TabClean: Scalable Tabular Data Cleaning via Reusable LLM-Synthesized Programs

Yibo Wang
Purdue University
West Lafayette, USA
wang7342@purdue.edu

Riteng Zhang
Purdue University
West Lafayette, USA
zhan5982@purdue.edu

Bharat Bhargava
Purdue University
West Lafayette, USA
bbshail@purdue.edu

Chunwei Liu
Purdue University
West Lafayette, USA
chunwei@purdue.edu

ABSTRACT

High-quality tabular data are essential for reliable machine learning. The real-world datasets are often corrupted by sensor failures and human errors. Traditional cleaning pipelines rely on handcrafted rules or narrowly scoped statistical models that are labor-intensive and difficult to generalize across domains and schemas. Large Language Models (LLMs) are a promising alternative, but existing LLM-based cleaners suffer from high token costs, slow inference, hallucination issues, and limited reusability.

We present TabClean, a cost-efficient and training-free tabular data cleaning system in which LLM agents synthesize reusable code for error detection and correction. The generated programs apply directly to any table with the same schema, making LLM usage a largely one-time rather than recurring cost. Across six benchmarks, TabClean achieves high precision and improves F1 over representative baselines on five datasets, all while substantially reducing runtime and API spend.

VLDB Workshop Reference Format:
Yibo Wang, Riteng Zhang, Bharat Bhargava, and Chunwei Liu. TabClean: Scalable Tabular Data Cleaning via Reusable LLM-Synthesized Programs. VLDB 2026 Workshop: Tabular Data Analysis (TaDA).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Wangyibo321/TabClean.

1 INTRODUCTION

Tabular datasets underpin decision-making across science, business, and the public sector and remain a dominant substrate for machine learning [9, 35]. In practice, however, tables are frequently contaminated by missing values, typos, inconsistent formats, unit mismatches, out-of-range measurements, and schema-dependent violations caused by sensor glitches and human workflows [5, 14]. It is widely estimated that data scientists spend most of their time cleaning and pre-processing raw data before analysis [1, 15, 23, 37]; at Massachusetts General Hospital, for instance, much of that effort goes to building data pipelines with extensive preparation and tuning [33]. Left unaddressed, such errors can propagate into downstream analytics, bias model training, and erode trust in automated pipelines [12, 16].

Proceedings of the VLDB Endowment. ISSN 2150-8097.

A large body of work [6, 10, 13, 24–26, 39] tackles data cleaning through handcrafted rules, integrity constraints, and statistical or learning-based detectors. Despite their successes, these approaches often require substantial domain expertise and iterative engineering. Rules must be written and maintained per dataset, constraints may be unknown or brittle under distribution shift, and specialized models often fail to transfer across schemas without retraining or feature redesign.

LLMs have already proven effective on data-management problems [3, 18–20]. For tabular data cleaning specifically, two paradigms dominate [43]. The first is prompt-based end-to-end cleaning [21, 28–30, 32, 41], which invokes an LLM API to directly identify and repair errors, sometimes paired with a knowledge base [28]. The second is task-adaptive fine-tuning [27, 40], which trains smaller language models on dataset-specific error distributions. Both scale poorly to large tables due to per-cell inference cost, hallucinate repairs under long contexts, and re-invoke the model for recurring errors. Fine-tuning also suffers a cold-start problem and generalizes poorly to new schemas. Code-generating frameworks such as LLMClean [4] and Cocoon [42] sidestep direct-modification bottlenecks but remain narrow: LLMClean targets only Ontological Functional Dependencies (OFDs), while Cocoon's reliance on regular expressions and SQL `CASE WHEN` clauses fails to scale and rarely transfers across tables sharing a schema.

This paper presents TabClean, a scalable, training-free data cleaning system that converts LLM reasoning into *executable cleaning programs*. A multi-agent pipeline synthesizes, tests, and iteratively refines Python repair code on a small human-annotated development set. Rather than hard-coding cleaning knowledge, agents consult a shared, schema-agnostic skills library to pick and validate repairs grounded in data evidence. The resulting programs are cached and reapplied to future tables with the same schema, making LLM usage a largely one-time, amortized cost. As new error patterns appear, TabClean incrementally extends its program library, enabling continual improvement without retraining. Experiments on six standard benchmarks show that TabClean matches or outperforms four state-of-the-art baselines, scales to tables too large for some of them, and keeps runtime and API cost stable.

2 METHODOLOGY

2.1 System Overview

Figure 1 illustrates the end-to-end workflow. **(1)** Given a raw table, the Dev-Set Sizer chooses how many rows a human should annotate for the development set. **(2)** The Dataset Profiler extracts profiling signals and structural statistics from the annotated development set. **(3)** The annotated table and profile are passed to the Diagnose

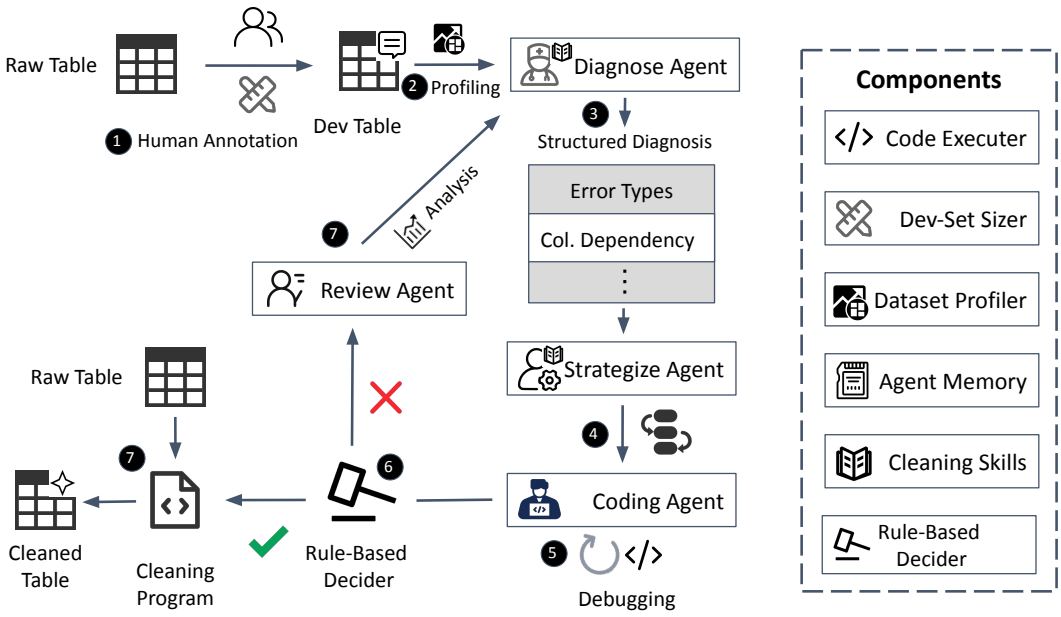

Figure 1: TabClean System Overview

Agent, which consults the shared Cleaning Skills library and emits a structured diagnosis of likely error patterns and repair opportunities. **(4)** The Strategize Agent turns the diagnosis into a cleaning plan that specifies repair operations and their order. **(5)** The Coding Agent translates the plan into an executable Python program run by the Code Executer. Runtime or syntax failures are routed to the Debugging module for repair and re-execution. **(6)** The Rule-Based Decider inspects the execution outcome and development-set metrics to decide whether to stop. **(7)** On stop, the best validated program is applied to the entire raw table to produce the cleaned output. Otherwise, the execution result, program, and intermediate artifacts flow to the Review Agent, which diagnoses failure causes and regressions and issues feedback for the next refinement round. Throughout, the Agent Memory buffer manages workflow context and routes only role-relevant information to each agent.

2.2 Core Components

Agentic Memory Buffer. Passing the full interaction trace to every agent is noisy and expensive [17, 34, 36]: outdated code, verbose logs, and stale reasoning distract agents as prompts grow across iterations. Our *agentic memory buffer* instead retains only high-value context and serves role-specific subsets, keeping durable evidence (e.g., profiles), active artifacts (e.g., the latest diagnosis), and compact cross-iteration feedback while discarding detailed past reasoning.

Cleaning Skills. A shared skills library collects general cleaning principles, structural heuristics, and reusable repair templates. Because these skills are technique-level rather than dataset-specific, they transfer across schemas and serve as the system's externalized knowledge base for repair reasoning.

Code Executer. The code executer runs candidate cleaning programs, detects syntax and runtime errors, and reports development-set metrics. Standardized I/O and isolated execution provide a reliable loop for evaluation, debugging, and rollback.

Adaptive Dev-Set Sizer. To bound annotation cost, this component decides how many rows to label. On randomly ordered rows, it stops once coverage is sufficient: enough rows are labeled, each error-bearing column has enough error instances, and total observed errors support stable development-set F1. To keep annotation sublinear in table size, the budget is bounded by a size-adaptive cap on the number of labeled rows: given N total rows, it labels at most $\min(500, 0.15N)$ rows for $N \leq 5{,}000$, $\min(700, 0.05N)$ for $5{,}000 < N \leq 50{,}000$, and $\min(1{,}000, 0.01N)$ for $N > 50{,}000$, so the labeled fraction shrinks as tables grow and excessive labeling on large tables is avoided.

Dataset Profiler. Before any LLM call, the profiler computes per-column statistics (cardinality, value distributions) to surface unusual formats, rare tokens, and missing-value hotspots. It also detects cross-row patterns such as low-cardinality grouping keys and nearly constant attributes within groups, which enable consistency-based repairs. It then produces a data preview and loads relevant skills into the agentic memory buffer.

Rule-based Decider. The rule-based decider applies deterministic rules over execution status and development-set performance to choose the next transition. After each evaluation, it checks whether the target F1 is reached, the iteration budget is exhausted, or the best F1 has not improved within a patience window. If so, the pipeline finalizes the best validated program. Otherwise, control passes to the review agent.

2.3 Specialized LLM Agents

Diagnose Agent. The diagnose agent takes as input the profile, development-set samples, and skills library. It identifies column-wise error types, distinguishes sporadic from systematic corruption, and estimates the in-table evidence available for repair. Its structured diagnosis lists error labels, column dependencies, and candidate repairs categorized as *deterministic*, *evidence-based*, or *uncertain*. This classification guides planning and curbs overly aggressive modifications.

Strategize Agent. The strategize agent converts the diagnosis into a repair plan specifying target columns, selected skills, execution order, and guard conditions. The initial iteration employs a conservative *deterministic-first* strategy that corrects only high-confidence errors, thereby establishing a high-precision baseline. Subsequent iterations are feedback-driven: using the latest evaluation summary and the best prior strategy, the agent preserves stable transformations, revises failing ones, and adds only a few new evidence-based repairs per round. This incremental protocol reduces oscillation and makes performance changes easier to attribute.

Coding Agent. The coding agent translates the strategy into an executable Python cleaning program. To ensure generalizability and suppress false positives, the code must preserve table shape, avoid row-level hardcoding, and enforce strict evidentiary guards. The agent edits incrementally, modifying the best previously validated program to reduce regressions. Programs are executed on the development set under a log-driven debugger. Successful outputs are scored cell-by-cell against the ground truth for precision, recall, and F1, and cross-iteration edit traces are summarized into per-column feedback for subsequent rounds.

Review Agent. The review agent jointly analyzes the current strategy, synthesized code, and evaluation output. Rather than proposing a new pipeline, it explains why the iteration succeeded or failed and flags the highest-impact revisions for the next round, distinguishing strategy errors, code-implementation errors, and insufficient data evidence while marking transformations to preserve. This structured root-cause analysis closes the refinement loop and hands actionable guidance to the next strategize step.

3 EXPERIMENTS

Our experiments answer three questions: (i) how TabClean compares with strong rule-based, learning-based, and LLM-based baselines in cleaning quality, (ii) whether synthesizing reusable cleaning code leads to practical end-to-end runtime, and (iii) how much LLM usage and monetary cost are required in practice.

3.1 Experimental Setup

Datasets. Table 1 summarizes the six standard benchmarks, spanning healthcare, aviation, tax records, literature screening, and movie metadata. They cover both small and large tables and heterogeneous error patterns, from typos and format inconsistencies to functional-dependency and missing-value errors.

Baselines. We compare TabClean against four representative baselines. **Holistic** [6] is a probabilistic data repairing system that combines integrity constraints with statistical signals and performs inference to impute/correct values. **Baran** [24] is a semi-supervised correction engine that learns value-repair transformations from data and (optionally) a small set of labeled corrections. **Cocoon** [42] uses an off-the-shelf LLM to synthesize executable cleaning logic (e.g., regex/SQL/Python) and validates it against the dataset in a feedback loop. **IterClean** [30] is an iterative LLM-based pipeline that alternates between error detection/verification and repair over multiple rounds.

Table 1: Datasets used in the experiments.

Dataset	Rows	Columns	Error Cells	Dev Rows
hospital [6, 38]	1,000	19	509	150
flight [22]	2,376	7	9,504	110
beers [11]	2,410	10	4,765	240
tax [2, 8]	200,000	15	121,219	150
rayyan [31]	1,000	11	960	150
movies [7]	7,390	17	7,675	369

Metrics. We report precision, recall, and F1 for cleaning quality, wall-clock runtime for end-to-end efficiency, and token/cost statistics for LLM-based systems when available. Cleaning quality is evaluated by exact cell-level matching over the common columns of the dirty, cleaned, and ground-truth tables: dirty cells repaired to the ground-truth value count as true positives, missed dirty cells as false negatives, and originally clean cells changed by a system as false positives.

Model Selection. We employ gpt-5-mini for both Cocoon and IterClean, as well as for the diagnose and review agents in TabClean. We use gpt-5.3-codex for the coding agent and gpt-5.2 for the strategize agent.

3.2 Cleaning Effectiveness

Table 2 shows that TabClean attains the best F1 on five of the six benchmarks, trailing only IterClean on *hospital*. It obtains the highest recall on those five datasets and the strongest F1, reflecting our design: the pipeline first applies deterministic, evidence-backed repairs and escalates to aggressive transformations only after development-set validation, suppressing false positives.

The largest gains appear on *flight*, *tax*, *rayyan*, and *movies*, where LLM reasoning compiled into executable code can be applied uniformly at scale. *flight*'s systematic time-format errors let TabClean lift F1 over Baran from 0.32 to 0.81, and *tax*'s regular formatting and canonicalization errors let a single synthesized program reach 0.99 F1. On *rayyan* and *movies*, TabClean still leads by combining structural repairs with cautious normalization, though recall trails *tax* because remaining errors lie in weak-evidence fields such as author lists, pagination, free-form descriptions, and identifier-like attributes. On *hospital*, IterClean edges out TabClean in F1 (0.95 vs. 0.87) by recovering more dirty cells, though TabClean matches its perfect precision at a fraction of the LLM cost.

Several baseline entries remain at 0.00 because the evaluation gives credit only for exact matches to the benchmark ground truth. Holistic makes no repairs on *flight*, *movies*, and *rayyan*; Cocoon and IterClean sometimes make plausible normalizations on zero-score datasets, but none of those repaired dirty cells exactly match the expected clean values, yielding no true positives.

3.3 Efficiency and Cost

TabClean remains practical despite multiple cooperating agents (Table 3): LLM calls concentrate in diagnosis, planning, and code

Table 2: Data cleaning performance (Precision, Recall, and F-1) across different benchmarks. Higher is better.

System	hospital			flight			beers			tax			rayyan			movies		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Holistic	0.98	0.22	0.36	0.00	0.00	0.00	0.78	0.02	0.04	0.75	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Baran	1.00	0.54	0.70	1.00	0.19	0.32	1.00	0.78	0.88	1.00	0.73	0.84	1.00	0.22	0.36	1.00	0.67	0.80
Cocoon	1.00	0.42	0.59	0.00	0.00	0.00	1.00	0.03	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
IterClean	1.00	0.91	**0.95**	0.00	0.00	0.00	1.00	0.60	0.75	0.03	0.01	0.02	0.01	0.00	0.00	0.21	0.05	0.08
TabClean	1.00	0.77	0.87	1.00	0.68	**0.81**	1.00	0.89	**0.94**	1.00	0.98	**0.99**	1.00	0.85	**0.92**	0.99	0.83	**0.91**

Table 3: End-to-end wall-clock runtime on each benchmark. Lower is better. Units: s=seconds, m=minutes, h=hours.

System	hospital	flight	beers	tax	rayyan	movies
Holistic	23s	25s	26s	10.6h	<1s	<1s
Baran	8.7m	8.7m	8.1m	89.8h	3.2m	80.7m
Cocoon	15.6m	3.2m	7.2m	16.9m	11.8m	17.9m
IterClean	2.1h	5.2h	58.3h	140.9h	71.0m	5.92h
TabClean	40.1m	13.4m	9.32m	4.42m	32.4m	16.0m

Table 4: LLM token usage (thousands) and API cost for LLM-based systems. Lower is better.

System	Dataset	Input (K)	Output (K)	Cost ($)
Cocoon	hospital	14	78	0.16
	beers	8	32	0.07
	flight	5	14	0.02
	tax	20.1	71.8	0.15
	movies	21	65	0.13
	rayyan	12	50	0.10
IterClean	hospital	3,727	3,571	8.07
	flight	1,303	1,553	3.43
	beers	7,560	13,096	28.08
	tax	87,251	36,643	93.49
	movies	13,807	1,633	6.72
	rayyan	1,008	303	0.85
TabClean	hospital	520	143	1.98
	beers	196	37	0.49
	tax	25	8	0.08
	flight	337	53	0.79
	movies	298	66	0.79
	rayyan	599	138	1.66

synthesis, after which cleaning runs as ordinary Python rather than invoking an LLM per row. Compared with Baran, TabClean cuts *tax* runtime from 89.8 hours to 4.42 minutes while lifting F1 from 0.84 to 0.99. It also beats IterClean on *beers* (9.32 minutes vs. 58.3 hours) at higher F1. Holistic is faster on several small datasets, but its recall is too low to offer comparable quality.

The runtime profile also reflects dataset difficulty. *tax* finishes after one iteration because its errors are highly regular, whereas *hospital* and *rayyan* require more refinement rounds and therefore longer end-to-end time. Even so, total runtime stays in the tens of minutes—practical for offline use and far better than repeatedly prompting large tables.

Table 4 highlights the economic advantage of compiling LLM reasoning into reusable code. TabClean stays below $2 on every reported dataset and drops to $0.08 on *tax*. On *flight* and *beers*, TabClean uses only 23% and 1.7% of IterClean's API cost respectively while achieving higher F1. The gap should widen further on large datasets like *tax*, since IterClean repeatedly feeds whole tables to the LLM across rounds, whereas TabClean pays the reasoning cost once to synthesize a reusable program and then executes it cheaply at scale. As the same schema reappears or the dataset grows, the marginal LLM cost of applying the resulting program is zero.

Table 5: Required human input per dataset.

System	Required Human Input	Avg. Time
Holistic	Denial constraints / FDs (∼5–10 rules)	1–2 h
Baran	Cell labels on 20 tuples (active learning)	5–10 min
Cocoon	None	0
IterClean	Cell labels on 5 seed tuples	2–5 min
TabClean	Cell labels on dev set (∼100–400 rows)	1–2 h

A third dimension is human labor. Table 5 summarizes what each system asks of a user per dataset. TabClean requires more up-front annotation than Baran or IterClean because it labels a small development set rather than a few seed tuples. However, this cost is *amortized*: the synthesized program is reused on every future table with the same schema without further annotation or LLM calls. Baran's labels, by contrast, are consumed by a dataset-specific model and do not transfer. Cocoon and IterClean instead shift the labor cost to recurring per-table LLM spend, with IterClean alone charging $28 on *beers* (Table 4). Holistic incurs comparable up-front effort but requires a SQL-literate expert to author denial constraints, whereas cell labeling needs no formal-logic expertise.

4 CONCLUSION

We presented TabClean, a training-free tabular data cleaning system that reframes LLM-based cleaning as a *program synthesis* problem. Rather than invoking an LLM per row or per table pass, a multi-agent pipeline produces a reusable Python cleaning program from a small annotated development set, validates it against ground-truth labels, and applies it to every future table sharing the same schema, reducing marginal LLM cost to zero after the first run. Experiments on six benchmarks show that TabClean matches or exceeds state-of-the-art cleaning quality while substantially reducing end-to-end runtime and LLM cost compared with prior LLM-based approaches.

REFERENCES

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

[2] Patricia C Arocena, Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing up with BART: error generation for evaluating data-cleaning algorithms. *Proceedings of the VLDB Endowment* 9, 2 (2015), 36–47.

[3] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. [n.d.]. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. ([n. d.]).

[4] Fabian Biester, Mohamed Abdelaal, and Daniel Del Gaudio. 2024. Llmclean: Context-aware tabular data cleaning via llm-generated ofds. In *European Conference on Advances in Databases and Information Systems*. Springer, 68–78.

[5] Xu Chu, Ihab F Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 international conference on management of data*. 2201–2206.

[6] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 458–469.

[7] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n.d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[8] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2, Article 6 (June 2008), 48 pages. https://doi.org/10.1145/1366102.1366103

[9] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '22)*. Curran Associates Inc., Red Hook, NY, USA, Article 37, 14 pages.

[10] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 international conference on management of data*. 829–846.

[11] J. N. Hould. n.d.. Craft beers dataset. Kaggle dataset. https://www.kaggle.com/datasets/nickhould/craft-cans Accessed: 2026-04-22.

[12] Zezhou Huang, Pavan Kalyan Damalapati, and Eugene Wu. 2023. Data ambiguity strikes back: How documentation improves gpt's text-to-sql. *arXiv preprint arXiv:2310.18742* (2023).

[13] Ihab F Ilyas and Xu Chu. 2015. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases* 5, 4 (2015), 281–393.

[14] Ihab F Ilyas and Xu Chu. 2019. *Data Cleaning*. Morgan & Claypool Publishers.

[15] Ihab F Ilyas and Xu Chu. 2019. Data Cleaning is a Machine Learning Problem that Needs Data Systems Help! ACM SIGMOD Blog. http://wp.sigmod.org/?p=2288

[16] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.* 9, 12 (2016), 948–959.

[17] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. 2024. Babilong: Testing the limits of llms with long context reasoning-in-a-haystack. *Advances in Neural Information Processing Systems* 37 (2024), 106519–106554.

[18] Jiale Lao and Immanuel Trummer. 2026. Sqlbarber: A system leveraging large language models to generate customized and realistic sql workloads. *Proceedings of the ACM on Management of Data* 4, 1 (SIGMOD (2026), 1–27.

[19] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. [n.d.]. GPTuner: A Manual-Reading Database Tuning System via GPT-Guided Bayesian Optimization. ([n. d.]).

[20] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4213–4216. https://doi.org/10.14778/3685800.3685838

[21] Lan Li, Liri Fang, and Vetle I Torvik. 2024. Autodcworkflow: Llm-based data cleaning workflow auto-generation and benchmark. *arXiv preprint arXiv:2412.06724* (2024).

[22] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2012. Truth finding on the deep web: is the problem solved? *Proc. VLDB Endow.* 6, 2 (Dec. 2012), 97–108. https://doi.org/10.14778/2535568.2448943

[23] Chunwei Liu, Enrique Noriega-Atala, Adarsh Pyarelal, Clayton T Morrison, and Mike Cafarella. 2025. Variable extraction for model recovery in scientific literature. In *Proceedings of the 1st Workshop on AI and Scientific Discovery: Directions and Opportunities*. 1–12.

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

[25] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*. 865–882.

[26] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: a database approach for statistical inference and data cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 75–86.

[27] Pavitra Mehra et al. 2024. Leveraging structured and unstructured data for tabular data cleaning. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 5765–5768.

[28] Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mohamed Y Eltabakh, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Data Cleaning Using Foundation Models and Data Lakes. *Proceedings of the VLDB Endowment* 17 (2024), 4421–4424.

[29] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).

[30] Wei Ni, Kaihang Zhang, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, and Jianwei Yin. 2024. Iterclean: An iterative data cleaning framework with large language models. In *Proceedings of the ACM Turing Award Celebration Conference-China 2024*. 100–105.

[31] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. 2016. Rayyan—a web and mobile app for systematic reviews. *Systematic reviews* 5, 1 (2016), 210.

[32] Danrui Qi, Zhengjie Miao, and Jiannan Wang. 2024. Cleanagent: Automating data standardization with llm-based agents. *arXiv preprint arXiv:2403.08291* (2024).

[33] El Kindi Rezig, Mourad Ouzzani, Ahmed K Elmagarmid, Walid G Aref, and Michael Stonebraker. 2019. Data Civilizer 2.0: a Holistic Framework for Data Preparation and Analytics. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1954–1957.

[34] Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*. PMLR, 31210–31227.

[35] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information fusion* 81 (2022), 84–90.

[36] Yuyang Song, Hanxu Yan, Jiale Lao, Yibo Wang, Yufei Li, Yuanchun Zhou, Jianguo Wang, and Mingjie Tang. 2026. QUITE: A Query Rewrite System Beyond Rules with LLM Agents. arXiv:2506.07675 [cs.DB] https://arxiv.org/abs/2506.07675

[37] Tamr. 2017. How to Clean Noisy and Erroneous Big Data Using Machine Learning. Tamr Blog. https://www.tamr.com/blog/how-to-clean-noisy-and-erroneous-big-data-using-machine-learning/

[38] U.S. Department of Health & Human Services. 2012. Hospital Compare. Web site. http://www.hospitalcompare.hhs.gov/

[39] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. 2013. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 553–564.

[40] Mengyi Yan, Yaoshu Wang, Yue Wang, Xiaoye Miao, and Jianxin Li. 2024. Gidcl: A graph-enhanced interpretable data cleaning framework with large language models. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–29.

[41] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Large language models as data preprocessors. *arXiv preprint arXiv:2308.16361* (2023).

[42] Shuo Zhang, Zezhou Huang, and Eugene Wu. 2025. Data cleaning using large language models. In *2025 IEEE 41st International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 28–32.

[43] Wei Zhou, Jun Zhou, Haoyu Wang, Zhenghao Li, Qikang He, Shaokun Han, Guoliang Li, Xuanhe Zhou, Yeye He, Chunwei Liu, et al. 2026. Can LLMs Clean Up Your Mess? A Survey of Application-Ready Data Preparation with LLMs. *arXiv preprint arXiv:2601.17058* (2026).