MAPPipe: A System for Bridging Efficiency and Quality in Data Preprocessing via Knowledge-Augmented Structural Pruning

Xiaoou Ding
Harbin Institute of Technology
dingxiaoou@hit.edu.cn

Haifeng Cheng
Harbin Institute of Technology
2021112251@stu.hit.edu.cn

Yanshuo Liu
Harbin Institute of Technology
25S103214@stu.hit.edu.cn

Chen Wang
Tsinghua University
wang_chen@tsinghua.edu.cn

Hongzhi Wang*
Harbin Institute of Technology
wangzh@hit.edu.cn

ABSTRACT

Despite significant progress in automated data preprocessing, existing systems still face an unfavorable trade-off between optimization quality and computational efficiency, producing pipelines that are either fast but suboptimal or effective but prohibitively slow. This paper presents MAPPipe, an demo-ready system that efficiently constructs near-optimal pipelines by bridging this gap through a knowledge-augmented, surrogate-free architecture that unifies pipeline recommendation, reconstruction, and search, transforming exponential combinatorial optimization into guided exploration within a high-potential subspace. MAPPipe supports six core preprocessing operations (missing value imputation, outlier detection, encoding, normalization, feature transformation, and feature selection), and provides an interactive, visual interface that enables users to inspect, intervene, and inject domain knowledge with minimal effort. Experiments on the DiffPrep show that MAPPipe achieves the best accuracy on 15 datasets (avg. Acc 84.9%), outperforming CtxPipe by 4.3%, and reducing the gap to the near-optimal solution to 0.2%, all within a 94s runtime. This demonstrates MAPPipe's ability to deliver accurate, interpretable, and user-friendly data preprocessing under practical time constraints for data engineers.

PVLDB Reference Format:
Xiaoou Ding, Haifeng Cheng, Yanshuo Liu, Chen Wang, and Hongzhi Wang. MAPPipe: A System for Bridging Efficiency and Quality in Data Preprocessing via Knowledge-Augmented Structural Pruning. PVLDB, 19(12): 4682 - 4685, 2026.
doi:10.14778/3827998.3828096
PVLDB Artifact Availability:
The source code, data, and other artifacts have been made available at https://github.com/sunnychf/MAPPipe_Demo.

1 Introduction

In data application tasks, *data quality* fundamentally determines model performance [1]. While high-quality data enhances a model's learning capacity, real-world datasets often suffer from missing values, outliers, and noise[2, 3], rendering them unsuitable for direct modeling. To ensure reliable learning, data preprocessing, comprising transformation and standardization steps such as imputation, outlier handling, encoding, normalization, and feature engineering, is essential. These operations are typically organized into a modular data preprocessing pipeline, which systematically converts raw input into *model-ready* features [6, 8].

Now, various automated preprocessing pipeline construction systems have been proposed, such as SAGA [5] (genetic programming), HAIpipe [6] and CtxPipe [7] (deep reinforcement learning), and DiffPrep [8] (differentiable relaxation). These systems reduce manual effort and can provide pipelines that improve model performance. Notably, CtxPipe achieves end-to-end automation, advancing the field toward practical deployment. However, there remains a trade-off between optimization quality and computational efficiency, i.e., "accurate vs. fast". Even CtxPipe falls short of optimal performance, i.e., on 18 public datasets, its pipelines lag behind the near-optimal configurations by 7.6% on average (up to 15% in some cases), which indicates significant room for improvement. Moreover, although these systems can automatically generate preprocessing pipelines, they usually provide limited visibility into how candidate pipelines are recommended, simplified, and optimized, which makes the overall construction process less transparent to users.

To overcome these limitations, we aim to build a preprocessing pipeline construction system that approaches near-optimal performance while remaining computationally efficient. This goal faces two challenges. (i) **The combinatorial explosion of the search space**. With M preprocessing modules and an average of K candidate implementations per module, the number of possible pipelines grows as $K^M \cdot M!$, making exhaustive exploration computationally infeasible. (ii) **The high cost of pipeline evaluation**. Assessing each candidate pipeline requires executing preprocessing and retraining the downstream model to estimate its performance. As the number of candidates and dataset scale increase, evaluation quickly becomes the dominant bottleneck in automated pipeline search.

To address these challenges, we present MAPPipe, an automated data preprocessing pipeline construction system that delivers near-optimal pipelines under practical time constraints. Meanwhile, MAPPipe emphasizes interpretability in the pipeline construction process, enabling users to inspect intermediate results and understand how the final pipeline is produced. The system highlights the following features:

(1) An surrogate-free end-to-end workflow for automated preprocessing pipeline construction: MAPPipe integrates pipeline recommendation (via meta-feature matching), pipeline reconstruction (redundancy removal and equivalent fusion), and pipeline search (customized MCTS) into a unified system. It helps users

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828096

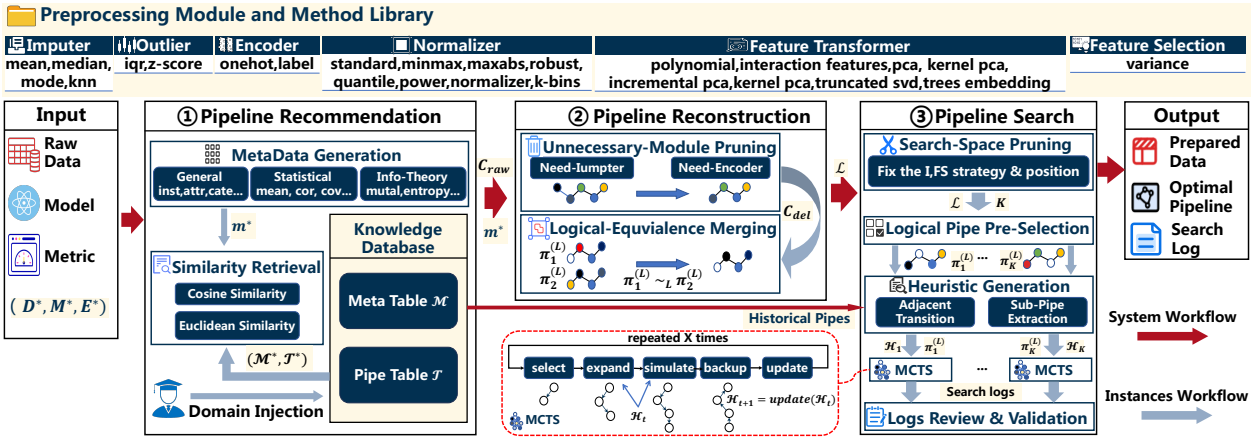

Figure 1: Overview of the MAPPipe system.

efficiently obtain near-optimal pipelines for a given classification task within a reduced search space. **(2) An empirically grounded strategy for search-space reduction:** MAPPipe adopts a pruning strategy based on empirical analysis to substantially reduce the search space while maintaining strong optimization quality. Our analysis shows that variations in standard imputation strategies have negligible impact on performance (difference < 0.005 in more than 95% of cases). Accordingly, the system fixes the imputation module at the pipeline start and the feature selection module at the end, reducing the physical search space from 207,360 to 6,912. **(3) An inspectable knowledge-guided search process:** MAPPipe constructs an adjacent-transition heuristic scoring table from historical high-performing pipelines and uses it to guide MCTS toward high-potential branches. This design improves search efficiency under limited budgets and makes the optimization process more transparent by exposing how prior knowledge affects intermediate exploration and the final pipeline recommendation.

Together, these features allow MAPPipe to construct near-optimal preprocessing pipelines in under two minutes on standard benchmarks [8]. It offers data scientists and ML engineers a fast, accurate, and interpretable tool for automating one of the most time-consuming yet critical stages of the machine learning workflow.

2 System Overview

Figure 1 outlines the architecture of MAPPipe that automatically constructs high-quality data preprocessing pipelines through three integrated stages:

- ***Pipeline Recommendation***. This stage mitigates the cold-start problem through an offline-online design. Offline, MAPPipe maintains a task-specific knowledge base that links dataset meta-features to high-performing pipelines under fixed downstream model and evaluation metric settings. Online, for a new task, it retrieves the Top-m most similar datasets and aggregates their Top-k pipelines as Candidates(C_{raw}). Upon completing evaluation for a new task, the system directly appends its results to the knowledge base. Users can also upload JSON files via interfaces to embed domain knowledge. This stage restricts the search space to a high-potential subspace of size at most $m * k$, thereby minimizing blind exploration during the search phase.
- ***Pipeline Reconstruction***. Since C_{raw} originate from historical tasks, they exhibit task bias and structural redundancy. Direct reuse would waste resources. Therefore, this stage employs a Delete-Fuse strategy to adapt C_{raw} to the task and streamline their structure. First, a *Deletion* step removes modules unnecessary for the current data characteristics (e.g., discarding imputation if no missing values exist), providing a refined set C_{del} and reducing search tree depth. Second, a *Fusion* step deduplicates logically equivalent pipelines within C_{del}, minimizing the number of parallel search trees required. The resulting compact candidate set is denoted as $\mathcal{L}$, which is exposed via an interactive interface, allowing users to inspect or refine.
- ***Pipeline Search***. This stage identifies the optimal physical pipeline from $\mathcal{L}$. Since rule-based reconstruction offers limited compression when no modules are removable, we further constrain the space using statistical insights: fixing imputation at the start and feature selection at the end. We then perform logical pipeline pre-selection over $\mathcal{L}$ and retain the top-K logical pipelines. Each selected pipeline initializes a parallel MCTS instance guided by a custom heuristic scoring table $\mathcal{H}$. These intermediate results are visualized in the UI, and users can directly modify entries in $\mathcal{H}$ to inject domain knowledge. The final optimal pipeline is derived from aggregating these search logs.

3 Implementation Details

Preliminaries. To support knowledge transfer and pipeline search, we formalize a data preprocessing pipeline as a two-level hierarchical structure consisting of a *logical pipeline* and a *physical pipeline*. A *logical pipeline* is denoted by $\pi^{(L)} = [o_1, o_2, \ldots, o_n]$, where each $o_i \in O$ is a preprocessing module and $\pi^{(L)}(i)$ denotes the i-th module in the logical pipeline. In our setting, O = {**I,O,E,N,FT,FS**}, corresponding to imputation, outlier handling, encoding, normalization, feature transformation, and feature selection. A *physical pipeline* is denoted by $\pi^{(P)} = [f_1, f_2, \ldots, f_n]$, where each $f_i \in \text{Instance}(\pi^{(L)}(i))$ is a implementation of the module at position i. Hence, the $\pi^{(L)}$ specifies what modules are performed, while the $\pi^{(P)}$ specifies how these modules are instantiated.

3.1 Knowledgebase construction and retrival

In the **offline** phase, MAPPipe builds a conditionally organized knowledge base from OpenML-CC18 and AutoML benchmarks. To avoid conflicting priors across tasks, historical pipelines are partitioned by downstream model M and evaluation metric E. For each configuration (M_l, E_r), the corresponding subset is $C_{lr} = \{t_{ijk} \mid$

$M_j = M_l,\ E_k = E_r\}$. Each subset maintains two components: a *meta-feature table* $\mathcal{M}$ and a *pipeline experience table* $\mathcal{T}$. The meta-feature table is defined as $\mathcal{M} = \{(d_i, m_i)\}_{i=1}^{N}$, where d_i is the dataset identifier and $m_i \in \mathbb{R}^p$ is the corresponding meta-feature vector. In our implementation, $p = 13$, and these features characterize dataset scale, feature composition, missingness, class-distribution complexity, and statistical properties. The pipeline experience table is defined as $\mathcal{T} = \left\{\left(d_i, \{(\phi_{ij}, \pi_{ij}^{(L)}, \pi_{ij}^{(P)})\}_{j=1}^{k}\right)\right\}_{i=1}^{N}$, where, for each dataset d_i, we store the Top-k high-performing pipelines together with their validation performance ϕ_{ij}, logical pipeline $\pi_{ij}^{(L)}$, and physical pipeline $\pi_{ij}^{(P)}$. In practice, MAPPipe adopts a dataset-centric storage scheme, where each d_i is associated with a separate `JSON` file for incremental maintenance.

During **inference**, given a new task with meta-feature vector m^* under configuration (M^*, E^*), MAPPipe loads the corresponding subset $(\mathcal{M}^*, \mathcal{T}^*)$ and computes cosine similarity between m^* and all vectors in $\mathcal{M}^*$, retrieves the Top-m most similar datasets, and aggregates their Top-k pipelines into the initial candidate set $C_{\text{raw}} = \{d_i \in \mathcal{S}\}\mathcal{T}_i^{(k)}$, where $\mathcal{S}$ denotes the retrieved Top-m similar datasets. We use cosine similarity by default because it is less sensitive to scale differences among meta-features and effectively captures directional alignment in the meta-feature space. Empirically, we set $m = 5$ and $k = 5$ to balance diversity and efficiency.

3.2 Space Pruning & Pipeline Search

To refine the initial candidate set, MAPPipe applies a *delete–fuse* strategy. **Delete**: For three explicitly evaluable modules, i.e., missing value imputation (**I**), outlier handling (**O**), and encoding (**E**), the system removes a module if its corresponding data characteristic is absent (e.g., no missing values, no outliers, or no categorical features). This shortens pipelines and reduces MCTS tree depth. **Fuse**: Pipelines sharing the same logical structure are merged into a compact set $\mathcal{L} = \{(\pi^{(L)}, \mathcal{D}, \mathcal{P})\}$, where each tuple consists of a logical pipeline, its supporting datasets, and its corresponding physical pipelines. This reduces redundancy and lowers the number of search trees. **Statistical pruning**: To further constrain the physical search space, MAPPipe fixes **I** at the beginning of the pipeline and **FS** at the end. Empirical analysis shows that, within complex preprocessing pipelines, different standard imputation strategies typically lead to only negligible differences in downstream accuracy, which supports fixing **I**. Since MAPPipe focuses on data preparation, FS is restricted to lightweight feature selection methods suitable for the preprocessing stage, rather than wrapper-based methods, and is therefore fixed as the final step. With instantiation counts of 2, 2, 8, and 9 for **O**, **E**, **N**, and **FT**, respectively, the physical search space per logical pipeline is 288, yielding a total of 6,912 configurations. This represents a 96.67% reduction from the original 207,360 possibilities, substantially improving search efficiency. Figure 2(a) shows an example of space pruning, in which $\pi_1^{(L)}$ and $\pi_2^{(L)}$ are merged into a single logical pipeline $\pi_3^{(L)}$ by applying the delete-fuse strategy and statistical pruning.

Given $\mathcal{L}$, MAPPipe finds the optimal physical pipeline through logical pipeline pre-selection, warm-started heuristic MCTS, and log-based final selection.

Logical pipeline pre-selection. Directly searching all candidates in $\mathcal{L}$ remains expensive. For each logical pipeline $(\pi^{(L)}, \mathcal{D}, \mathcal{P}) \in \mathcal{L}$,

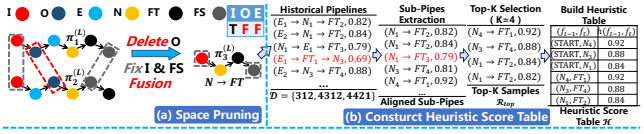

Figure 2: An example of Prune and Heuristic scoring table construction.

MAPPipe first constructs a default physical pipeline $\hat{\pi}^{(P)}$ by selecting, for each module, the most frequent instantiation method in $\mathcal{P}$. $\hat{\pi}^{(P)}$ is evaluated to obtain $\hat{\phi}$, and logical candidates are ranked accordingly. Only the top-K logical pipelines are retained for subsequent search, which reduces the number of search trees while preserving the most promising structures.

Heuristic scoring table construction. To guide search without surrogate models, MAPPipe summarizes historical high-performing pipelines into an heuristic scoring table $\mathcal{H}$. Let f_t denote the instantiation method selected at step t, with $f_0 = \varnothing$. Then each adjacent transition (f_{t-1}, f_t) is assigned a score $\mathcal{H} : (f_{t-1}, f_t) \mapsto h(f_{t-1}, f_t)$. Instead of relying only on the limited $\mathcal{P}$ in $\mathcal{L}$, MAPPipe extracts sub-pipeline evidence from historical pipelines and keeps the top-K matched records by performance. For any (f_{t-1}, f_t), its heuristic score is defined as $h(f_{t-1}, f_t) = \max\{\phi \mid \exists(\pi'^{(L)}, \pi'^{(P)}, \phi) \in R_{\text{top}},\ (f_{t-1}, f_t)$ is an adjacent pair in $\pi'^{(P)}\}$. Using the *maximum* score provides an optimistic weak prior, which is more effective than *mean* score under limited and sparse historical samples. Figure 2(b) illustrates the complete process of constructing the heuristic scoring table for the logical pipeline $\pi_3^{(L)}$.

Customized MCTS. For each retained logical pipeline, MAPPipe launches an independent MCTS process in parallel. The default pipeline $(\hat{\pi}^{(P)}, \hat{\phi})$ is used to initialize the search tree. Given a logical pipeline $\pi^{(L)} = [o_1, \ldots, o_n]$, the node at depth t stores only the previous instantiation method $s_t = f_{t-1}, f_0 = \varnothing$, and the action selects the instantiation method for the current module $a_t = f_t \in \mathcal{A}(s_t) = \textsc{Instance}(o_t)$. The four phases are following. **Selection.** If the current node is fully expanded, MCTS descends according to the UCB: $s_{t+1}^* \leftarrow \arg\max_{s_{t+1}} (Q(s_{t+1}) + c\sqrt{\frac{\ln N(s_t)}{N(s_{t+1})}})$, where $N(\cdot)$ is the visit count. Unlike standard MCTS, MAPPipe uses the *maximum* rollout score as Q, so that high-reward branches can be emphasized under small budgets. **Expansion.** If the node is not fully expanded, an untried action is selected using an ε-greedy strategy: with probability ε, f_t is sampled uniformly from U_t; otherwise, it is sampled from a softmax distribution over U_t based on $h(cond(s_t), f)$, where U_t is the untried action set, $cond(s_t) = \varnothing$ for $t = 1$, and $cond(s_t) = f_{t-1}$ otherwise. **Simulation.** After expansion, the remaining undecided modules are completed by heuristic-guided rollout. For each remaining module o_j, the instantiation method is sampled by $p(f_j = f \mid cond(s_j)) = \frac{\exp(h(cond(s_j); f))}{\sum_{f' \in \textsc{Instance}(o_j)} \exp(h(cond(s_j); f'))}$. This produces a complete physical pipeline $\pi^{(P)} = [f_1, \ldots, f_n]$, which is then evaluated by executing the pipeline and training the downstream model on a data subset to obtain the score ϕ. **Backpropagation.** After each rollout, the obtained score ϕ is propagated from the leaf to the root: $N(v) \leftarrow N(v) + 1, Q(v) \leftarrow \max\{Q(v), \phi\}$. Meanwhile, the heuristic table is updated. Let the current rollout score be ϕ_t and the previous one be ϕ_{t-1}, with $\Delta = \phi_t - \phi_{t-1}$. Then for each transition (f_i, f_{i+1}) in the sampled pipeline, $h(f_i, f_{i+1}) \leftarrow h(f_i, f_{i+1}) + \alpha\Delta$, and the start transition is updated by $h(f_0, f_1) \leftarrow h(f_0, f_1) + \alpha \max(\Delta, 0)$.

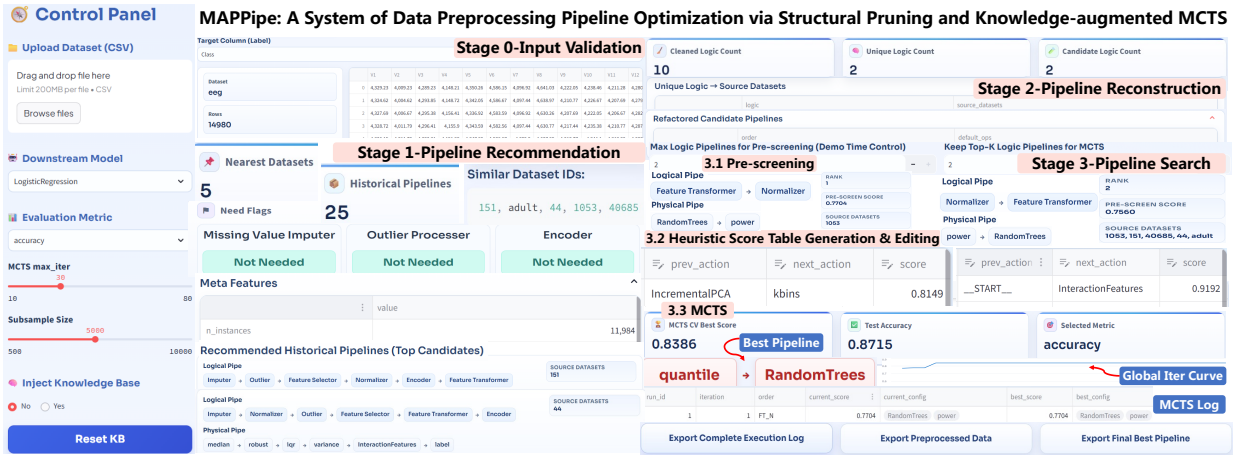

Figure 3: Pages demonstration in MAPPipe.

Table 1: Overall results on DiffPrep.

Method	Acc. ↑	Rank ↓	Time (s) ↑	F1 ↑
ES	0.851	–	–	–
DL	0.704	7.444	**9.3**	0.570
DEF	0.724	7.083	36.2	0.604
SAGA	0.731	5.917	1384.8	0.618
RS	0.761	5.889	803.9	0.650
HAI-AI	0.760	5.722	22.20	0.613
DP-Fix	0.780	4.306	1024.8	0.687
DP-Flex	0.784	3.861	11148.9	0.695
CtxPipe	0.806	3.194	65.20	0.745
MAPPipe	**0.849**	**1.583**	94.0	**0.798**

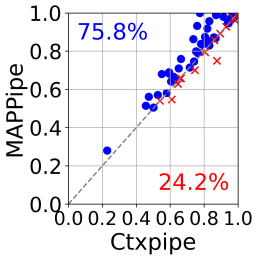

Figure 4: Comparison on Extend.

Log review and final selection. After all parallel searches finish, MAPPipe aggregates search logs and keeps the top-performing candidates. Pipelines whose score gap to the best result is within threshold δ are retained. Finally, cross-validation is applied to determine the optimal physical pipeline.

4 Demonstration

4.1 Experimental Results

Table 1 reports average performance of automated data preprocessing systems on the DiffPrep benchmark [8], using the ranking metric from CtxPipe (lower is better) [7]. MAPPipe achieves the best results across all metrics: test accuracy of 0.849, average ranking of 1.583, and F1-score of 0.798. It outperforms DP-Flex, HAI-AI, CtxPipe, and others. Its accuracy is only 0.2 percentage points below the near-optimal evolutionary strategy (ES, 0.851), while maintaining a runtime of just 94 seconds. It is marginally higher than HAI-AI but with substantially better accuracy and stability. Figure 4 further compares MAPPipe and CtxPipe on the Extend dataset. MAPPipe outperforms CtxPipe on 75.8% of datasets, confirming its robust generalization across diverse data distributions.

4.2 Demonstration Interface

As illustrated in Figure 3, MAPPipe provides an intuitive visual interactive that supports the full preprocessing workflow, from data ingestion to optimized pipeline export.

▹ **Input Configuration & Pipeline Recommendation**. Users begin by uploading a dataset, selecting a downstream model and evaluation metric, and optionally contributing new results to the knowledge base. MAPPipe then extracts meta-features and retrieves similar historical tasks to generate and visualize CandidatePipes, displaying both logical structure and physical instantiations.

▹ Next, **pipeline reconstruction** automatically prunes irrelevant modules (e.g., imputation for complete data) and merges redundant candidates. All intermediate artifacts, including removed modules, and the refined candidate set $\mathcal{L}$ are exposed for inspection.

▹ **Pipeline Search**. MAPPipe performs space pruning, pre-screens candidate logical pipelines from $\mathcal{L}$, retains the top-K high-potential ones, and constructs the heuristic scoring table $\mathcal{H}$. It also supports manual editing of $\mathcal{H}$ before search. Parallel MCTS is then launched to optimize pipeline instantiations, with progress and intermediate results shown in the interface. The final optimal pipeline is presented with its performance score after logs review.

Finally, users can export the preprocessed data, optimal pipeline, and complete execution log. MAPPipe achieves to empower data scientists to deploy accurate, interpretable, and reproducible data preprocessing workflows with minimal manual effort.

Acknowledgments

This work is supported by the National Key Research and Development Program of China (2024YFB3309601), the National Natural Science Foundation of China (NSFC) (62572151, 62232005, 52642071), and the Heilongjiang Provincial Young Talents Support Program for Science and Technology Innovation (2025QNTJ001).

References

[1] X. Ding, H. Wang, G. Li, H. Li, Y. Li, and Y. Liu. IoT data cleaning techniques: A survey. Intell. Converged Networks, 3(4):325–339, 2022.
[2] X. Ding, Z. Qian, H. Wang, S. Chen, Y. Tang, H. Su, H. Hu, and C. Wang. UniClean: A scalable data cleaning solution for mixed errors based on unified cleaners and optimized cleaning workflow. Proc. VLDB Endow., 18(11):4117–4130, 2025.
[3] X. Ding, Y. Liu, Z. Chen, H. Wang, C. Wang, and J. Wang. TARImpute: Task-aware auto-recommender system for missing value imputation algorithms with clustering case studies. Proc. VLDB Endow., 18(12):5343–5346, 2025.
[4] C. Chai, N. Tang, J. Fan, and Y. Luo. Demystifying artificial intelligence for data preparation. SIGMOD, 13–20, ACM, 2023.
[5] S. Siddiqi, R. Kern, and M. Boehm. SAGA: A scalable framework for optimizing data cleaning pipelines for machine learning applications. Proc. ACM Manag. Data, 218:1–218:26, ACM, 2023.
[6] Sibei Chen, Nan Tang, Ju Fan, et al: HAIPipe: Combining Human-generated and Machine-generated Pipelines for Data Preparation. SIGMOD 1(1): 91:1-91:26 (2023)
[7] H. Gao, S. Cai, T. T. A. Dinh, Z. Huang, and B. C. Ooi. CtxPipe: Context-aware data preparation pipeline construction for machine learning. Proc. ACM Manag. Data, 231:1–231:27, ACM, 2024.
[8] P. Li, Z. Chen, X. Chu, and K. Rong. DiffPrep: Differentiable data preprocessing pipeline search for learning over tabular data. Proc. ACM Manag. Data, 183:1–183:26, ACM, 2023.