values. To reduce this cost, we introduce tuple pruning and domain pruning strategies to prune cells that do not need inspection and candidate values that cannot be correct answers, respectively.

Tuple pruning. Given a tuple T and an attribute A_i, we design a filtering mechanism to determine whether $T[A_i]$ should be inferred. The intuition is that if $T[A_i]$ co-occurs with other values of T more frequently, indicating a stronger correlation, then $T[A_i]$ is more likely to be correct and $T[A_i]$ has a lower priority to be inferred. We define a function $\mathsf{Filter}(T, A_i)$ with a threshold τ_{clean} to determine whether $T[A_i]$ needs inference:

$$\mathsf{Filter}(T, A_i) = \frac{1}{m-1} \sum_{A_j \in \mathcal{A} \setminus \{A_i\}} \frac{\mathsf{count}(T[A_i], T[A_j])}{\mathsf{count}(T[A_j])}. \tag{13}$$

During cleaning, we first compute $\mathsf{Filter}(T, A_i)$. If the result is not less than τ_{clean}, we treat $T[A_i]$ as relatively reliable and skip BN inference in the current iteration; otherwise, $T[A_i]$ is deemed obscure and requires repair. Note that tuple pruning differs from standard error detection, while both can identify erroneous values, tuple pruning prioritizes cells with the fewest conflicts.

Domain pruning. We treat each sub-network as an independent semantic space and perform domain pruning from a semantic perspective, akin to the cloze test task in natural language processing.

In this semantic space, we take the dividing variable as the fill-in-the-blank variable, with all variables excluding the dividing variable acting as the context, and compute each answer under these semantics. For every value $v \in \mathsf{dom}(A_i)$, we assign a weight using TF-IDF. Formally,

$$\begin{aligned} \mathsf{score}(v) &= \mathsf{TF}(v, \mathsf{context}) \cdot \mathsf{IDF}(v, D) \\ &= \mathsf{context}(v) \cdot \log\left(\frac{|D|}{1 + \mathsf{count}(v, D)}\right), \end{aligned} \tag{14}$$

where $\mathsf{context}(v)$ denotes the number of sub-networks filled with v. These semantics treat different sub-networks as distinct contexts. The more frequently v appears in a specific semantic and the less frequently v appears in other semantics, the more likely v is to become the ground truth in that semantic. For every non-semantic $v \in \mathsf{dom}(A_i)$, its score is significantly lower, possibly even zero. Each sub-network only computes a few candidates likely to be the ground truth during inference.

VIII. Generalizing the Framework: Automated PPL Synthesis

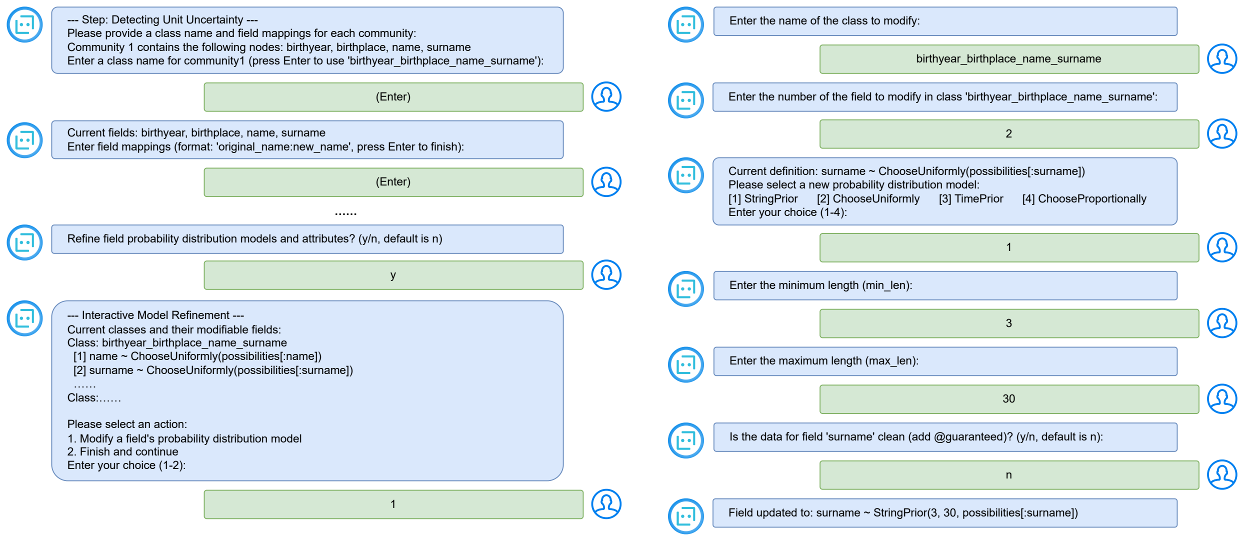
The preceding sections have detailed the core data repair framework of BClean+, where the automated prior generation module leverages learned data characteristics to construct and maintain a reusable template library. This foundation naturally extends to the automated synthesis of probabilistic programming language (PPL) code, directly benefiting PPL-based systems such as PClean, which rely heavily on domain experts to manually write complex PPL programs. By enabling the automated synthesis of high-quality PPL code, BClean+ significantly reduces the barrier to adoption of such systems.

A complete PPL program typically comprises two core components: a *data relationship model* and a *probabilistic distribution model*. BClean+ automates the construction of both as follows.

Data Relationship Model. The data relationship model encodes dependencies among attributes. BClean+ derives this by utilizing the Bayesian network construction and community detection (see Section IV). The BN captures global statistical dependencies, while community detection partitions attributes into distinct clusters. Each cluster represents a semantically related group of attributes and can be translated into a PPL @class definition (Fig. 7). This design ensures that attributes with strong correlations are co-located in the same class, preserving both local context and global structure in the synthesized program.

Probabilistic Distribution Model. The probabilistic distribution model specifies how attribute values are generated within each class. To this end, BClean+ integrates the reusable library containing candidate distribution models drawn from PClean, which serve as priors for latent "clean" data. These models provide the quantitative basis to distinguish probable errors from rare-but-correct values, thereby enabling inference of the most likely repairs. Representative models used in PPL are as follows:

- StringPrior, for textual data, assessing string likelihoods based on character frequencies and transition probabilities.
- ChooseUniformly, which applies a uniform probability distribution over a fixed discrete set of options, used when no prior frequency information is available.
- ChooseProportionally, which learns non-uniform probabilities from the observed data to capture real-world skew.
- Specialized models, such as TimePrior for parsing and repairing time-of-day data, and TransformedGaussian for



```
--- Step: Detecting Unit Uncertainty ---
Please provide a class name and field mappings for each community:
Community 1 contains the following nodes: birthyear, birthplace, name, surname
Enter a class name for community1 (press Enter to use 'birthyear_birthplace_name_surname'):
                                        (Enter)
Current fields: birthyear, birthplace, name, surname
Enter field mappings (format: 'original_name:new_name', press Enter to finish):
                                        (Enter)
......
Refine field probability distribution models and attributes? (y/n, default is n)
                                        y
--- Interactive Model Refinement ---
Current classes and their modifiable fields:
Class: birthyear_birthplace_name_surname
  [1] name ~ ChooseUniformly(possibilities[:name])
  [2] surname ~ ChooseUniformly(possibilities[:surname])
  ......
Class:......

Please select an action:
1. Modify a field's probability distribution model
2. Finish and continue
Enter your choice (1-2):
                                        1
Enter the name of the class to modify:
                                        birthyear_birthplace_name_surname
Enter the number of the field to modify in class 'birthyear_birthplace_name_surname':
                                        2
Current definition: surname ~ ChooseUniformly(possibilities[:surname])
Please select a new probability distribution model:
[1] StringPrior     [2] ChooseUniformly     [3] TimePrior     [4] ChooseProportionally
Enter your choice (1-4):
                                        1
Enter the minimum length (min_len):
                                        3
Enter the maximum length (max_len):
                                        30
Is the data for field 'surname' clean (add @guaranteed)? (y/n, default is n):
                                        n
Field updated to: surname ~ StringPrior(3, 30, possibilities[:surname])
```

Fig. 6. An example of lightweight interactive session for user validation and adjustment of automatically generated PPL code.

```
PClean.@model GeneratedModel begin
   @class City begin
      city ~ ChooseUniformly(possibilities[:City])
      countyname ~ ChooseUniformly(possibilities[:CountyName])
   end;
   @class Measure begin
      measurecode ~ StringPrior(3, 30, possibilities[:MeasureCode])
      ......
   end;
   ......
   @class Hospital begin
      city_~City
      hospitalname ~ ChooseUniformly(possibilities[:HospitalName])
      @learned emergencyservice_dist::ProportionsParameter
      emergencyservice ~ ChooseProportionally(possibilities[:EmergencyService], emergencyservice_dist)
      ......
   end;
```

Fig. 7. An example of the final synthesized PPL code.

continuous numerical attributes that may have undergone systematic, invertible transformations (e.g., unit errors).

The selection and configuration of distribution models are automated, using a heuristic approach that relies on inherent column characteristics such as value distribution, cardinality, and character composition. For example, a column with low cardinality but skewed distribution (dominated by a few values) is mapped to the ChooseProportionally model, whereas natural-language fields with regular value lengths and alphabetic patterns (e.g., names and addresses) are assigned to the StringPrior model. Once the most suitable distribution is inferred, BClean+ selects the corresponding model from the library and performs fine-grained parameter tuning using the data samples (e.g., estimating minimum/maximum length for textual data). This process yields an optimized data distribution model tailored to each specific attribute.

After deriving both data relationships and distribution models, BClean+ compiles them into a draft PPL program. A lightweight interactive refinement step (Fig. 6) then allows users to confirm or adjust the automatically generated code, ensuring that the final PPL programs balance automation with domain expertise. An example of the synthesized PPL code is shown in Fig. 7.

IX. Experiments

A. Experiment Setup

Datasets. We used six benchmark datasets of varying sizes and error types as summarized in Table III. (1) **Hospital** (from [5], [45]) contains hospital details (e.g., HospitalName and Address) with ∼5% errors and full-cell ground truth, featuring substantial duplication and strong attribute dependencies. (2) **Flights** (used in [5] and collected from [46]) records airline departure and arrival times from multiple websites, with partial ground truth. (3) **Soccer** contains football players profiles (clean version from [47]) with ∼5% errors. (4) **Beers** (used in [3], [6]) contains two numerical attributes (ounces, abv) and provides both clean and dirty versions from the same source. (5) **Inpatient** contains inpatient profiles collected from CMS [48]. (6) **Facilities** contains medical enterprise information from CMS [48].

Error Injection. In line with the error injection approach utilized in the benchmarks (e.g., Hospital) by Raha+Baran [3], [6] and HoloClean [5], by default, we categorized errors into three types: typos (T), missing values (M), inconsistencies (I). For T, we modified the original value by randomly adding, deleting, or replacing a character. For M, we randomly replaced a value with NULL. For I, we interchanged two values from the domains of two columns or a specific column. Their frequencies do not exhibit a significant difference, e.g., in the Inpatient dataset, we observed 1210, 1800, and 1480 instances of T, M, and I errors, respectively. In addition to these default error settings, we also evaluated two types of errors separately: synthetic errors (S), which include typos, formatting issues, and violations of functional dependencies, following the synthesis method in Baran [6]; swapping value errors (SV), which were generated by swapping values within the same attribute, i.e., the same domain.

Methods. We compared the following methods. (1) BClean: the basic version without the optimizations in Section VII; its user constraints (UCs) are manually defined. (2) BClean+: a variant of BClean with automatically synthesized UCs via our automated knowledge acquisition framework. (3) BClean$_{\text{-UC}}$: a variant of BClean without UCs. (4) BClean$_{\text{PI}}$: a variant of BClean with Partition Inference (PI) to reduce unnecessary computation by partitioning the network. (5) BClean+$_{\text{PI}}$: a variant of BClean+ combined with PI. (6) BClean$_{\text{PIP}}$: a variant of BClean that enables PIP (PI and Pruning based on sub-network semantics). (7) BClean+$_{\text{PIP}}$: a variant of BClean+ with PIP optimizations. (8) BClean+$_{\text{PI-nosemantic}}$: an ablation

TABLE III
DATASET STATISTICS WITH ERROR TYPES: TYPOS (T), MISSING VALUES (M), INCONSISTENCY (I), SYNTHETIC (S), AND SWAPPING VALUE ERRORS (SV)

Dataset	Size (#rows, #columns, #cells)	Noise rate	Error types	BN(#nodes, #edges)	#UCs (BClean+)	#DCs (HoloClean)	#lines of PPL (PClean)	#labels of tuples (Raha+Baran)
Hospital	(1000, 15, 15k)	∼5%	T, M, I	(15, 12)	15	13	51	20+20
Flights	(2376, 6, 14k)	∼30%	T, M	(5, 4)	5	4	36	20+20
Soccer	(200000, 10, 2M)	∼1%	T, M, I	(10, 14)	10	4	37	20+20
Beers	(2410, 11, 27k)	∼13%	T, M, I	(6, 4)	6	6	26	20+20
Inpatient	(4017, 11, 44k)	∼10%	T, M, I, SV	(11, 10)	11	3	54	20+20
Facilities	(7992, 11, 88k)	∼5%	T, M, I, SV	(10, 8)	10	8	48	20+20

TABLE IV
LIST OF EXPERT-AUTHORED USER CONSTRAINTS (UCS) IN BCLEAN

Dataset	Dataset-specific regex UCs
Hospital	∧([1-9][0-9]{4, 4})[ProvideNumber, ZipCode]; ∧([1-9][0-9]{9, 9})[PhoneNumber];
Flights	([1-9]:[0-5][0-9][s][ap].[m].—1[0-2]:[0-5][0-9][s][ap].[m].—0[1-9]:[0-5][0-9][s][ap].[m].)[sched_dep_time, act_dep_time, sched_arr_time, act_arr_time]
Soccer	([1][9][6-9][0-9])[birthyear]; ([2][0][0-9][0-9])[season]
Beers	\d + \. \ d + |(\d+)[ounces, abv]
Inpatient	N/A
Facilities	N/A

All datasets additionally enforce generic min/max length constraints on textual attributes and non-null constraints on all attributes.
N/A indicates that no dataset-specific regex UCs are manually provided for that dataset in BClean.

variant of BClean+$_{\text{Pl}}$ that bypasses semantic type identification. (9) PClean [4]: a PPL-based Bayesian cleaning system that requires manual authoring of data relationships and distributions. (10) PClean-auto: a variant of PClean utilizes its inference engine with PPL program automatically generated by our framework. (11) Raha+Baran [3], [6]: an ML-based cleaning pipeline including Raha for detection and Baran for correction; we use its default settings (e.g., labeling budget and models). (12) HoloClean [5]: a semi-supervised inference system that compiles integrity rules, matching dependencies, and statistical signals into features in a factor Graph; we use signals provided by dataset owners or collected by ourselves. (13) Garf [49]: a rule-based method using sequence generative adversarial networks (GANs); we used the default settings as provided in the paper. Since the DCs of HoloClean [5] and the labels used for Raha+Baran [3], [6] are not publicly available, we configured them by ourselves. This may result in different experimental results from the numbers reported in the original papers.

Prior Knowledge. Table III reports the statistical numbers of user inputs for all the competitors. For the baseline methods, prior knowledge was manually constructed by data-quality experts for each dataset. This involved writing DCs for HoloClean, designing complete PPL programs in Julia for PClean, labeling 20 tuples for Raha, and providing 20 corrected tuples for Baran.

For our own methods, BClean serves as a semi-automated baseline where experts manually craft 12 regular expressions (Table IV) based on the attribute formats. In contrast, BClean+ eliminates this manual effort by automatically synthesizing UCs; it infers the semantic type of each attribute and, by default, selects from the reusable template library (Section V) the pattern with the highest syntactic match rate. Users may optionally select an alternative pattern, but no mandatory user input is required. In addition, PClean-auto leverages our framework to automatically synthesize the core Julia scripts for PPL model definition and execution (Section VIII), fully replacing the expert-driven programming effort required by PClean. To ensure fairness, automated pattern generation excludes specific values in regular expressions to prevent ground-truth leakage.

Metrics. We used the following metrics to measure accuracy. (1) **Precision**: The fraction of the correctly repaired errors over the total number of modified cells. (2) **Recall**: The fraction of the correctly repaired errors over the total number of errors labeled with ground truth. (3) **F1-score**: The harmonic mean of precision and recall. To measure time and user cost, we collected the following metrics. (1) **User time**: The total time spent by users to inject prior knowledge into each baseline method. Initially, two experts were trained to understand the inputs and outputs of each baseline, including DCs of HoloClean, UCs of BClean, PPLs of PClean, and labeling of Raha+Baran. User Time represents the average time these experts spent to inject prior knowledge into a novel, unseen dataset. This metric assesses the usability of the system for trained users. We excluded training time as it varies across individuals. (2) **Execution time**: The total runtime of the system modules to complete the data cleaning task, given a dataset and the users' injected prior knowledge.
Parameters. For BClean and BClean+, we set $\lambda = 1$, $\beta = 2$, and $\tau = 0.5$. These parameters reflect the principle that tuples with lower confidence in their relationships, as indicated by UC violations, are considered less trustworthy. Specifically, any tuple with a confidence score below τ is penalized with β and λ (see Section VI). For other baselines, we adopted the default settings in PClean and HoloClean. Following the suggestions in Raha and Baran, We set the number of labeled tuples to $20 + 20$.
Environment. We conducted the experiments on a single machine with 128GB RAM and 32 Intel(R) Core(TM) Xeon(R) 6326 CPUs at 2.90GHz. We ran each experiments 5 times and reported the average performance for both effectiveness and efficiency.

B. *Effectiveness and Efficiency*

1) *Data Cleaning Quality*

We first evaluated the performance of BClean+, our extended framework, together with its predecessor BClean, several ablation variants, and representative baselines. As shown in Table V, the original BClean performs particularly well on datasets with adequate relational information and low error rates, such as Hospital and Soccer, where its F1-scores are significantly higher than those of HoloClean, Raha+Baran, and Garf. These findings establish a robust performance benchmark for assessing the improvements brought by BClean+.

Turning to BClean+, the extended framework demonstrates strong and competitive performance. On Hospital (0.982 F1) and

TABLE V
PRECISION (P), RECALL (R), AND F1-SCORE (F1) OF DATA CLEANING METHODS

Method	Hospital			Flights			Soccer			Beers			Inpatient			Facilities		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
BClean$_{\text{-UC}}$	**1.000**	0.935	0.966	0.807	0.729	0.766	0.927	0.982	**0.954**	0.880	0.065	0.121	0.934	**0.883**	**0.908**	0.810	**0.805**	0.807
BClean	0.998	0.956	0.976	0.852	0.816	0.834	**0.928**	0.979	0.952	0.916	0.887	0.901	0.909	0.845	0.876	–	–	–
BClean$_{\text{PI}}$	**1.000**	0.960	0.980	0.831	0.780	0.805	0.919	0.986	0.951	0.948	0.949	0.949	0.934	**0.883**	**0.908**	0.810	**0.805**	0.807
BClean$_{\text{PIP}}$	0.997	0.903	0.948	0.830	0.784	0.807	0.845	0.931	0.885	0.948	0.882	0.914	0.929	0.791	0.855	0.753	0.730	0.741
BClean+	**1.000**	**0.965**	**0.982**	0.852	0.816	0.834	0.927	0.982	**0.954**	0.917	0.887	0.901	0.815	0.817	0.816	0.948	0.792	0.863
BClean+$_{\text{PI}}$	**1.000**	**0.965**	**0.982**	0.830	0.780	0.804	0.921	**0.990**	**0.954**	0.949	**0.954**	**0.951**	0.816	0.824	0.820	0.950	0.799	**0.868**
BClean+$_{\text{PIP}}$	**1.000**	0.943	0.971	0.830	0.780	0.804	0.836	0.923	0.878	0.949	0.882	0.914	0.828	0.728	0.774	0.921	0.747	0.825
PClean	**1.000**	0.927	0.962	0.907	**0.884**	**0.895**	0.184†	0.672†	0.289†	0.028†	0.028†	0.028†	0.576†	0.460†	0.512†	–†	–†	–†
HoloClean	**1.000**	0.456	0.626	0.742	0.352	0.477	–	–	–	**1.000**	0.024	0.047	0.966	0.219	0.357	**1.000**	0.612	0.759
Raha+Baran	0.971	0.585	0.730	0.829	0.650	0.729	0.768	0.103	0.182	0.873	0.872	0.873	0.643	0.442	0.524	0.499	0.309	0.382
Garf	**1.000**	0.556	0.715	**0.968**	0.012	0.024	0.667	0.534	0.583	0.973	0.011	0.021	**0.971**	0.091	0.166	0.963	0.281	0.435

† We invite people familiar with PClean to author the data models in PPL.
– Out-of-memory, out-of-runtime (24h), or no repairs.

TABLE VI
PRECISION (P), RECALL (R), AND F1-SCORE (F1) ON SAMPLED SOCCER

	BClean+	BClean	HoloClean	PClean	Raha+Baran
P	0.859	0.345	0.919	0.150	0.523
R	0.992	0.931	0.551	0.665	0.133
F1	0.921	0.504	0.689	0.244	0.212

Soccer (0.954 F1), it performs on par with the original BClean, despite relying on automatically generated constraints rather than manually crafted UCs. More notably, BClean+ successfully handles Facilities (0.863 F1), a dataset where BClean failed to run, and achieves robust results on Beers (0.901 F1) and Inpatient (0.816 F1), while remaining competitive performance on the high-error-rate Flights (0.834 F1). These confirm the high quality of its automated constraints and enhanced robustness.

The efficiency-oriented variants of BClean+ further highlight the accuracy-efficiency trade-off. For example, BClean+$_{\text{PI}}$, which incorporates partition inference, shows stable performance and even improves the F1-score on Beers to 0.951. In contrast, BClean+$_{\text{PIP}}$, which enables both partitioning and pruning for maximum efficiency, achieves significant speedups (Table VIII) at the cost of slight to moderate accuracy drops on datasets such as Soccer (0.878 F1) and Inpatient (0.774 F1). These verify the effectiveness of the proposed optimization strategies.

Overall, BClean+ consistently outperforms the four baselines (PClean, HoloClean, Raha+Baran, and Garf), achieving an average F1-score of 0.89 (up to 0.98), which represents an average improvement of 0.42 (up to 0.57). Due to the out-of-memory issue encountered with HoloClean, we also randomly sampled 50,000 tuples from the Soccer dataset for evaluation. As shown in Table VI, BClean+ still outperforms all methods, e.g., it achieves an F1-score of 0.921, which is 0.512 higher than the baselines on average.

By contrast, PClean achieves the highest performance on Flights, however, this largely depends on expert-crafted domain knowledge. On Soccer, users find it challenging to specify data distributions accurately and could only estimate them from observation, leading to unsatisfactory results. These highlight the limitations of methods that are heavily reliant on user expertise, while also emphasizing the practicality and robustness of BClean+.

2) *Error Analysis*

Next, we studied how error types (T, M, and I, as specified in the experiment setup) and their ratios affect the performance. The distributions of errors across four datasets are shown in Fig. 8(a).

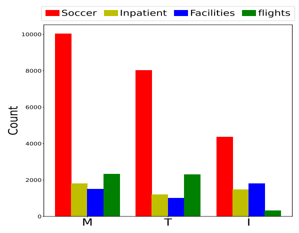
Varying error types. To assess the robustness of the competitors, we evaluated the recall for each type of error on the Soccer, Inpatient, and Facilities datasets (Table VII). We did not report precision here, as it is challenging to determine which type of error a corrected value from the baselines originally belongs to. As shown in Table VII, both BClean and BClean+ consistently outperform, or at least remain competitive with, other methods across all error types. On average, BClean+ achieves recalls of 0.89, 0.93, and 0.94 for T, M, and I, respectively, compared with 0.84, 0.97, and 0.89 for BClean. Moreover, BClean+ outperforms other methods with average improvements of 0.29, 0.22, and 0.34 across the three error types. This shows that the automated framework enhances cleaning effectiveness without sacrificing the robustness of its predecessor.

TABLE VII
RECALL FOR DIFFERENT TYPES OF ERRORS (T, M, AND I)

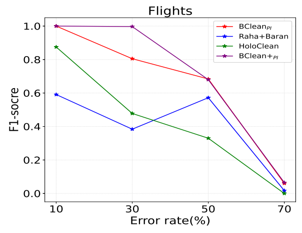
Method	Soccer			Inpatient			Facilities		
	T	**M**	**I**	**T**	**M**	**I**	**T**	**M**	**I**
BClean$_{\text{PI}}$	0.997	1.000	0.990	0.840	1.000	0.843	0.683	0.900	0.837
BClean+$_{\text{PI}}$	0.987	0.988	0.990	0.851	0.906	0.957	0.833	0.898	0.872
PClean	1.000	0.568	0.953	0.323	0.760	0.477	0.0	0.0	0.0
HoloClean	0.749	1.000	0.923	0.954	0.612	0.949	0.804	1.000	0.851
Raha+Baran	0.047	0.244	0.018	0.491	0.890	0.109	0.295	0.501	0.213

Varying error ratio. As depicted in Figs. 8(b) – 8(d), we evaluated the competing methods by varying the error ratio from 10% to 70%, following the evaluation in [3]. As expected, all methods exhibit decreasing F1-scores as the error ratio increases. Nevertheless, BClean+ demonstrates greater robustness and consistently outperforms the others. On average, its F1-score is 0.63, as opposed to 0.54, 0.36, and 0.25 for BClean, HoloClean, and Raha+Baran, respectively. When the error rate is below 30%, BClean+ achieves an average F1-score of 0.92, exceeding the three by 0.11, 0.37, and 0.56, which maintains a high performance under relatively low error rates.

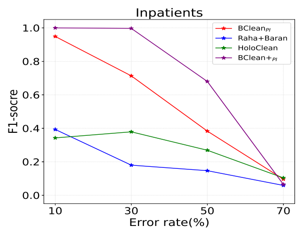
Swapping value errors. We evaluated the performance of BClean+ and other methods under conditions where 10% and 5% swapping value errors are randomly injected into Inpatient and Flights, respectively. Two types of swapping errors were considered: (i) same-domain swapping, where values within a column are assigned to the wrong records; and (ii) different-domain swapping, where values are misplaced across columns. As illustrated in Figs. 8(e) and 8(f), BClean+ and BClean consistently outperform other baselines, e.g., BClean+ (resp. BClean) achieves an average recall of 0.82 (resp. 0.83), which is 0.59, 0.11, and 0.30 (resp. 0.60, 0.12, and 0.31) higher than that of PClean, HoloClean, and Raha+Baran, respectively. This confirms the effectiveness of our automated prior generation and Bayesian-based cleaning approach.



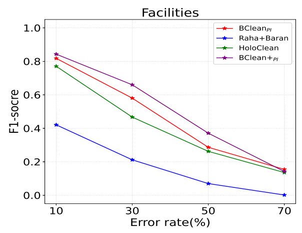
(a) Error distributions



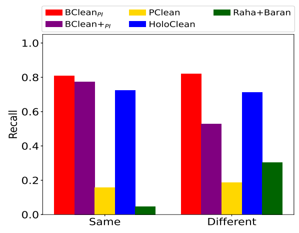
(b) Flights: varying error ratio



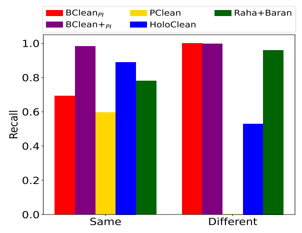
(c) Inpatient: varying error ratio



(d) Facilities: varying error ratio



(e) Inpatient: swapping value error



(f) Flights: swapping value error

Fig. 8. Error analysis: evaluating performance under diverse error conditions.

3) *Runtime Analysis*

We reported runtime statistics for all the competitors across the datasets. As demonstrated in Table VIII, BClean+ achieves a breakthrough in user time through its automated prior generation framework, requiring considerably less effort than all competing methods except Garf, which requires no user input as it is based on a generative model. Furthermore, by extending this framework to the automated synthesis of PPL code, PClean-auto reduces the user time of PClean from more than 72 hours to under 5 minutes.

Regarding execution time, BClean+ also demonstrates superior scalability. The basic version of BClean is inefficient on large datasets, e.g., it takes more than 72h on Facilities. In contrast, BClean+ requires only a few hours or even minutes, e.g., it reduces runtime on Inpatient from 7h41m to 9m48s, which in turn highlights the effectiveness of automatically generated UCs in facilitating efficient cleaning. Moreover, equipped with optimization strategies, BClean+$_{\mathrm{PI}}$ and BClean+$_{\mathrm{PIP}}$ further accelerate BClean+ by 1.53× and 3.2× on average, respectively, up to 2.77× and 9.88×. This verifies the effectiveness of the proposed partition inference and pruning strategies.

Considering the total time cost, the advantage of BClean+ is clear. With user time reduced to only a few minutes, its overall runtime is significantly lower than that of other baselines, e.g., it achieves average speedups of 42.54×, 420.84×, and 98.48× over BClean, PClean, and HoloClean, respectively. Although Garf requires no user time, its execution is typically slower. Overall, BClean+ achieves an average 113.28× speedup across all baselines, while maintaining reliable cleaning performance.

TABLE VIII
RUNTIME OF DATA CLEANING METHODS, INCLUDING USER TIME (USER) AND EXECUTION TIME (EXEC)

Method	**Time**	Hospital	Flights	Soccer	Beers	Inpatient	Facilities
BClean	user	5h	2h	3h	2h	3h	3h
	exec	25s	17s	10h48m	1m40s	7h41m	$\geq$ 72h
BClean$_{\mathrm{PI}}$	user	5h	2h	3h	2h	3h	3h
	exec	22	12s	30m42s	31s	7m57s	17m16s
BClean$_{\mathrm{PIP}}$	user	5h	2h	3h	2h	3h	3h
	exec	22s	12s	27m46s	30s	7m2s	14m35s
BClean+	user*	< 5m	< 5m	< 5m	< 5m	< 5m	< 5m
	exec	31s	43s	4h49m	2m38s	9m48s	33m15s
BClean+$_{\mathrm{PI}}$	user*	< 5m	< 5m	< 5m	< 5m	< 5m	< 5m
	exec	25s	30s	2h42m	57s	9m5s	36m59s
BClean+$_{\mathrm{PIP}}$	user*	< 5m	< 5m	< 5m	< 5m	< 5m	< 5m
	exec	17s	30s	1h55m	16s	5m40s	18m2s
PClean	user	$\geq$ 72h	$\geq$ 72h	$\geq$ 72h	$\geq$ 72h	$\geq$ 72h	$\geq$ 72h
	exec	16s	7s	30m44s	2m55s	3m17s	1m32s
PClean-auto	user*	< 5m	< 5m	< 5m	< 5m	< 5m	< 5m
	exec	15s	28s	2h	4m49s	14m45s	9m18s
HoloClean	user	15h	12h	14h	15h	15h	15h
	exec	1m 40s	36s	–	1m37s	4m14s	6m2s
Raha+Baran	user	30m	30m	30m	30m	30m	30m
	exec	1m 46s	41s	8m59s	3m2s	10m36s	10m55s
Garf	user	0	0	0	0	0	0
	exec	5m24s	1m57s	18h30m	2m8s	26m48s	30m10s

$*$ The user time is estimated and depends on the user's familiarity with the data.
– Out-of-memory.

4) *Automated PPL Synthesis*

To evaluate the effectiveness of our framework in automated PPL code synthesis, we compared the data cleaning performance of PClean, which relies on expert-authored PPL programs, and PClean-auto, which employs PPL programs automatically synthesized by our framework, across five datasets (see Table X).

As shown in Table IX, PClean-auto consistently outperforms or at least is comparable to the expert-authored PClean. The advantages on Soccer and Facilities are particularly pronounced. On Soccer, PClean-auto achieves an F1-score of 0.82, substantially higher than 0.29 for PClean. On Facilities, PClean failed to produce results due to out-of-memory or runtime errors, a common limitation when manually modeling complex schemas, whereas PClean-auto successfully synthesized viable PPL programs and achieved an F1-score of 0.71. These demonstrate the effectiveness of our automated framework in PPL synthesis, where manually specifying complete and correct PPL code is often difficult and may lead to underfitted (as in Soccer) or computationally intractable (as in Facilities) programs.

Furthermore, as noted in Table VIII, the automated PPL synthesis capability of PClean-auto reduces the user time of PClean from more than 72 hours to under 5 minutes, eliminating the heavy reliance on expert-provided PPL programs. Although its execution time is slightly higher than that of PClean, the overall time cost (user time + execution time) is still substantially lower, e.g., PClean-auto speeds up PClean by 434.91× on average, up to 822.91×. These also demonstrate the effectiveness of our automated synthesis framework.

C. *Analysis of User Interactions*

In this section, we analyzed the performance impact of three key components in BClean+: the automatically synthesized user constraints (UCs), the optional and manual manipulation of the Bayesian network (BN), and semantic type identification.

TABLE IX
PRECISION (P), RECALL (R), AND F1-SCORE (F1) OF PCLEAN-AUTO

Method	Hospital			Flights			Rents			Soccer			Facilities		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
PClean	1.000	0.927	0.962	0.907	0.884	0.895	0.679	0.684	0.682	0.184†	0.672†	0.289†	–†	–†	–†
PClean-auto	0.918	0.862	0.890	0.909	0.885	0.897	0.780	0.729	0.754	0.776	0.864	0.818	0.901	0.580	0.706

† We invite people familiar with PClean to author the data models in PPL.
– No repairs.

TABLE X
STATISTICS OF AUTO-GENERATED PPLS

	Hospital	Flights	Rents	Soccer	Facilities
Lines of PPLs	84	46	59	62	37
#classes	5	2	3	5	1
#distribution models	15	6	10	10	5

1) Impact of User Constraints

We assessed the significance of the automatically generated UCs by systematically removing specific types of constraints and examining their impact on system performance. The UCs include max length (Max), min length (Min), allowing null values (Nul), and patterns (Pat). We compared performance with each individual constraint type removed, as well as with all constraints removed (All), against the complete version (Com).

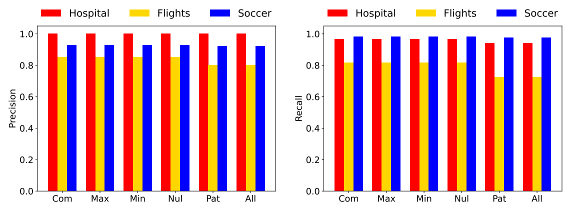
As demonstrated in Fig. 9, the Pat constraint proves to be the most influential, while removing Max, Min, and Nul leads to only minor or negligible changes in precision and recall. When users fail to provide valid pattern constraints (regular expressions), or when no patterns exist to characterize the dataset, a noticeable decline in precision and recall emerges, as shown in Figs. 9(a) and 9(b), respectively. This is because accurate pattern constraints can effectively filter out candidate values that violate user-specified requirements before the inference process begins, which is particularly valuable for datasets (e.g., Flights) in which erroneous values outnumber correct ones but violate the constraints.

Without pattern constraints, dirty values may be mistakenly treated as valid candidates during inference. For example, when repairing a null value of attribute act_arr_time in Flights, the system may assign a higher probability to an erroneous candidate (e.g., g_0 = "5:42 a.m.(Estimated runway)") than to a clean one (e.g., g_1 = "5:42 a.m."), grounding g_0 as the final but incorrect repair. In this case, the system achieves sub-optimal inference and less effective error correction with lower precision (0.80) and recall (0.72). With pattern constraints applied, however, g_0 is discarded pre-inference for non-compliance, restricting inference to the valid domain and boosting performance (precision=0.85, recall=0.82). Similarly, in Hospital, the pattern filters out the invalid candidate g_3[ZipCode] = "1xx18", since ZipCode must be a five-digit numeric string. This situation also occurs in Soccer.

These findings validate the effectiveness of our automated UC synthesis framework. By automatically generating the most critical constraints, BClean+ successfully captures the majority of the performance benefits derived from prior knowledge without requiring expert intervention. This demonstrates that BClean+ is not only user-friendly for non-experts but also efficient in design.

2) Impact of Network Manipulation

We conducted experiments where the automatically constructed BNs were modified through user interaction. For the Hospital and Soccer datasets, the results show no substantial difference before and after the modification. In Hospital, an edge state $\rightarrow$ state_avg was added, enabling the cleaning of one additional cell, while no changes were observed in Soccer. In contrast, for the Flight dataset, the BN structure automatically generated was inaccurate, leading to poor cleaning performance with a precision of only 0.258 and recall of 0.355. After user adjustments, performance improves significantly, achieving a precision of 0.852 and recall of 0.816 (see Table V). Notably, these adjustments require less than 5 minutes, suggesting that the additional user effort is minimal.



(a) Precision (b) Recall

Fig. 9. Effect of incomplete UCs on precision and recall.

Taken together, these findings demonstrate that while the automated BN construction of BClean+ is robust and reliable in most cases, the user interaction interface serves as a critical and efficient mechanism for achieving optimal performance in complex scenarios, with only negligible time overhead.

3) Impact of Semantic Type Identification

To evaluate the impact of semantic type identification on automated knowledge acquisition in BClean+, we compared two variants across six datasets: (1) BClean+$_{\text{PI}}$, which derives UCs by automatically inferring the semantic type of each attribute and selecting the optimal pattern from the corresponding candidates in the maintained template library; and (2) BClean+$_{\text{PI-nosemantic}}$, a variant of BClean+$_{\text{PI}}$ that skips semantic type inference and instead brute-forces pattern selection by matching each attribute against the entire template library, ultimately choosing the pattern with the highest syntactic match rate to auto-generate UCs. Both methods adopt partition inference (PI) optimization, with all other settings kept identical; the only difference lies in whether semantic type identification is employed during pattern selection.

As shown in Table XI, BClean+$_{\text{PI}}$ consistently outperforms BClean+$_{\text{PI-nosemantic}}$ across most datasets. On Beers, the two methods achieve comparable performance, while on Flights, BClean+$_{\text{PI-nosemantic}}$ failed to repair errors, resulting in an F1-score of zero. On the remaining datasets, BClean+$_{\text{PI}}$ achieves an average F1-score of 0.86, which is 0.36 higher than that of BClean+$_{\text{PI-nosemantic}}$. These demonstrate the necessity of semantic type identification. By clearly identifying the semantic type of each attribute, BClean+$_{\text{PI}}$ can effectively reduce the search space of candidate patterns to highly relevant ones. In contrast, the brute-force matching strategy of BClean+$_{\text{PI-nosemantic}}$ is susceptible to interference from syntactic coincidences, where a semantically incorrect pattern with a high syntactic similarity to the data may be selected, resulting in flawed repair rules and

TABLE XI
IMPACT OF SEMANTIC TYPE IDENTIFICATION IN AUTOMATED UC GENERATION

Method	Hospital			Flights			Soccer			Beers			Inpatient			Facilities		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
BClean+$_{\text{PI}}$	0.802	0.96	0.874	0.830	0.780	0.804	0.843	0.993	0.912	0.948	0.948	0.948	0.815	0.806	0.811	0.813	0.804	0.855
BClean+$_{\text{PI-nosemantic}}$	0.172	0.796	0.283	0	0	–	0.095	0.904	0.172	0.952	0.947	0.949	0.506	0.789	0.617	0.238	0.704	0.355

TABLE XII
VARYING λ ON HOSPITAL WITH $\beta = 2$ AND $\tau = 0.5$

λ	0	1	2	5	10	15
F1	0.98195	0.98195	0.98195	0.98195	0.98195	0.98195

TABLE XIII
VARYING β ON HOSPITAL WITH $\lambda = 1$ AND $\tau = 0.5$

β	0	1	2	10	50
F1	0.98195	0.98195	0.98195	0.98195	0.98195

TABLE XIV
VARYING τ ON HOSPITAL WITH $\lambda = 1$ AND $\beta = 2$

τ	0.1	0.3	0.5	0.7	0.9
F1	0.98195	0.98195	0.98195	0.97974	0.97974

a sharp decline in cleaning performance.

D. *Parameter Tuning*

In BClean+, three parameters are adjustable: λ, β, and τ. To assess their impact, we fixed two parameters at a time and varied the other. Tables XII – XIV report the F1-scores on the Hospital dataset. All three parameters have a minimal effect on the F1-score. This observation highlights that the performance of BClean+ is stable and largely unaffected by parameter changes, aligning with the easy-to-deploy design of BClean+, which requires users to spend little effort on parameter tuning.

X. RELATED WORK

Data cleaning. Data cleaning is an extensively-studied problem. We can broadly categorize the methods into five types.

- **Generative methods**: Generative data cleaning methods integrate error detection and correction components, often leveraging probabilistic inference to iteratively clean datasets towards maximum likelihood [25], [26], [50], [51]. Other notable methods in this category use PPLs [4], [24] or user queries [6], [18]. Generative models are data-driven and do not require any user labels, but many of them necessitate some prior knowledge to achieve good performance.
- **Rule-based methods**: These employ data quality rules that check data inconsistency, such as FDs [52], conditional FDs [32], [53], DCs [45], user-defined functions [9], [54], and regular expression entities [55]. Rule discovery algorithms [56]–[58] have been proposed to mine these rules from relatively clean training data. Correspondingly, error detection [59] and data cleaning algorithms [7], [60] have been designed to exploit these rules to efficiently locate and correct errors in big data.
- **ML pipeline**: ML models have been widely used for data clean. For instance, SCAREd [17] adopts ML and likelihood methods. In CleanML [61], five error types were evaluated, and the impact of data cleaning on the ML pipeline was discussed. Picket [62] employs a self-supervised strategy to rectify data errors in ML pipelines. [63] proposed a novel framework based on Shapley values to interpret the results of any data cleaning module. ActiveClean [64] is a semi-supervised model with convex loss functions for data cleaning.
- **ML imputation**: Additionally, many ML methods focus on imputing missing values. A common approach is the denoising autoencoder [65]. More recent approaches use generative adversarial networks (GANs) [66], the attention mechanism [67], [68], or both [69]. However, many of them require considerable effort in tuning hyperparameters and training the model, especially for GAN-based methods.
- **Hybrid models**: These methods combine logical rules and ML models, e.g., by encoding rules as features for ML prediction or integrating auxiliary components to reduce manual effort, e.g., [1], [3], [5], [6]. Recent methods also leverage LLMs for cleaning, e.g., ZeroEC [70] and GIDCL [71]; both typically require a small labeling budget (e.g., 20 tuples) for demonstrations.

In contrast, BClean performs UC-guided Bayesian cleaning with BN construction refined via lightweight user interaction, and BClean+ automates LLM-based knowledge acquisition and continuously maintains a reusable template library, further enabling automated PPL synthesis (e.g., for PClean).

Bayesian network construction. Existing BN structure learning methods include automatic greedy search [21], [38], [72] and user-constructed networks [4], [24]. Automatic methods tend to be weakly robust to dirty data due to erroneous propagation, while user-constructed methods often incur high user costs when dealing with large datasets. In contrast, BClean supports interaction-based refinement of BN structure, and BClean+ further introduces Infomap-based modularization to improve interpretability and facilitate user validation and modification.

Bayesian inference. Bayesian inference can be categorized into exact inference and approximate inference. Exact inference yields posterior probability distributions, but methods like variable elimination and belief propagation [21]can be computationally intensive and susceptible to erroneous propagation. Approximate inference, based on sampling techniques such as Gibbs sampling [73], typically trades runtime improvement for accuracy. Exact inference is challenging to execute outside clean data, and approximate inference can propagate errors when sampling dirty data. BClean partitions the BN to enable efficient approximate inference, while BClean+ keeps the same mechanism but benefits from higher-quality automatically generated priors, yielding more robust inference and repair, with significantly reduced runtime.

XI. CONCLUSION

In this paper, we introduced BClean+, an enhanced Bayesian data cleaning system that extends our earlier work BClean with a novel framework for automated prior generation. The main contributions of BClean+ include: (1) an automated prior generation framework that significantly reduces reliance on manually provided priors; (2) a hierarchical structure discovery method that enhances BN construction by improving interpretability and enabling more effective refinement for accurate inference; (3) the integration with LLMs to bootstrap the entire automation process; and (4) the generalization to automated PPL Synthesis for systems such as PClean, addressing a critical usability challenge in PPL-based data cleaning. Extensive experiments demonstrate that BClean+ reduces user configuration time from hours to under five minutes while achieving state-of-the-art performance and significantly outperforming other leading methods.

Despite the promising results, BClean+ has several limitations. First, it primarily addresses cell-level syntactic and semantic errors and does not explicitly handle quantitative errors, such as aggregate-level numerical inconsistencies or systematic measurement biases. Second, its repair quality depends on the learned BN structure and may degrade when attribute dependencies are weak or poorly captured. Third, it operates under a closed-domain assumption, restricting repairs to a fixed set of candidate values and thereby limiting its ability to correct rare or previously unseen valid values. We leave incorporating quantitative constraints, strengthening BN learning, and leveraging external knowledge to enable open-domain repairs as future work.

Acknowledgments

This work was supported by NSFC 62472289, 62532007, 62502325, and Guangdong Province Key Laboratory of Popular High Performance Computers 2017B030314073.

References

[1] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "HoloDetect: Few-shot learning for error detection," in *SIGMOD*, 2019, pp. 829–846.

[2] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, "Detecting data errors: Where are we and what needs to be done?" *PVLDB*, vol. 9, no. 12, pp. 993–1004, 2016.

[3] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *SIGMOD*, 2019, pp. 865–882.

[4] A. K. Lew, M. Agrawal, D. A. Sontag, and V. Mansinghka, "PClean: Bayesian data cleaning at scale with domain-specific probabilistic programming," in *AISTATS*, 2021, pp. 1927–1935.

[5] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *PVLDB*, vol. 10, no. 11, pp. 1190–1201, 2017.

[6] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *PVLDB*, vol. 13, no. 11, pp. 1948–1961, 2020.

[7] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas, "Guided data repair," *PVLDB*, vol. 4, no. 5, pp. 279–289, 2011.

[8] S. Kolahi and L. V. S. Lakshmanan, "On approximating optimum repairs for functional dependency violations," in *ICDT*, 2009, pp. 53–62.

[9] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang, "NADEEF: a commodity data cleaning system," in *SIGMOD*, 2013, pp. 541–552.

[10] W. Fan, S. Ma, N. Tang, and W. Yu, "Interaction between record matching and data repairing," *JDIQ*, vol. 4, no. 4, pp. 1–38, 2014.

[11] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye, "KATARA: A data cleaning system powered by knowledge bases and crowdsourcing," in *SIGMOD*, 2015, pp. 1247–1261.

[12] Y. Zheng, J. Wang, G. Li, R. Cheng, and J. Feng, "QASCA: A quality-aware task assignment system for crowdsourcing applications," in *SIGMOD*, 2015, pp. 1031–1046.

[13] J. Wang and N. Tang, "Towards dependable data repairing with fixing rules," in *SIGMOD*, 2014, pp. 457–468.

[14] E. K. Rezig, M. Ouzzani, A. K. Elmagarmid, W. G. Aref, and M. Stonebraker, "Towards an end-to-end human-centric data cleaning framework," in *HILDA*, 2019, pp. 1–7.

[15] P. H. Oliveira, D. S. Kaster, C. T. Jr., and I. F. Ilyas, "Batchwise probabilistic incremental data cleaning," *arXiv preprint arXiv:2011.04730*, 2020.

[16] C. Mayfield, J. Neville, and S. Prabhakar, "A statistical method for integrated data cleaning and imputation," 2009.

[17] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid, "Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes," in *SIGMOD*, 2013, pp. 553–564.

[18] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan, "Data cleaning and query answering with matching dependencies and matching functions," in *ICDT*, 2011, pp. 268–279.

[19] J. Kubica and A. W. Moore, "Probabilistic noise identification and data cleaning," in *ICDM*, 2003, pp. 131–138.

[20] W. Li, L. Li, Z. Li, and M. Cui, "Statistical relational learning based automatic data cleaning," *Frontiers of Computer Science*, vol. 13, no. 1, pp. 215–217, 2019.

[21] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[22] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," *JMLR*, vol. 20, no. 1, pp. 973–978, 2019.

[23] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *JSS*, vol. 76, no. 1, pp. 1–32, 2017.

[24] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov, "1 blog: Probabilistic models with unknown objects," *Statistical Relational Learning*, p. 373, 2007.

[25] Y. Hu, S. De, Y. Chen, and S. Kambhampati, "Bayesian data cleaning for web data," *arXiv preprint arXiv:1204.3677*, 2012.

[26] S. De, Y. Hu, V. V. Meduri, Y. Chen, and S. Kambhampati, "BayesWipe: A scalable probabilistic framework for improving data quality," *JDIQ*, vol. 8, no. 1, pp. 5:1–5:30, 2016.

[27] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han, "A bayesian approach to discovering truth from conflicting sources for data integration," *PVLDB*, vol. 5, no. 6, pp. 550–561, 2012.

[28] Y. Zhang, Z. Guo, and T. Rekatsinas, "A statistical perspective on discovering functional dependencies in noisy data," in *SIGMOD*, 2020, pp. 861–876.

[29] J. Qin, S. Huang, Y. Wang, J. Zhu, Y. Zhang, Y. Miao, R. Mao, M. Onizuka, and C. Xiao, "BClean: A bayesian data cleaning system," in *ICDE*, 2024, pp. 3407–3420.

[30] A. Bartoli, A. D. Lorenzo, E. Medvet, and F. Tarlao, "Inference of regular expressions for text extraction from examples," *TKDE*, vol. 28, no. 5, pp. 1217–1230, 2016.

[31] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Regex Generator++," 2016, http://regex.inginf.units.it/.

[32] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *ICDE*, 2007, pp. 746–755.

[33] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian, "Metric functional dependencies," in *ICDE*, 2009, pp. 1275–1278.

[34] S. Wu, S. Sanghavi, and A. G. Dimakis, "Sparse logistic regression learns all discrete pairwise graphical models," in *NeurIPS*, 2019, pp. 8069–8079.

[35] G. Raskutti and C. Uhler, "Learning directed acyclic graph models based on sparsest permutations," *Stat*, vol. 7, no. 1, p. e183, 2018.

[36] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *PNAS*, vol. 105, pp. 1118 – 1123, 2007.

[37] M. Chickering, D. Geiger, and D. Heckerman, "Learning bayesian networks: Search methods and experimental results," in *AISTATS*, 1995.

[38] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine Learning*, vol. 65, no. 1, pp. 31–78, 2006.

[39] A. Ankan and A. Panda, "pgmpy: Probabilistic graphical models using python," in *SciPy*, 2015, pp. 6–11.

[40] P. L. Spirtes and C. Glymour, "An algorithm for fast recovery of sparse causal graphs," *SSCR*, vol. 9, pp. 62 – 72, 1991.

[41] P.-L. Loh and P. Bühlmann, "High-dimensional learning of linear causal networks via inverse covariance estimation," *JMLR*, vol. 15, no. 1, pp. 3065–3105, 2014.

[42] D. Edler, L. Bohlin, and M. Rosvall, "Mapping higher-order network flows in memory and multilayer networks with infomap," *Algorithms*, vol. 10, no. 4, p. 112, 2017.

[43] B. Feuer, Y. Liu, C. Hegde, and J. Freire, "Archetype: A novel framework for open-source column type annotation using large language models," *PVLDB*, vol. 17, no. 9, p. 2279–2292, 2024.

[44] tdda, "Python tdda (version 2025.1.20)," 2025, https://github.com/tdda/tdda.

[45] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in *ICDE*, 2013, pp. 458–469.

[46] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava, "Truth finding on the deep web: Is the problem solved?" *arXiv preprint arXiv:1503.00303*, 2015.

[47] J. Rammelaere and F. Geerts, "Explaining repaired data with cfds," *PVLDB*, vol. 11, no. 11, pp. 1387–1399, 2018.

[48] Centers for Medicare & Medicaid Services, "Provider data catalog," https://data.cms.gov/provider-data/.

[49] J. Peng, D. Shen, N. Tang, T. Liu, Y. Kou, T. Nie, H. Cui, and G. Yu, "Self-supervised and interpretable data cleaning with sequence generative adversarial networks," *PVLDB*, vol. 16, no. 3, pp. 433–446, 2022.

[50] C. D. Sa, I. F. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas, "A formal framework for probabilistic unclean databases," in *ICDT*, 2019, pp. 6:1–6:18.

[51] P. Doshi, L. G. Greenwald, and J. R. Clarke, "Using bayesian networks for cleansing trauma data," in *FLAIRS*, 2003, pp. 72–76.

[52] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[53] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, vol. 33, no. 2, pp. 6:1–6:48, 2008.

[54] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The LLUNATIC data-cleaning framework," *PVLDB*, vol. 6, no. 9, pp. 625–636, 2013.

[55] W. Fan, P. Lu, and C. Tian, "Unifying logic rules and machine learning for entity enhancing," *Science China Information Sciences*, vol. 63, no. 7, p. 172001, 2020.

[56] W. Fan, F. Geerts, J. Li, and M. Xiong, "Discovering conditional functional dependencies," *TKDE*, vol. 23, no. 5, pp. 683–698, 2011.

[57] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong, "Discovering conditional functional dependencies," in *ICDE*, 2009, pp. 1231–1234.

[58] W. Fan, Z. Han, Y. Wang, and M. Xie, "Parallel rule discovery from large datasets by sampling," in *SIGMOD*, 2022, pp. 384–398.

[59] W. Fan, C. Tian, Y. Wang, and Q. Yin, "Parallel discrepancy detection and incremental detection," *PVLDB*, vol. 14, no. 8, pp. 1351–1364, 2021.

[60] W. Fan, F. Geerts, and X. Jia, "Semandaq: a data quality system based on conditional functional dependencies," *PVLDB*, vol. 1, no. 2, pp. 1460–1463, 2008.

[61] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: A study for evaluating the impact of data cleaning on ML classification tasks," in *ICDE*, 2021, pp. 13–24.

[62] Z. Liu, Z. Zhou, and T. Rekatsinas, "Picket: guarding against corrupted data in tabular data during learning and inference," *VLDB Journal*, vol. 31, no. 5, pp. 927–955, 2022.

[63] D. Deutch, N. Frost, A. Gilad, and O. Sheffer, "Explanations for data repair through shapley values," in *CIKM*, 2021, pp. 362–371.

[64] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "ActiveClean: Interactive data cleaning for statistical modeling," *PVLDB*, vol. 9, no. 12, pp. 948–959, 2016.

[65] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *JMLR*, vol. 11, pp. 3371–3408, 2010.

[66] J. Yoon, J. Jordon, and M. van der Schaar, "GAIN: missing data imputation using generative adversarial nets," in *ICML*, 2018, pp. 5675–5684.

[67] S. Tihon, M. U. Javaid, D. Fourure, N. Posocco, and T. Peel, "DAEMA: denoising autoencoder with mask attention," in *ICANN*, 2021, pp. 229–240.

[68] R. Wu, A. Zhang, I. F. Ilyas, and T. Rekatsinas, "Attention-based learning for missing data imputation in holoclean," in *MLSys*, 2020.

[69] J. Kawagoshi, Y. Dong, T. Nozawa, and C. Xiao, "CAGAIN: Column attention generative adversarial imputation networks," in *DEXA*, vol. 14147, 2023, pp. 1–16.

[70] Y. Wu, C. Yang, M. Zhu, X. Miao, W. Ni, M. Xi, X. Zhao, and J. Yin, "A zero-training error correction system with large language models," in *ICDE*, 2025, pp. 2949–2962.

[71] M. Yan, Y. Wang, Y. Wang, X. Miao, and J. Li, "GIDCL: A graph-enhanced interpretable data cleaning framework with large language models," *SIGMOD*, vol. 2, no. 6, pp. 1–29, 2024.

[72] K. Tzoumas, A. Deshpande, and C. S. Jensen, "Lightweight graphical models for selectivity estimation without independence assumptions," *PVLDB*, vol. 4, no. 11, pp. 852–863, 2011.

[73] C. Zhang and C. Ré, "DimmWitted: A study of main-memory statistical analytics," *PVLDB*, vol. 7, no. 12, pp. 1283–1294, 2014.

Ziyan Han received the B.E. degree in Computer Science and Technology from Xidian University, Xi'an, China, in 2018, and the Ph.D. degree in Computer Software and Theory from Beihang University, Beijing, China, in 2025. She has published papers in top-tier database conferences, including SIGMOD and ICDE. Her research interests include data management, data quality, data mining, and machine learning.

Jing Zhu received the B.E. degree in Data Science and Big Data Technology from Guangdong University of Technology, Guangzhou, China, in 2023. She is currently pursuing the M.S. degree in Computer Science and Technology at Shenzhen University, Shenzhen, China. She has published papers in top-tier database conferences such as ICDE. Her research interests include data management, data quality, data cleaning, and machine learning.

Jinbin Huang received the Ph.D. degree from Hong Kong Baptist University (HKBU), Hong Kong, in 2024. He is currently an Assistant Professor in the School of Artificial Intelligence at Shenzhen University, Shenzhen, China. His research interests include data management and graph mining algorithms.

Sifan Huang received the B.E. degree in Software Engineering from Guangdong University of Technology, Guangzhou, China, in 2020, and the M.S. degree in Computer Technology from Shenzhen University, Shenzhen, China, in 2023. He has published papers in top-tier database conferences such as ICDE. His research interests include data management, data quality, data cleaning, and machine learning.

Yaoshu Wang received the Ph.D. degree from The University of New South Wales, Australia, in 2018. He is currently a Senior Researcher at Shenzhen Institute of Computing Science, Shenzhen, China. His research interests include data quality, big data, machine learning, and large language models.

Rui Mao received the B.S. degree (1997) and the M.S. degree (2000) in Computer Science from the University of Science and Technology of China, and the M.S. degree (2006) in Statistics and the Ph.D. degree (2007) in Computer Science from The University of Texas at Austin. He is currently a Distinguished Professor in the College of Computer Science and Software Engineering at Shenzhen University, and the Executive Director of Shenzhen Institute of Computing Sciences. His research interests include metric-space data processing, parallel computing, data mining, and machine learning.

Jianbin Qin received the B.E. degree from Northeastern University, China, in 2007, and the Ph.D. degree from The University of New South Wales, Australia, in 2013. He is currently a Distinguished Professor in the College of Computer Science and Software Engineering at Shenzhen University, and a Research Scientist at Shenzhen Institute of Computing Sciences. His research interests include database theory, database systems, similarity query, information retrieval, and data management.