



PrepBench: How Far Are We from Natural-Language-Driven Data Preparation?

Jingzhe Xu, Rui Wang, Jiannan Wang, Guoliang Li

Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing, China
j1n9zhe@gmail.com, akane@hust.edu.cn, jnwang@tsinghua.edu.cn, liguoliang@tsinghua.edu.cn

ABSTRACT

Data preparation is a central and time-consuming stage in data analysis workflows. Traditionally, commercial tools have relied on graphical user interfaces (GUIs) to simplify data preparation, allowing users to define transformations through visual operators and workflows. Recent advances in large language models (LLMs) raise the possibility of a paradigm shift toward natural language (NL)-driven data preparation, in which users can specify preparation intents in NL directly. However, it remains unclear how far current LLM-based agents are from this paradigm shift in practice. Existing code generation benchmarks do not capture key characteristics of data preparation, including ambiguous user intents, imperfect real-world data, and the need to translate code into interpretable workflows for validation. To bridge this gap, we present PrepBench, a benchmark designed to evaluate NL-driven data preparation along three core capabilities: interactive disambiguation, prep-code generation, and code-to-workflow translation. We crawl data from the Preppin' Data Challenges, and then extend it into a systematically designed benchmark. The benchmark covers diverse domains, and each task involves 3 to 18 data preparation steps. Nearly half of the tasks require over 100 lines of Python code, and the longest solutions approach 300 lines. Our evaluation shows that, despite recent progress, realizing this paradigm shift remains challenging for state-of-the-art LLMs. PrepBench provides a principled benchmark for measuring this gap and helps identify key challenges toward realizing NL-driven data preparation.

PVLDB Reference Format:

Jingzhe Xu, Rui Wang, Jiannan Wang, Guoliang Li. PrepBench: How Far Are We from Natural-Language-Driven Data Preparation? PVLDB, 19(10): 2866 - 2879, 2026.
doi:10.14778/3828612.3828638

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/TsinghuaDatabaseGroup/prepbench>.

1 INTRODUCTION

Data preparation has long been considered a major bottleneck in data analysis [23, 40]. In practice, analysts often spend considerable time cleaning, transforming, and reshaping data before any downstream analysis can begin [1]. To address this issue, the industry has adopted commercial data preparation tools such as Tableau Prep [48] and SAS Data Preparation [44]. These tools allow users to construct data preparation workflows through graphical user interfaces (GUIs) by chaining visual operators (e.g., filter, join, aggregate,

pivot). We refer to this paradigm as *GUI-driven data preparation*. Figure 1(a) shows an example in which two raw tables need to be transformed into a prepared form. Figure 1(b) shows one workflow constructed by analysts to perform this data preparation.

While effective, this paradigm requires users to translate natural language intents into GUI operations, which can be indirect and error-prone. Moreover, GUI-based tools still impose a non-trivial learning curve despite being more accessible than programming. Recent advances in large language models (LLMs) offer a possible way to close this gap [11, 33]. As shown in Figure 1(c), users can upload data to an LLM-based agent (e.g., ChatGPT [34]) and directly express their data preparation intent in natural language (e.g., "Align the schemas, merge the two tables, and normalize date formats"). ChatGPT then generates executable code and produces the prepared data. We refer to this interaction model as *NL-driven data preparation*. This paradigm can significantly lower the barrier to data preparation and make data analysis more accessible to a broader range of users.

As a result, one natural question is: *how far are we from NL-driven data preparation?* To answer this question, we introduce PREPBENCH, a new benchmark for NL-driven data preparation. Our goal is not only to conduct an end-to-end evaluation, but also to understand the fundamental capabilities that LLM-based systems must support to make NL-driven data preparation practical. Specifically, PREPBENCH evaluates systems along three capabilities: (1) *Interactive Disambiguation*, which tests whether systems can detect and resolve ambiguous intents through interaction; (2) *Prep-Code Generation*, which tests whether systems can generate correct preparation code from user requests; and (3) *Code-to-Workflow*, which tests whether generated code can be mapped to GUI workflows.

These capabilities are motivated by practical requirements, but have not been evaluated together in existing benchmarks. Prior benchmarks have studied NL interfaces for various data-related tasks, including NL-to-SQL [9, 21], NL-to-Code [26, 52], NL-to-Pipeline [12, 22], and Code-to-Workflow understanding [30]. However, as summarized in Table 1, these benchmarks either address different problem settings or focus on individual capabilities in isolation (see Section 2.3). To the best of our knowledge, PREPBENCH is the first to systematically evaluate all three capabilities for NL-driven data preparation.

Evaluating these capabilities requires a benchmark based on realistic tasks. PREPBENCH is constructed by crawling data from the Preppin' Data Challenges [37], a weekly series that provides practical data preparation tasks to Tableau Prep users. We then extend the collected data into a systematically designed benchmark. PREPBENCH consists of 306 tasks over 829 input tables and covers 32 application domains, including retail, sports, education, finance, and transportation. The tasks are diverse and non-trivial: each task involves 3 to 18 data preparation steps, nearly half require more than 100 lines of Python code, and the longest solutions approach 300 lines. Although derived from Tableau Prep training materials, the operations and workflows in PREPBENCH are generic and supported

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 10, 2026 ISSN 2150-8097.
doi:10.14778/3828612.3828638

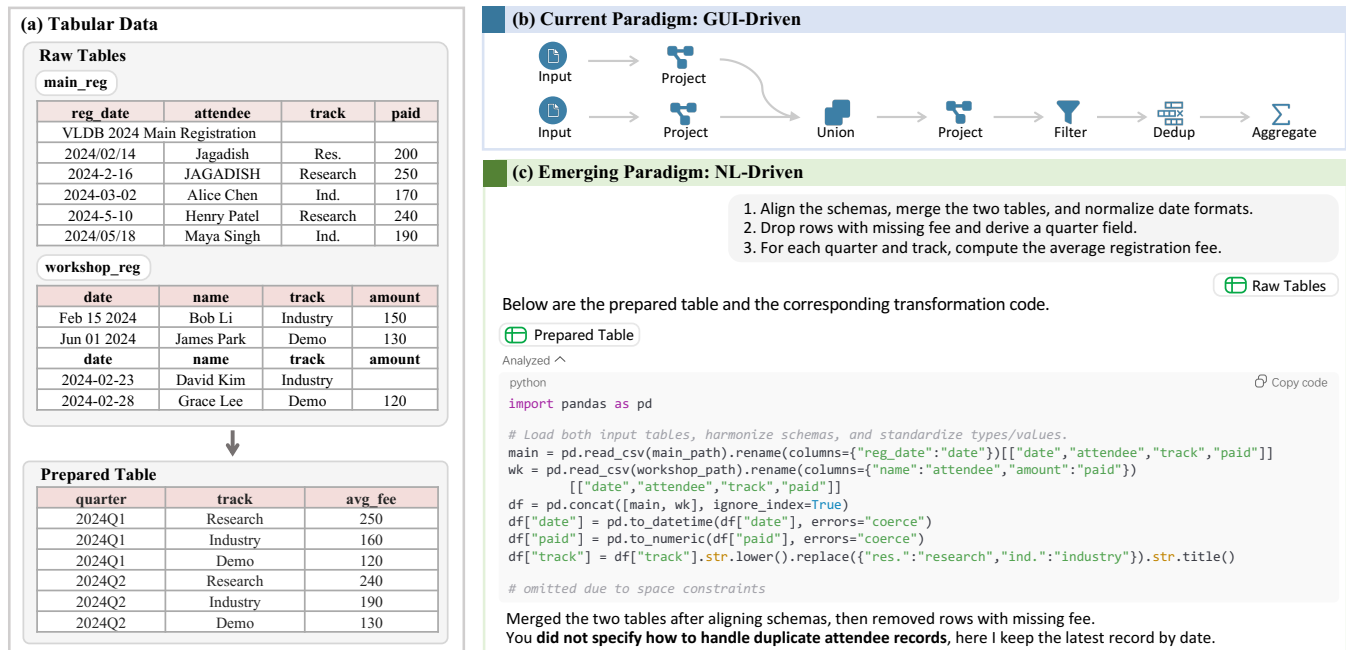


Figure 1: (a) Unprepared and prepared tables: Raw data from two tables alongside the resulting prepared data; (b) GUI-driven data preparation: Users define a preparation workflow through visual operators (e.g., Project, Union, Filter); (c) NL-driven data preparation: Users specify tasks in natural language, and the system generates the corresponding code.

by a wide range of commercial data preparation systems [15]. These characteristics make PREPBENCH well-suited for evaluating realistic tabular data preparation tasks.

It is worth noting that Preppin' Data alone cannot be directly used as a benchmark to evaluate our three target capabilities. There are two challenges that need to be addressed. First, Preppin' Data lacks executable ground-truth code and unambiguous task descriptions—both essential but expensive to create manually at scale. We address this by designing an agent-based pipeline that automatically generates ground-truth code and derives unambiguous task descriptions from it. This approach enables scalable benchmark construction with reduced manual effort.

The second challenge lies in evaluating interactive disambiguation. In real-world data preparation scenarios, NL requests are often ambiguous, requiring systems to identify missing information, ask clarification questions, and refine solutions based on user feedback. As part of benchmark construction, we build a disambiguation knowledge base that records ambiguity cases in the original requests together with their corresponding resolutions. We further design a *user simulator* that automatically responds to clarification questions using this knowledge base. This setup simulates realistic user interaction, where a system detects ambiguity, requests clarification, and receives additional input to complete the task.

We conduct comprehensive experiments on PREPBENCH across ten state-of-the-art proprietary and open-weight models, evaluating both end-to-end performance and each core capability. The results reveal several important insights. First, although current LLMs show strong code generation ability, the best model (GPT-5.1-Codex) achieves only 54.9% accuracy on prep-code generation. Ambiguity has a large impact: removing it improves accuracy to 85.3%. Second, interactive disambiguation helps, but current LLMs frequently ask incomplete or ineffective clarification questions, which limits its benefits. Third, code-to-workflow translation remains a major bottleneck: even when correct prep-code is generated, the resulting workflows often do not match the intended data preparation steps. Finally, higher model cost does not consistently predict better

performance: a lightweight model (Gemini 3 Flash) nearly matches the best model's accuracy at less than one-fifth of the cost. Overall, these results indicate that NL-driven data preparation is promising but remains challenging in practice. They also demonstrate the importance of PREPBENCH as a benchmark for systematically revealing limitations that are not captured by existing evaluations.

In summary, the paper makes the following contributions:

- We present PREPBENCH, a new benchmark that systematically evaluates NL-driven data preparation across three core capabilities: interactive disambiguation, prep-code generation, and code-to-workflow translation.
- We construct PREPBENCH by extending the Preppin' Data Challenges with executable ground-truth data, a disambiguation knowledge base, and unambiguous task specifications.
- We design three execution modes that enable both end-to-end evaluation and fine-grained analysis of individual capabilities.
- We conduct a comprehensive evaluation of LLM-based agents on PREPBENCH, identifying key limitations and highlighting opportunities for future research.

Section 2 defines NL-driven data preparation. Sections 3 and 4 then detail the construction of PREPBENCH and its corresponding execution modes. Our experimental findings and subsequent research opportunities are discussed in Sections 5 and 6, respectively. Finally, we review related work in Section 7 and conclude in Section 8.

2 NL-DRIVEN DATA PREPARATION

We first define NL-driven data preparation (Section 2.1), then present three required capabilities (Section 2.2), and finally position PREPBENCH against existing benchmarks (Section 2.3).

2.1 Problem Statement

Given a set of raw tables D and an NL request q , the goal of NL-driven data preparation is to transform D into one or multiple output tables according to the specifications in q . In this paper, *data preparation* refers to structural and syntactic table transformations,

such as joining, reshaping, filtering, and formatting (e.g., parsing dates and normalizing strings). We do not target semantic data repair that cannot be determined from D and q alone, such as inferring missing values and correcting out-of-date values.

To illustrate, Figure 1(c) depicts a typical NL-driven data preparation scenario. The input D comprises two raw tables, `main_reg` and `workshop_reg`, which record registration data for a main conference and its workshops. These tables exhibit structural heterogeneity (e.g., schema mismatches like attendee vs. name) and syntactic inconsistencies (e.g., mixed date formats). The user provides an NL request q —"Align the schemas... Drop rows... compute the average registration fee". Guided by q , the system transforms D into the target output by generating executable code that unifies schemas and standardizes values (e.g., normalizing "Ind." to "Industry").

2.2 Three Fundamental Capabilities

Accurate code generation alone does not make NL-driven data preparation practical. This is because: (1) user requests are often ambiguous and can be interpreted in multiple ways; (2) real datasets are often too large to examine thoroughly, and the code written for the sample may miss important data issues; and (3) the generated code is hard for users, who are familiar with GUI-based workflows, to understand and check. Therefore, a practical NL-driven data preparation system requires the following three capabilities.

Interactive Disambiguation. Consider the example in Figure 1(c). The NL request does not specify how to handle duplicate attendees in the `main_reg` table (e.g., "keep the latest record" vs. "keep the one with the highest fee"). A wrong guess here may introduce silent errors into the data. The interactive disambiguation capability addresses this by detecting such ambiguities and resolving them through user clarification instead of default assumptions.

Prep-Code Generation. After disambiguation, the system generates executable prep-code. In the above example, a user might only see standard dates (e.g., "2024/02/14") in the sample of the `main_reg` table, but row 10,000 might contain a different format ("February 15 2024"). The prep-code generated solely based on the sample will fail. The prep-code generation capability addresses this by profiling the full dataset to discover hidden irregularities and generating robust code that handles all variations.

Code-to-Workflow. After generating prep-code and prepared tables, users often need to verify the transformation logic. However, verifying raw code can be challenging; for instance, confirming that a complex Pandas script correctly dropped "missing fee" rows may be difficult for non-experts. In contrast, validating a visual "Filter" node in a workflow diagram is typically more intuitive. The *workflow translation* capability addresses this by translating generated code into an interpretable workflow (as shown in Figure 1(b)), which can be run on the input tables to facilitate verification.

2.3 Comparison with Existing Benchmarks

Table 1 compares PREPBENCH with relevant benchmarks.

Benchmarks for Interactive Disambiguation. Recent benchmarks such as BIRD-INTERACT [21] and PRACTIQ [9] introduce interaction into NL-to-SQL tasks. BIRD-INTERACT and PRACTIQ both evaluate ambiguity handling in NL-to-SQL. BIRD-INTERACT lets the system ask follow-up questions during evaluation, while PRACTIQ provides a fixed clarification dialogue for each ambiguous task. Both benchmarks use clarification to recover the intended SQL query. In contrast, PREPBENCH evaluates disambiguation for data

Table 1: Comparison with representative benchmarks (✓: evaluated; ✗: not targeted; parentheses specify the setting).

Benchmark	Task	Capability		
		Interactive Disambig.	Prep-Code Generation	Workflow Translation
BIRD-INTERACT	NL2SQL	✓(SQL)	✗	✗
PRACTIQ	NL2SQL	✓(SQL)	✗	✗
DS-1000	NL2Code	✗	✓(cell)	✗
ARCADE	NL2Code	✗	✓(cell)	✗
PARROT	NL2Pipeline	✗	✓(schema + cell)	✗
ELT-Bench	NL2Pipeline	✗	✓(schema)	✗
CRABS	Code2Workflow	✗	✗	✓(descriptive)
PREPBENCH	NL2Code Code2Workflow	✓(prep-code)	✓(schema + cell)	✓(descriptive + executable)

preparation, where clarification concerns how raw tables should be cleaned, transformed, and combined, and the resolved intent is used to generate prep-code and a workflow.

Benchmarks for Prep-Code Generation. Benchmarks such as DS-1000 [26] and ARCADE [52] evaluate NL-to-code generation for data analysis. DS-1000 tasks are typically short snippets, with 3.6 lines on average, while ARCADE generates code for the next cell in a data science notebook. These benchmarks may involve cell-level irregularities, but they do not evaluate schema-level irregularities which are common in real-world data preparation. PARROT [12] generates executable pipelines rather than code snippets and covers both schema-level and cell-level irregularities, but it does not evaluate clarification for ambiguous requests or code-to-workflow translation. ELT-Bench [22] evaluates end-to-end ELT pipeline construction with schema-level irregularities, but does not target cell-level dirty values. In contrast, PREPBENCH evaluates prep-code generation for multi-step data preparation workflows with both schema-level and cell-level irregularities.

Benchmarks for Code-to-Workflow. CRABS [30] extracts dependency graphs from notebook code to support interpretability, capturing information flow and cell execution dependencies. These graphs provide descriptive workflow representations that help users reason about code. PREPBENCH builds on this insight but focuses on NL-driven data preparation. In this setting, workflows are evaluated as both descriptive and executable artifacts, supporting inspection as well as validation against data.

3 PREPBENCH

This section presents PREPBENCH. We first describe the data source (Section 3.1), and then discuss how to extend it into PREPBENCH (Section 3.2). We finally report benchmark statistics (Section 3.3).

3.1 Data Source

Preppin' Data [37] is a collection of data preparation challenges based on tabular datasets. Each challenge requires applying multiple preparation steps to turn raw data into analysis-ready data. Since 2019, the collection has grown to over 300 challenges, covering a wide range of practical tasks.

Preppin' Data has three characteristics that make it well suited as the basis for our benchmark. First, it offers broad coverage: the challenges span 32 application domains, including retail, sports, education, finance, and transportation, capturing a wide range of data preparation scenarios. Second, the tasks are non-trivial. They typically require multi-step transformations over irregular data and are substantially more complex than small, isolated operations. Third, the tasks are realistic. They are derived from practical data preparation problems and reflect the skills required in real analytics

workflows. As a result, performance on these tasks is a meaningful indicator of real-world data preparation capability.

3.2 Benchmark Construction

We convert each Preppin' Data weekly challenge into a benchmark task through four stages. We (i) parse the challenge pages and clean the extracted task data and descriptions, (ii) generate executable code that reproduces the ground-truth output, (iii) use the verified code as a reference to disambiguate the request, and (iv) translate the code into a workflow.

Task Parsing and Cleaning. Each challenge is published as a blog post containing a scenario, step-by-step instructions, and the expected output schema, along with downloadable input and output tables. We parse each page to extract three assets: *original request* (scenario, steps, and output schema), *input tables*, and *ground-truth output*. *Original request* is cleaned by removing non-task text such as announcements, navigation links, and author information. We exclude one out-of-scope challenge (2024 Week 49), which is not a table preparation task.

Ground-Truth Code Generation. If we were only interested in evaluating end-to-end accuracy, generating ground-truth code would not be necessary. By using the ground-truth output, we could directly compare it to the system-generated output to determine if the data preparation code is correct. However, since we aim to evaluate the three core capabilities independently, as will be seen in later sections, ground-truth code is essential for each capability.

We first consider two approaches to generate ground-truth code, but both have limitations. First, although Preppin' Data provides solution posts, they are expressed as Tableau Prep workflows stored in a proprietary format (.tflx) and depend on Tableau's closed-source runtime. This makes it difficult to reliably translate them into ground-truth code. Second, manually writing ground-truth code is also challenging. These weekly challenges are often complex, and producing correct code is time-consuming. Moreover, NL requests are frequently ambiguous, requiring repeated interpretation, code revision, and output comparison to ensure correctness.

We adopt an agent-based pipeline to generate *ground-truth code*. The agent performs two tasks: profiling tables to summarize their characteristics, and synthesizing code based on the request. The profiling process is described in Section 4.2. Unlike a standard NL-driven data preparation task, here the ground-truth output is available. This gives the agent direct feedback on whether the generated code is correct and allows it to revise the code automatically.

Specifically, our agent pipeline proceeds in two phases:

- (1) **Initial Phase:** The agent begins by reading the original request, profiling the relevant tables, and generating the code. The generated code is then executed, and its output is compared to the ground-truth output.
- (2) **Iterative Phase:** If the generated output does not match the ground-truth output, the agent analyzes the differences (such as row mismatches, missing keys, or value differences). Using this feedback, the agent revises the code and executes it again. This process continues until the output matches the expected result or the iteration limit is reached.

This method works for 57.3% of tasks, showing that it can generate the right code most of the time. For the remaining tasks, the feedback helps narrow down the possible issues, and we complete the code through manual debugging.

Disambiguation Assets Construction. To evaluate interactive disambiguation, we need to figure out where an NL request can be

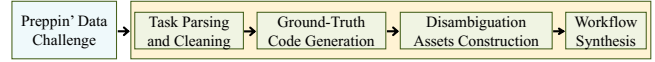


Figure 2: An overview of PREPBENCH construction.

interpreted in multiple reasonable ways and how such ambiguity can be resolved through interaction. To support this evaluation, we construct two disambiguation-related assets using the ground-truth code as a reference. The first asset is a disambiguated request (abbreviated as *disamb request*). It rewrites the original request so that it has no ambiguities. The second asset is a disambiguation knowledge base (abbreviated as *disamb KB*). It records ambiguity cases observed in the original request together with one corresponding resolution. In this section, we focus on ambiguities related to data transformations. Issues caused by irregular input data are handled separately in the profiling stage (Section 4.2).

disamb request. We construct *disamb request* through a validation-based process. We first prompt an LLM (GPT-5.1) to rewrite the *original request* by clarifying ambiguous parts. This produces a candidate *disamb request*. We then assess whether the rewritten request is sufficiently clear for downstream use. Specifically, we ask a strong coding model (GPT-5.1-Codex) to generate code using only the rewritten request and the *input tables*, and compare the resulting output with the *ground-truth output*. If the outputs match, we treat the rewritten request as an acceptable *disamb request*.

If the outputs do not match, we manually inspect the failure and assign it to one of three cases. (i) The rewritten request allows multiple interpretations. (ii) The rewritten request is clear, but irregular input data leads to execution failures or output mismatches. (iii) The rewritten request is clear, but the coding model fails to produce correct code. Among these cases, only (i) indicates that the request may still contain ambiguity. In such cases, we revise the rewritten request and repeat the validation process until the remaining failures no longer fall into case (i). For cases (ii) and (iii), the mismatch does not indicate ambiguity in the rewritten request, so we accept the *disamb request* as final without further revision.

disamb KB. We construct *disamb KB* by carefully aligning ambiguous parts of the original request with their clarified counterparts in *disamb request*. Each entry in *disamb KB* captures a concrete ambiguity case. It includes (a) the specific text in the original request that can be interpreted in multiple reasonable ways, (b) the corresponding clarified text in *disamb request* that reflects one intended interpretation, and (c) a minimal code snippet extracted from the ground-truth code that implements this interpretation.

We manually review extracted entries to ensure that they correspond to valid ambiguity cases, and that the recorded resolutions are consistent with the ground-truth code. This manual review also helps ensure that different ambiguity cases are treated consistently across tasks. Section 4.1 further describes the ambiguity taxonomy, the structure of *disamb KB*, and how these assets are used to evaluate interactive disambiguation.

Workflow Synthesis. To evaluate the code-to-workflow capability, we need a workflow representation that allows users to inspect and verify data preparation logic. We define a workflow abstraction with a fixed set of operators and an execution engine that applies a workflow to *input tables* and produces output tables. For each task, we construct a ground-truth workflow corresponding to *ground-truth code*. We first prompt an LLM with the operator definitions and *ground-truth code* to generate a *candidate workflow*, and then validate it by executing the workflow and comparing its output with the ground-truth output. If the *candidate* produces an incorrect

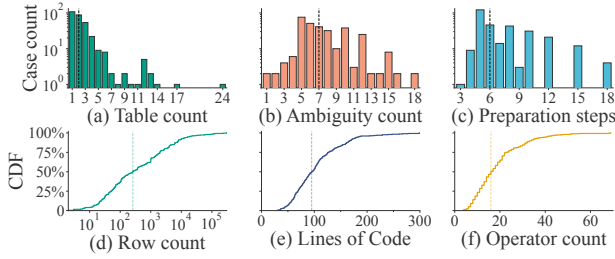


Figure 3: PREPBENCH Statistics.

result, we manually revise it until the outputs match. The final validated workflow is recorded as *ground-truth workflow*.

Since every task admits a validated *ground-truth workflow*, the operator set is sufficient to express the transformations required by PREPBENCH. Section 4.3 describes the operators, execution engine, and evaluation protocol.

LLMs assist the construction of *ground-truth code*, *disamb request*, *disamb KB*, and *ground-truth workflow*. Although execution-based validation and manual review reduce construction errors and mitigate LLM-induced bias, they cannot eliminate all remaining risks. (i) For *ground-truth code* and *ground-truth workflow*, execution is the main check. We accept them only when their execution on *input tables* yields an output table that matches *ground-truth output*, which is provided by Preppin’ Data. The remaining risk is implementation preference, since the accepted code or workflow may reflect one LLM-preferred way to realize the transformation. (ii) For *disamb request*, we check whether a coding model can reproduce *ground-truth output* from the rewritten request alone. This check helps detect remaining ambiguity, but it does not prove that *disamb request* is fully disambiguated, since the model may guess under-specified choices. When a rewrite remains ambiguous, we manually revise it. (iii) For *disamb KB*, entries link ambiguous spans in *original request* to clarifications in *disamb request* and relevant snippets in *ground-truth code*. This process may miss some ambiguity cases or align them imperfectly with their clarifications. We manually review entries for validity and consistency with *ground-truth code*. For valid clarification questions not covered by *disamb KB*, the simulator refers to *ground-truth code* so the agent can still receive an appropriate response. A remaining risk is that LLM-assisted artifacts may reflect the style of the construction model, which could give models from a similar family an advantage in evaluation.

3.3 Benchmark Statistics

PREPBENCH comprises 306 realistic tasks over 829 input tables across 32 domains. We report detailed benchmark statistics to better characterize the benchmark.

Input-Data Scale. To explore the input data scale, we count *input tables* and their total rows per task. As shown in Figure 3(a), most tasks involve at most six tables, while some require integrating many tables (up to 24). Figure 3(d) further shows substantial diversity in input size. Over 30% of tasks exceed 1,000 rows, and the largest inputs reach 291,287 rows. These results show that PREPBENCH demands multi-table processing and scalable data handling.

User-Request Diversity. We characterize the diversity of user requests by the distribution of ambiguities in *original request* and preparation steps in *disamb request*. As illustrated in Figure 3(b) and (c), PREPBENCH covers a broad spectrum of difficulty. The tasks range from simple requests with few ambiguities and steps (e.g., ≤ 3) to highly complex ones exceeding 15 ambiguities or 12 steps. These statistics indicate that PREPBENCH evaluates systems on a diverse mix of scenarios.

Table 2: Preparation steps covered by PREPBENCH.

Type	Covered	Type	Covered
Join & union	✓	Derive column	✓
Filter rows	✓	Fill empty cells	✓
Sort	✓	Edit & replace cell data	✓
Deduplicate data	✓	Normalize strings	✓
Pivot & unpivot	✓	Detect & change encoding	×
Split column	✓	Change column data type	✓
Merge columns	✓	Assign semantic data type	×
Delete empty & invalid rows	✓	Discover & merge external data	×
Delete column	✓	Generate primary key	✓
Rename column	✓		

Preparation-Step Coverage. Finally, we map each task’s ground-truth implementation to a fixed set of preparation step types (Table 2) and report which types are covered in PREPBENCH. PREPBENCH provides broad coverage of preparation step types, including core steps such as *Derive column*, *Change column data type*, and *Join & union*, reflecting common needs like computed fields, format normalization, and multi-table integration. It also includes less frequent operations such as *Pivot & unpivot*, *Split column*, and *Merge columns* for reshaping and column-level restructuring. We exclude several step types, such as those that rely on external data or operate on file encodings rather than the input tables. Overall, PREPBENCH emphasizes robustness to diverse step types and their multi-step composition.

Solution Complexity. To assess solution complexity, we count the number of lines in *ground-truth code* and the number of operators in *ground-truth workflow*. As shown in Figure 3(e), nearly half of the tasks require more than 100 lines of *ground-truth code*, and the longest solutions approach 300 lines. Figure 3(f) shows a similar distribution for workflows, with a median of about 20 operators and a maximum of 69. These statistics indicate that many PREPBENCH tasks require multi-step data-preparation solutions.

4 CAPABILITY EVALUATION AND ANALYSIS

Figure 4 provides an overview of PREPBENCH. In terms of benchmark assets, PREPBENCH consists of source materials and constructed materials. Source materials are extracted from Preppin’ Data, including *original request*, *input tables*, and *ground-truth outputs*. We construct four new materials (detailed in Section 3.2), including *ground-truth code*, *disamb request*, *disamb KB* and *ground-truth workflow*. In terms of evaluation modes, PREPBENCH consists of three modes. We explain their details in the rest of this section.

4.1 Interactive Disambiguation

User requests in NL-driven data preparation are often ambiguous, resulting in multiple valid interpretations. This section presents how PREPBENCH evaluates the *interactive-disambiguation* capability of an agent system.

Evaluation Mode. As shown in Mode 1 of Figure 4, PREPBENCH provides *original request*, *input tables*, and an LLM-based *user simulator*. An agent system may inspect *input tables* and interact with the user simulator to clarify ambiguities in the *original request*, subject to a question budget. After the interaction, the agent generates prep-code and then translates it into a workflow, following the procedures as described in Section 4.2 and Section 4.3.

User simulator. When an agent identifies ambiguities in an *original request*, it ideally relies on a human user to answer clarification questions. To automate this process, recent work has adopted LLM-based user simulators that provide natural-language feedback [51]. Such a simulator takes a clarification question posed by the agent as input and generates a response intended to mimic a human user. However, unconstrained simulators may leak information from the reference solution or deviate from the task objectives [21]. To address this issue, we define a response policy as follows.

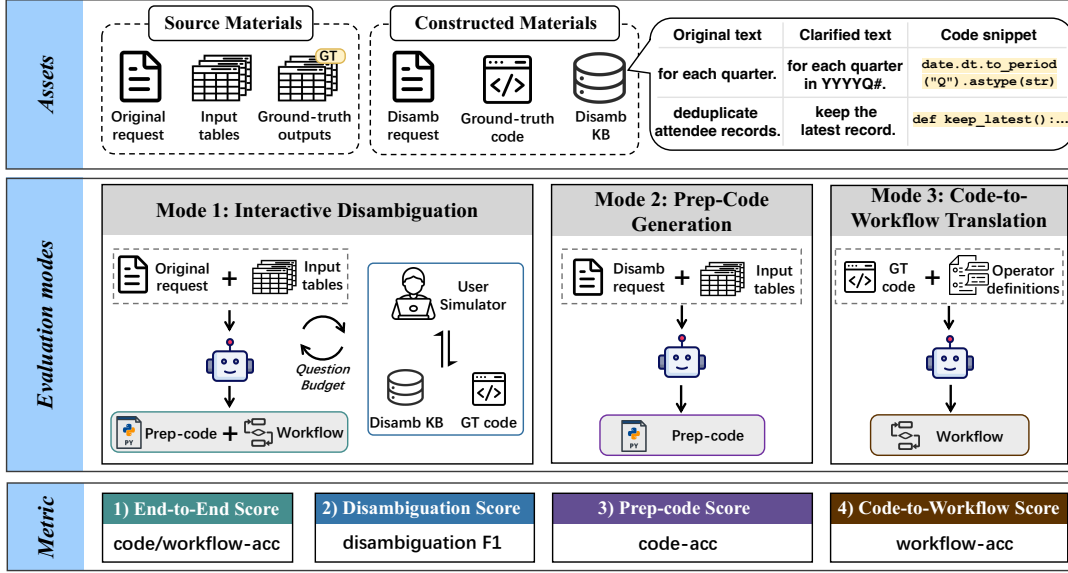


Figure 4: Evaluation pipeline in PREPBENCH. All modes share the same input tables, reference outputs, and comparator; varying the input assets isolates different capabilities.

A question is considered *invalid* if it cannot be mapped to a specific ambiguity entry, such as requesting the reference output or the full solution. For invalid questions, the user simulator returns a rejection response with an explanation. The agent is informed of the invalid-question definition before evaluation. For a valid question, if it matches an ambiguity entry in *disamb KB*, the simulator returns the corresponding resolution in natural language. Otherwise, the simulator provides a response based on *ground-truth code*, staying strictly within the scope of the question. For example, it rejects invalid questions such as “Can you provide the prepared table?” (Figure 1(c)). For valid questions, it either returns a natural-language resolution referencing a *disamb KB* item (e.g., “deduplicate: keep the latest record”, Figure 4) or provides scoped information from *ground-truth code* (e.g., “quarter uses YYYYQ# like 2024Q1”).

Evaluation Metrics. While Mode 1 (Figure 4) is designed for evaluating disambiguation quality, it also plays an important role in measuring end-to-end performance for NL-driven data preparation. We report end-to-end performance using *code-acc* and *workflow-acc*, and evaluate disambiguation quality using *disambiguation F1*.

We compute *code-acc* and *workflow-acc* by executing the generated prep-code or workflow on *input tables* to obtain an output table $\hat{T}_i$, and checking whether it matches *ground-truth output* T_i with a *comparator*. The *comparator* aligns rows in $\hat{T}_i$ and T_i by a primary key, which we label for each task using a script and manually verify. When no single column uniquely identifies rows, we use a composite key. After alignment, the *comparator* compares the remaining columns with light normalization on date formats, numeric precision, and whitespace. The *comparator* is invariant to row order unless explicitly required by the request. For tasks with multiple output tables, all tables need to match. Formally, $\text{Acc} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\text{COMPARE}(\hat{T}_i, T_i)]$.

We use *disambiguation F1* to measure disambiguation quality. Let A be the set of *disamb KB* entries for a task. Each valid question can match at most one entry in A . Let $M \subseteq A$ be the subset of entries that are matched at least once, with each entry counted only once. Let k be the total number of questions. We define $\text{PRECISION} = |M|/k$ and $\text{RECALL} = |M|/|A|$, and report their harmonic mean as *disambiguation F1*. Redundant or off-target questions increase k without enlarging M , thereby lowering precision.

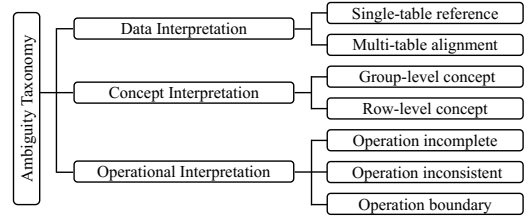


Figure 5: Ambiguity taxonomy in PREPBENCH.

Capability Requirements. To operate in Mode 1, an agent system should support asking clarification questions before code generation. At each round, the agent receives *original request*, *input tables*, the clarification history, and the remaining question budget. It then chooses one of two actions: asking a clarification question to the simulator, or stopping the interaction and proceeding to code generation. We set the budget to $[2.5 \cdot |A|]$ by default. Since the agent does not know $|A|$, the budget should exceed $|A|$ to leave room for redundant or off-target questions. Yet it should not be too large, or it would let agents succeed by asking questions indiscriminately. The $2.5\times$ multiplier balances the two. The interaction ends when the agent determines that disambiguation is complete, or when the question budget is exhausted.

Taxonomy and Statistical Analysis. To better understand ambiguities in data preparation tasks, we present a taxonomy of common ambiguity types in user requests. We observe that a request may be ambiguous for three main reasons: (i) *Data interpretation*: the request does not clearly specify which part of the data an operation should apply to; (ii) *Concept interpretation*: the request refers to concepts that are not well defined; (iii) *Operational interpretation*: the request does not clearly specify how to handle edge cases for certain operations. Figure 5 illustrates this taxonomy.

(i) *Data Interpretation*. We further identify two common subtypes under this category. *Single-table reference* occurs when an entity mentioned in the request cannot be uniquely linked to a specific column or value within a table (e.g., “sales” could refer to both *sales_amount* and *sales_count*). *Multi-table alignment* arises when a request involves multiple tables but does not specify which fields should be used to align them.

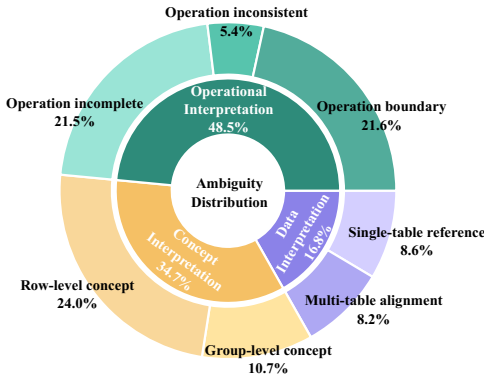


Figure 6: Distribution of ambiguities in PREPBENCH.

(ii) Concept Interpretation. We observe two common cases. *Group-level concept* refers to quantities defined over a group of rows, but the aggregation logic is unclear. For example, “average price” may refer to a simple average or a weighted average, and “monthly revenue” may be computed based on different date fields. *Row-level concept* refers to conditions or labels applied to individual rows, but their definitions are underspecified. For example, terms such as “recent records,” “high-value customers,” or “valid entries” lack clear thresholds or criteria.

(iii) Operational Interpretation. Consider this rule: “Charge \$5 for fragile items and charge \$3 for orders of \$100+.” This example illustrates three common types of operational ambiguity. *Operation incomplete* occurs when an order matches none of the rules. For example, a non-fragile order under \$100 is not covered by either rule, and the request does not specify what fee to apply. *Operation inconsistent* occurs when an order matches multiple rules. For example, a fragile order over \$100 satisfies both rules, but it is unclear whether the fee should be \$5, \$3, or their sum. *Operation boundary* occurs when threshold conditions are unclear. For example, it may be ambiguous whether an order of exactly \$100 qualifies as “\$100+.” **Statistics.** Figure 6 reports the distribution of ambiguity types in PREPBENCH. *Operational interpretation* is the most common, accounting for 48.5% of all ambiguities. This suggests that users often leave edge-case handling unspecified. In contrast, *data interpretation* accounts for only 16.8% of ambiguities, but mistakes in this category often have a broad impact. For example, an incorrect column reference or join key can affect all subsequent steps. These results indicate that resolving data references early is important to avoid downstream errors, and that systems should actively encourage users to clarify edge-case handling.

4.2 Prep-Code Generation

After disambiguation clarifies the intended operations, the system generates executable *prep-code* for the input tables. However, disambiguation alone is not sufficient. In practice, real-world tables often contain irregular or unexpected values. These irregularities may not appear in small table samples or may be unknown to the user at the time of specification. As a result, they are not mentioned in the request, but can still lead to incorrect results or execution failures if not properly handled.

Evaluation Mode. Mode 2 (Figure 4) shows how to evaluate prep-code generation in isolation, without requiring disambiguation. PREPBENCH provides *disamb request* (instead of *original request*) and *input tables*, and the correctness is measured by *code-acc*.

Capability Requirements. Mode 2 evaluates whether an agent can generate correct prep-code under practical table constraints. That is, table samples may miss irregular values in full tables, while full tables may exceed LLM context limits or make it harder for the

Table 3: Data irregularity patterns in PREPBENCH.

Pattern	Definition	Percentage
Header Irregularities	Problematic header definitions	7.1%
Row Irregularities	The table contains invalid rows	15.2%
Format Variants	Diverse formats for the same data type	34.0%
Spelling Variants	Diverse entity spellings	20.5%
Missing Variants	Inconsistent null-value representation	33.0%

LLM to focus on relevant details. Moreover, different serialization formats can lead to varying model behavior, and no single format is consistently best [46].

To succeed under Mode 2, an agent needs to reason about properties of the full tables without directly putting them into a prompt. We therefore require the agent to support data profiling. One possible profiling procedure is as follows. Before generating prep-code, the agent uses *disamb request* and a table sample to generate profiling code that scans the full tables. If the initial profiling result is insufficient, the agent may revise and rerun the profiling once. The profiling results are summarized and provided as additional input for prep-code generation.

Taxonomy and Statistical Analysis. To better understand the challenges of Mode 2, we analyze the data irregularity patterns within PREPBENCH. Through a manual inspection of *input tables* alongside *ground-truth code*, we identified 112 tasks (constituting 36.6% of PREPBENCH) that exhibit irregularities in the raw inputs. A task may involve multiple patterns. Table 3 presents a taxonomy of patterns, categorized into *structural issues* and *cell-level variants*.

Structural issues affect table schema integrity. (i) *Header irregularities* refer to problematic header definitions (e.g., missing/duplicate column names). (ii) *Row irregularities* denote extraneous rows within the table body, such as notes or interleaved headers. These two issues are less common but can break downstream processing if not detected early.

Cell-level variants are more common. (i) *Format variants* refer to values of the same type expressed in different formats, such as dates written as 2020-01-02 and 09/04/2021 in the same column. (ii) *Spelling variants* refer to the same entity appearing under different spellings, such as Jagadish and JAGADISH. (iii) *Missing variants* refer to inconsistent null-value representation, such as empty strings, N/A, or null. These issues are easy to overlook in a small sample, which can lead to wrong grouping, filtering, and aggregation results.

4.3 Code-to-Workflow Translation

Executable code can be difficult for non-technical users to inspect or modify. We therefore study *code-to-workflow translation*, which converts prep-code into an equivalent operator workflow. The workflow is a JSON-serialized DAG, where each node corresponds to a single preparation step with its parameters, making the logic easier to understand, check, and refine.

Design Rationale. Visual workflows are widely used to help end users inspect and edit data preparation logic. Commercial systems such as Tableau Prep [48] and SAS Data Preparation [44] are built around this paradigm, and prior studies on interactive data transformation provide further support [23, 43]. In PREPBENCH, workflows support both execution and interpretation. They can be executed on the input tables to produce the output, while exposing intermediate tables at operator nodes to help users localize errors.

Evaluation Mode. Mode 3 (Figure 4) shows how to evaluate code-to-workflow translation in isolation. PREPBENCH provides *ground-truth code* and a fixed set of workflow operator definitions. For each task, all agents are given the same *ground-truth code*. Each agent

Table 4: Operators and their frequency in PREPBENCH.

Operator	Description	Frequency
INPUT	Load table from file	100%
JOIN	Join tables on key columns	70.3%
UNION	Append tables by column names	35.0%
PROJECT	Select, rename, cast, or derive columns	100%
FILTER	Select rows by predicate	64.4%
AGGREGATE	Group by keys and compute aggregates	62.8%
SORT	Order rows by keys	69.3%
PIVOT	Reshape between wide and long layouts	25.8%
DEDUP	Select one row per key	24.8%
OUTPUT	Write table to file	100%
SCRIPT	Inline code fallback	14.4%

is required to output an executable workflow that preserves the semantics of *ground-truth code*.

Evaluation Metrics. Mode 3 (Figure 4) evaluates code-to-workflow translation in isolation using *workflow-acc*. Each translated workflow is executed on *input tables* and classified into one of three outcomes: (i) *Fail*, if the workflow violates format or operator specifications and cannot be executed; (ii) *Wrong*, if it executes but produces an output different from *ground-truth output*; or (iii) *Correct*, if it executes and produces an output that matches *ground-truth output*. *workflow-acc* is the fraction of workflows classified as *Correct*.

Capability Requirements. Mode 3 evaluates two capabilities. The agent is expected to follow the workflow operator definitions, including their semantics and parameter constraints. It should also apply these operators to translate *ground-truth code* into an operator DAG that preserves the semantics of the original code.

Operators and Statistical Analysis. To better understand the challenges of code-to-workflow translation, we define a workflow representation and an execution engine, and analyze the frequency of operator usage across tasks in PREPBENCH.

Operator Design. Our operator set is inspired by Tableau Prep and covers common steps in visual data preparation. Each node in the workflow represents a single atomic transformation with its parameters. When a GUI step involves multiple actions, we decompose it into separate nodes. For logic that is hard to express with operators (e.g., complex string parsing or custom date handling), we provide a SCRIPT node as a fallback, with code limited to 1,500 characters. **Execution Engine.** The execution engine first checks whether the workflow has a valid structure and conforms to the operator specifications. It then executes the workflow in topological order, creating intermediate tables as needed and discarding them once they are no longer used. The engine also provides optional tracing and error messages to support debugging.

Operator Overview. We define a total of 11 workflow operators in PREPBENCH. Table 4 lists these operators and their descriptions. It also reports the frequency of each operator across all tasks to show operator usage in the benchmark. From the table, we observe that PIVOT and DEDUP each appear in about 25% of the tasks, highlighting the significance of reshaping and record-level resolution in data preparation workflows. The SCRIPT operator appears in 14.4% of the tasks, and is used when the workflow includes logic that is hard to express with standard operators, such as complex parsing or custom date handling. In comparison, PIVOT, DEDUP, and SCRIPT are less common in Text-to-SQL QA benchmarks, highlighting the distinct characteristics and additional challenges of PREPBENCH.

User-Centric Verification Study. Workflow-acc evaluates whether a translated workflow is correct, but it does not measure whether users can inspect the transformation logic. We conducted a verification study with 16 participants who had experience with data

Table 5: User-centric verification study.

Metric	Code	Workflow
Judgment accuracy	72.9%	85.4%
Inspection confidence (1–5)	3.6	4.1
Preferred for inspection	4/16	9/16

preparation. The study used six tasks drawn from PREPBENCH cases, including four incorrect and two correct candidate implementations. None of the participants was involved in constructing PREPBENCH.

Each task included a transformation goal, an input preview, and one candidate implementation, shown as code or workflow. The workflow presented each step as an operator card with its purpose and parameters. In a counterbalanced design, each participant inspected three code tasks and three workflow tasks, and each task appeared in both formats across participants. They judged correctness, reported confidence, and stated their format preference.

Table 5 reports the results. Workflow yielded higher judgment accuracy than code, increasing from 72.9% to 85.4%. Participants also reported higher inspection confidence with workflow, increasing from 3.6 to 4.1 on a 1–5 scale. In the final preference question, 9 of 16 participants preferred workflow for inspection, 4 preferred code, and 3 reported no clear preference. These results provide preliminary user-centric evidence that workflow representations help users inspect data-preparation logic by exposing operator boundaries and parameters.

5 EXPERIMENTS

In this section, we evaluate LLM-based agents on PREPBENCH and analyze the results, aiming to answer the following questions:

- **RQ 1:** Are agents ready for end-to-end NL-driven data preparation?
- **RQ 2:** Does NL ambiguity significantly hurt performance?
- **RQ 3:** Can agents effectively resolve ambiguity through interaction?
- **RQ 4:** Can agents accurately generate prep-code from unambiguous NL?
- **RQ 5:** Can agents reliably translate prep-code into GUI workflows?

5.1 Experimental Settings

We systematically evaluated agents for NL-driven data preparation, covering end-to-end performance as well as three core capabilities: interactive disambiguation, prep-code generation, and code-to-workflow translation.

Agent Actions. In PREPBENCH, an LLM-based agent can take four actions: (1) *Clarify*: ask clarification questions, subject to a budget K ; (2) *Profile*: inspect input tables by generating profiling code; (3) *Code*: generate data preparation code; (4) *Translate*: convert code into an executable GUI workflow. For *Profile*, *Code*, and *Translate*, the system executes the generated artifacts and returns execution results or an error. *Profile* allows up to 2 attempts, while *Code* and *Translate* allow up to 3. *Profile* does not enter evaluation directly. The agent can still generate code from the sample if profiling fails. *Code* and *Translate* produce the evaluated artifacts, so they need more retries to recover from execution errors.

Models. We evaluated ten models: five proprietary (GPT-5.1-Codex, Claude-Sonnet-4.5, Gemini 3 Flash, Grok Code Fast 1, GPT-4o) and five open-weight (Kimi K2 Thinking, GLM-4.7, Qwen3-235B-A22B, DeepSeek-V3.2, DevStral 2). All models were run with temperature $\tau = 0.7$. For interactive evaluation, we used DeepSeek-V3.2 ($\tau = 0$) as the user simulator to answer clarification questions.

Metrics. We report accuracy via *code-acc* and *workflow-acc*, and disambiguation quality via *disambiguation F1* (Section 4.1). To assess how well agents resolved different ambiguity types, we report *type-level recall* (i.e., entry coverage) for each ambiguity category. For a type t , let A_t be the set of *disamb KB* entries labeled as t , and let $M_t \subseteq A_t$ be the entries matched by at least one valid question, with each entry counted at most once. We focused on recall,

Table 6: RQ1 results across five evaluation settings. Best and second-best in each column are in bold and underlined, respectively.

Model	End-to-End				Isolated Capabilities					
	Prep-Code		GUI Workflow		Interactive Disambiguation		Prep-Code Generation		Code-to-Workflow Translation	
	Acc.	Cost	Acc.	Cost	F1 Score	Cost	Acc.	Cost	Acc.	Cost
<i>Proprietary Models</i>										
GPT-5.1-Codex	54.9	115.40	34.6	264.10	51.4	74.82	85.3	82.74	67.7	159.10
Claude-Sonnet-4.5	52.0	114.00	24.5	223.20	<u>51.3</u>	66.87	70.3	70.12	44.8	102.30
Gemini 3 Flash	<u>53.3</u>	21.40	22.2	41.66	51.0	10.67	<u>74.8</u>	11.27	<u>52.3</u>	16.48
Grok Code Fast 1	30.1	18.61	13.1	35.56	34.5	13.25	57.5	10.66	45.1	20.46
GPT-4o	16.7	105.40	5.2	141.70	44.6	42.17	39.5	43.28	19.9	61.89
<i>Open-Weight Models</i>										
Kimi K2 Thinking	49.7	36.04	<u>30.1</u>	75.04	45.9	18.29	69.3	23.68	46.1	18.33
GLM-4.7	41.5	24.74	19.9	53.88	48.5	14.74	65.7	16.70	34.6	19.74
Qwen3-235B-A22B	38.2	36.99	19.3	47.51	41.3	26.68	67.7	16.76	35.0	22.29
DeepSeek-V3.2	44.8	<u>6.62</u>	15.7	<u>11.59</u>	45.7	<u>4.48</u>	65.4	<u>3.40</u>	27.8	<u>4.85</u>
DevStral 2	33.3	1.89	8.5	3.16	42.3	1.24	44.1	1.06	10.5	1.82

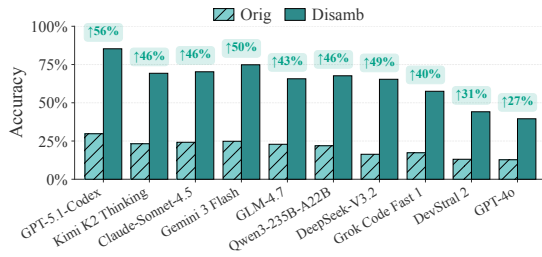


Figure 7: Ambiguity gap across agents.

as unmatched questions cannot be assigned to a specific ambiguity type and thus do not support per-type precision. We define $\text{RECALL}(t) = |M_t|/|A_t|$. We also reported the average cost per task in USD ($\times 10^{-3}$). Cost is computed over the execution path used by each evaluation setting, including all LLM calls and retries. For example, Interactive Disambiguation includes only *Clarify*, while End-to-End Prep-Code additionally includes *Profile* and *Code*.

5.2 RQ1: End-to-End Performance

RQ1 asks whether current agents are ready for end-to-end NL-driven data preparation. Table 6 reports performance across five evaluation settings. The first two followed the end-to-end protocol of Mode 1 (Section 4.1): the agent received *original request* and could perform *Clarify*, *Profile*, *Code*, and *Translate* actions. We reported task accuracy for both prep-code and GUI workflows. The remaining three capabilities were evaluated in isolation. Interactive Disambiguation was evaluated in Mode 1 using disambiguation F1. Prep-Code Generation was evaluated in Mode 2 (Section 4.2), where the agent received *disamb request* and only performed *Profile* and *Code*; Code-to-workflow Translation was evaluated in Mode 3 (Section 4.3), where the agent received *ground-truth code* and only performed *Translate*. For brevity, we refer to these five settings as E2E-Code, E2E-Workflow, Disambig, CodeGen, and Translation.

Current agents are not ready for end-to-end NL-driven data preparation. As shown in the E2E-Code and E2E-Workflow accuracy columns of Table 6, even the best-performing model (GPT-5.1-Codex) achieved only 54.9% on E2E-Code. Performance dropped further in E2E-Workflow, where no model exceeded 35% accuracy.

Disambiguation leaves substantial room for improvement. As shown in Disambig (Table 6), disambiguation F1 scores ranged from 34.5 to 51.4, indicating that current models struggled to identify ambiguities in user requests. Moreover, interactive disambiguation did

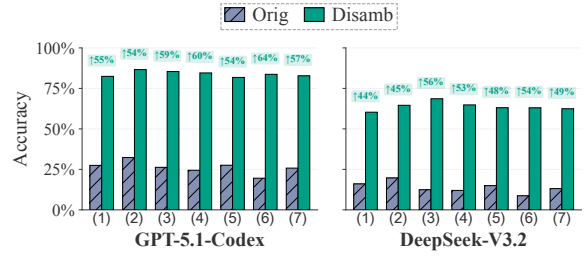


Figure 8: Ambiguity gap by type: (1) single-table reference, (2) multi-table alignment, (3) group-level concept, (4) row-level alignment, (5) operation incomplete, (6) operation inconsistent, and (7) operation boundary.

not fully recover the accuracy achievable with a disambiguated request: comparing E2E-Code with CodeGen, GPT-5.1-Codex dropped from 85.3% to 54.9%, and Gemini 3 Flash from 74.8% to 53.3%. We examined this gap further in RQ3.

Code-to-workflow translation remains the bottleneck. Comparing E2E-Code with E2E-Workflow, GPT-5.1-Codex dropped from 54.9% to 34.6%, reflecting errors introduced during translation. Even with correct prep-code as input (Translation), accuracy remained capped at 67.7%, confirming that workflow generation is a distinct and error-prone task. This gap is structural: prep-code is free-form, while workflows need to conform to a schema-constrained, operator-based representation that models have not seen during training. We analyze translation failures in RQ5.

Cost-effectiveness varies across models. Higher cost does not guarantee better performance: GPT-4o is expensive yet performs poorly on E2E-Workflow. In contrast, Gemini 3 Flash nearly matched GPT-5.1-Codex on E2E-Code (53.3% vs. 54.9%) at under one-fifth the cost. These results highlight the need to consider both accuracy and cost in deployment decisions.

Takeaway I. Current agents are not ready for end-to-end NL-driven data preparation. Performance is constrained by two bottlenecks: insufficient disambiguation limits prep-code accuracy, and workflow translation further degrades output.

5.3 RQ2: Impact of NL Ambiguity

RQ2 examines how NL ambiguity affects prep-code accuracy. We compared two settings: Orig uses *original request*, while Disamb

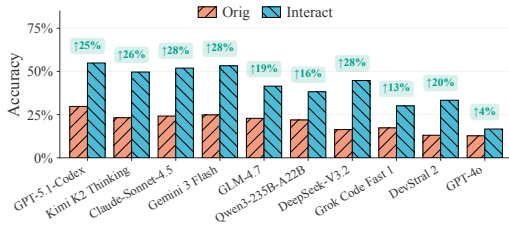


Figure 9: Interact gains across models.

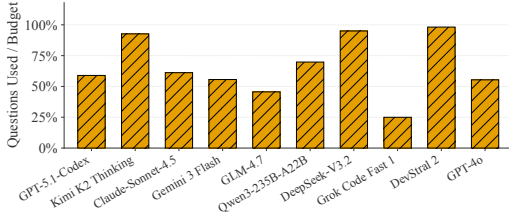


Figure 10: Question budget usage across models.

uses *disamb request*. The agent received the request and could perform *Profile* and *Code* actions. We define the *ambiguity gap* as the accuracy increase from Orig to Disamb.

NL ambiguity substantially degrades solution accuracy. Figure 7 shows a consistent ambiguity gap across all models, with improvements ranging from +27 to +56 points. GPT-5.1-Codex achieved the highest Disamb accuracy (85%) and the largest gap (+56%), suggesting stronger models benefit more from disambiguation. We examined whether interaction can close this gap in RQ3.

Concept interpretation ambiguities show robust gains from disambiguation. Figure 8 breaks down the ambiguity gap by type for GPT-5.1-Codex and DeepSeek-V3.2. Both models show large and consistent gains on concept interpretation ambiguities, including group-level and row-level cases, where aggregation logic or thresholds need to be explicitly specified. Operation inconsistent also improves substantially, as conflicting rules require explicit resolution. Data interpretation ambiguities show smaller or more moderate gaps, as models can infer column mappings from schema.

Takeaway II. NL ambiguity substantially degrades accuracy, but the largest recoveries come from resolving ambiguities that determine transformation semantics rather than input interpretation alone.

5.4 RQ3: Effectiveness of Interaction

The ambiguity gap in RQ2 raises a practical question: can agents resolve ambiguities through interaction? We compared two settings, Orig and Interact, both using the *original request*. In both settings, the agent could perform *Profile* and *Code* actions; Interact additionally permitted *Clarify* actions to resolve ambiguities. This setting simulated realistic usage, where the agent needed to identify ambiguities and resolve them through interaction.

Interaction helps, but gains vary by model. All agents improved under Interact, with accuracy gains ranging from +4 to +28 points over Orig (Figure 9). Models that start higher under Orig often gain more from interaction. For example, GPT-5.1-Codex and Kimi K2 Thinking achieved large gains, while GPT-4o improved the least. DeepSeek-V3.2 is a notable exception, showing a large gain despite its low accuracy under Orig. Overall, recovery depends not only on code generation quality but also on how well a model identifies ambiguities and resolves them.

More questions do not guarantee better gains. A natural hypothesis is that asking more questions leads to higher accuracy, but Figure 10 shows otherwise. Models varied in how much of the

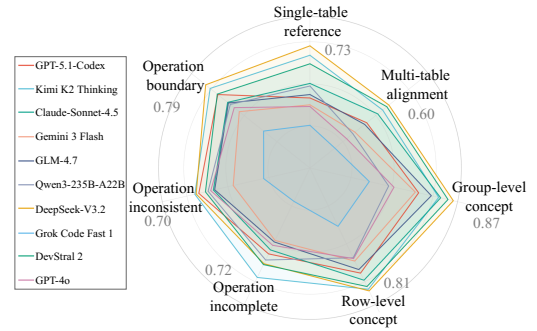


Figure 11: Disambiguation recall by ambiguity type.

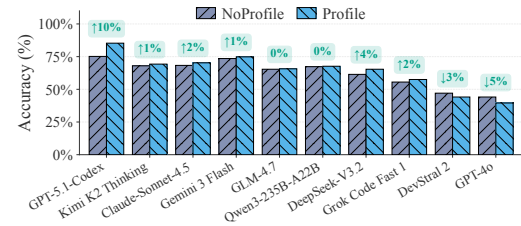


Figure 12: Profiling gains across models.

budget they used, yet gains depended on question quality rather than volume. For example, DevStral 2 consumed over 95% of the budget for a 20-point gain, while GPT-5.1-Codex gained 25 points using about 60%. This suggests that effective agents can prioritize impactful ambiguities and decide when to stop, thereby avoiding unnecessary or redundant questions.

Multi-table alignment ambiguities are hardest to identify. Figure 11 shows that multi-table alignment ambiguities have the lowest recall, as models often misinterpret join fields, assuming their understanding aligns with the user’s intent. In contrast, *Group-level concept* has the highest recall, as unclear definitions of concepts such as averages or sums make the ambiguity easier to identify. DeepSeek-V3.2 performs well across types, suggesting robust identification capabilities.

Takeaway III. Interaction improves accuracy, but question quality matters more than quantity. Models show significant performance differences across ambiguity types, with Multi-table alignment being the hardest to resolve.

5.5 RQ4: Prep-Code Generation

RQ4 asks whether agents can accurately generate prep-code given an *unambiguous request*. We compared two settings that both provided *disamb request* and a small input sample. In NoProfile, the agent directly performed *Code*. In Profile, the agent could perform *Profile* on the full input tables before performing *Code*. This setup isolates irregularity handling and tests whether profiling improves prep-code generation under irregular data.

Data profiling is a double-edged sword. Figure 12 shows that data profiling has uneven effects across models. GPT-5.1-Codex and DeepSeek-V3.2 gained 10 and 4 points, while several models changed little; DevStral 2 and GPT-4o even regressed. Profiling helps only when summaries are concise, relevant, and correctly used in code. Otherwise, models may misfire on noisy signals or lose focus, skipping required steps.

Profiling effects vary by irregularity type. As shown in Figure 13, profiling helps stronger models (e.g., GPT-5.1-Codex and Kimi K2 Thinking) across most irregularity patterns, but the gains vary. Header irregularities yield the most consistent improvements. Without profiling, models often overlook header cleanup even when

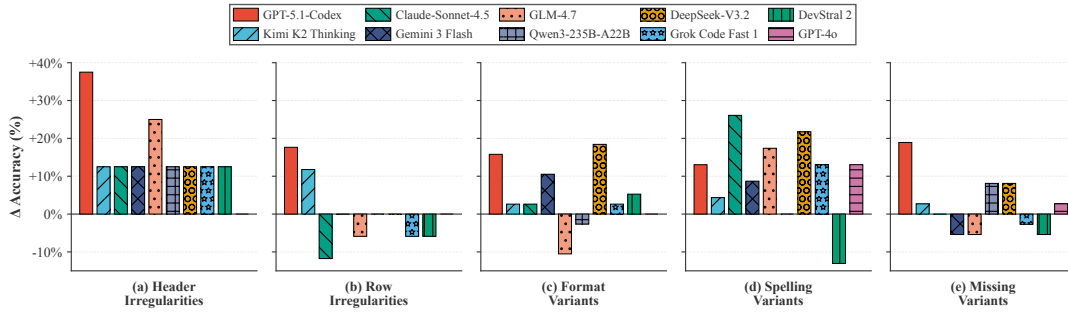


Figure 13: Profiling gains by irregularity type.

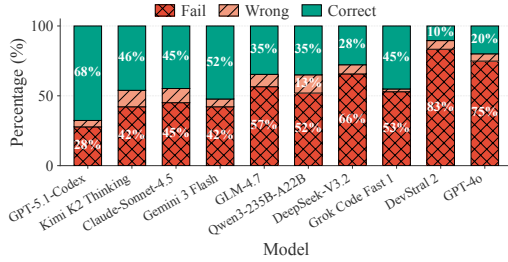


Figure 14: Translation outcome composition across models.

input samples contain suspicious column names. Profile directs attention to input validation, and header issues are easy to fix once identified. In contrast, other irregularity patterns are more diverse and sparse, making them harder to detect and address. When exposed, weaker models may misinterpret partial cues and apply harmful or excessive cleaning.

Takeaway IV. Profiling yields uneven gains and its utility varies by model. Without explicit instructions, agents often overlook or mishandle data irregularities.

5.6 RQ5: Reliability of Workflow Translation

The preceding experiments focus on code correctness. In practice, systems also need to generate GUI workflows that users can inspect. RQ5 evaluates whether models can reliably translate code into executable workflows. In this setting, each model was given *ground-truth code* together with the workflow operator definitions and could only perform *Translate*. We executed each translated workflow on *input tables* and classified its execution outcome as *Fail*, *Wrong*, or *Correct* following Section 4.3.

Failures are dominated by non-executable workflows. Figure 14 shows that *Wrong* is consistently rare, never exceeding 13%. When translation fails, the cause is typically *Fail* rather than an incorrect but executable workflow. Conversely, workflows that execute are typically *Correct*, indicating that agents grasp the logic but struggle to express it with valid workflow structures. The primary challenge lies in generating structurally valid workflows. Since our operators are not seen during pretraining and are learned solely from the prompt, models often produce incorrect bindings or parameters that block execution.

Complexity degrades translation reliability. Figure 15 shows that translation remains imperfect even for short workflows. At 3–5 steps, GPT-5.1-Codex reached 77.27% *Correct*, while Kimi K2 Thinking lagged at 57.58%. As workflows grew longer, *Correct* dropped mainly due to rising *Fail* rates, not *Wrong*, indicating that the dominant challenge lies in generating executable workflows. These failures stem from binding and parameter errors under unfamiliar

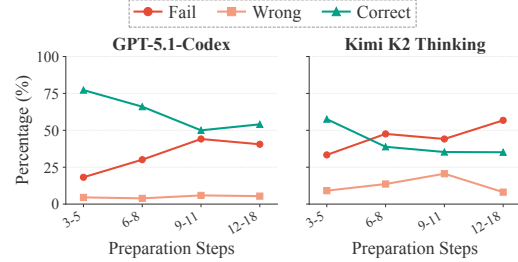


Figure 15: Translation outcomes by preparation steps.

operator definitions. Although Kimi K2 Thinking is the strongest open-weight model in code-to-workflow translation (Figure 14), it lags behind GPT-5.1-Codex across all workflow lengths (Figure 15).

Takeaway V. Workflow translation remains unreliable: failures are mostly due to non-executable workflows, and increasing complexity further reduces both executability and accuracy.

6 RESEARCH OPPORTUNITIES

Improving End-to-End Accuracy. End-to-end NL-driven data preparation requires both correct outputs and verifiable procedures. RQ1 (Section 5.2) shows that current agents remain unreliable, mainly due to failures in ambiguity detection, irregular-input handling, and verifiable workflow generation. We outline research opportunities below.

Interactive Disambiguation. A key opportunity is to improve clarification seeking through: (i) *foundation-model training* that rewards asking rather than guessing for underspecified requests, e.g., via preference optimization or RL [55]; and (ii) *agentic system design* that rewrites raw requests into explicit specifications, so downstream execution need not re-interpret the original request.

Prep-Code Generation. NL-to-code pipelines often struggle on irregular tables when relying only on requests and small samples. Recent systems use profiling and execution feedback for iterative refinement [16, 20]. A promising profile-first direction is to extract table properties such as data types, missing values, and key constraints, then use them to guide code generation and add runtime assertions for error localization and repair [45].

Code-to-Workflow Translation. Workflows matter because users often cannot inspect generated code directly. One opportunity is an *LLM+compiler* design: the compiler analyzes prep-code to derive an initial workflow, while the LLM fills remaining parameters and generates human-readable annotations. The workflow should expose intermediate results to support inspection and edits [23].

User Profiles for Data Preparation. In data preparation, users often make recurring choices across tasks, such as how to handle missing values or resolve duplicates. Repeatedly asking for these

choices increases interaction cost, while global defaults may misalign with user preferences and downstream objectives [13]. This motivates user profiles that capture reusable decisions across tasks.

Designing such profiles requires policies for when to ask, store, and reuse user-specific choices, as well as mechanisms to scope profile entries by schema, operation, or table characteristics, resolve conflicts, and update entries as preferences change.

Preparation Beyond Structured Data. PREPBENCH focuses on tabular inputs, so its empirical conclusions are limited to tabular data preparation. A natural extension is semi-structured and unstructured sources such as JSON, logs, and PDFs, where disambiguation requires schema-free alignment over entities, fields, or extracted values; profiling must summarize nested structures and noisy text; and workflows need operators for extraction, parsing, normalization, and nested-structure manipulation. Extending PREPBENCH to non-tabular sources is therefore an important direction, with a full treatment left to future work.

7 RELATED WORK

LLM-based Data Preparation. Data preparation has long been a core topic in data management research [8, 23, 41, 42]. Recent studies have explored LLMs for data preparation [2, 57], which can be broadly grouped into two lines of work by the role they assign to the model in the pipeline. Some use LLMs as data-processing operators, while others use them as interfaces that translate user intent into executable preparation actions.

LLM as Data Processor. This line uses LLMs directly as data-processing operators, including single-step transformations [31], pipeline preprocessing [54], semantic annotation and curation [7], record-level matching [50], and metadata generation for table discovery [24]. Our work instead studies LLMs as an NL interface rather than as the execution engine.

LLM as NL Interface. This line translates NL requests into executable data preparation actions. SheetCopilot [28], SheetAgent [6], and Dango [5] operate within spreadsheet or GUI environments, mapping user intents to application-specific actions. CleanAgent [38] targets data standardization with tool invocation and iterative refinement. AutoPrep [11] and HAIPipe [3] generate task-specific pipelines for downstream question answering or machine learning, while AgenticData [47] targets heterogeneous data analytics. LLMs have been applied to data integration, including semantic matching [36], schema matching [35], and harmonization pipelines [39, 43], which support preparation workflows.

A central challenge in NL interfaces is handling ambiguous user requests. In Text-to-SQL, *Schema Linking* and *Value Linking* identify referenced columns and values, and prior work has studied ambiguity in input tables [19, 49]; these problems mainly correspond to *Data Interpretation* in our taxonomy. NL-driven data preparation further requires users to define multi-step transformations, introducing *Concept Interpretation* and *Operational Interpretation*. Together, these two categories account for 83.2% of ambiguities in PREPBENCH, and our experiments show that they substantially affect prep-code accuracy. Unlike prior work that builds agents for specific data preparation tasks, PREPBENCH targets NL-driven data preparation in general and introduces a benchmark that systematically evaluates its core capabilities.

LLM4DATA Benchmarks. Recent surveys provide comprehensive overviews of LLM benchmarks for data-centric tasks over structured, semi-structured, and unstructured inputs [58, 59]. We

group representative benchmarks by primary task: querying and reasoning [4, 29, 53], spreadsheets [14, 32], executable data analysis [10, 17, 18, 26, 56], and pipelines [22, 25, 27, 43].

In spreadsheet-centric settings, TEMPTABQA [14] studies table reasoning with temporal constraints, and SpreadsheetBench [32] evaluates realistic spreadsheet manipulation tasks. For data analysis, DS-1000 [26] and DA-Code [18] benchmark executable data-science coding, while DA-Bench [17], DABstep [10], and DSEval [56] focus on multi-step analysis with execution-based verification. For pipelines, Spider 2.0 [27] moves Text-to-SQL toward workflow-oriented settings, ELT-Bench [22] benchmarks end-to-end ELT construction, Harmonia [43] evaluates mixed-initiative harmonization workflows, and KramaBench [25] benchmarks end-to-end pipelines built from real artifacts. Existing benchmarks target different settings or evaluate individual capabilities in isolation. PREPBENCH systematically evaluates the three core capabilities required for NL-driven data preparation: interactive disambiguation, prep-code generation, and code-to-workflow translation.

8 CONCLUSION

This paper examined NL-driven data preparation and evaluated how far current LLM-based agents are from supporting it in realistic settings. We presented PREPBENCH, a benchmark systematically constructed from real-world Preppin' Data challenges. We converted each challenge into a benchmark task with *ground-truth code*, *disamb request*, *disamb KB* and *ground-truth workflow*. We designed three execution modes that enabled both end-to-end evaluation and targeted analysis of individual capabilities. We analyzed various aspects of PREPBENCH, including task complexity, ambiguity types, data irregularities, and operator usage. These statistics characterized the coverage and difficulty of the benchmark.

We evaluated ten state-of-the-art proprietary and open-weight LLMs on PREPBENCH. The results showed that NL-driven data preparation remained challenging for current models. Firstly, the best-performing model (GPT-5.1-Codex) achieved 54.9% accuracy on end-to-end prep-code generation. Removing ambiguity increased accuracy to 85.3%, suggesting that ambiguous user requests were a major source of error. Secondly, interactive disambiguation improved accuracy, but its effectiveness was limited by the quality of clarification questions, which were often incomplete or ineffective. Thirdly, code-to-workflow translation was also challenging, as it required the model to reason over explicit operator definitions provided in the prompt, rather than relying solely on prior knowledge learned during pretraining. Fourthly, model cost did not consistently correlate with performance; for prep-code generation, Gemini 3 Flash nearly matched the best accuracy at less than one-fifth of the cost. Overall, our study quantitatively characterizes the limitations of current LLM-based agents and identifies the capability gaps that need to be addressed to support NL-driven data preparation in practice. We released PREPBENCH, the evaluation framework, and baseline implementations at <https://github.com/TsinghuaDatabaseGroup/prepbench>.

ACKNOWLEDGMENTS

This paper was supported by the National Key R&D Program of China (2023YFB4503600), NSF of China (62525202, 62232009), the Shenzhen Project (CJGJZD20230724093403007), China Railway Science Research Institute Group Co., Ltd., Zhongguancun Lab, Huawei, and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

- [1] Anaconda, Inc. 2020. State of Data Science 2020. <https://www.anaconda.com/resources/whitepaper/state-of-data-science-2020> Accessed Jan. 11, 2026.
- [2] Mengshi Chen, Yuxiang Sun, Tengchao Li, Jianwei Wang, Kai Wang, Xuemin Lin, Ying Zhang, and Wenjie Zhang. 2025. Empowering Tabular Data Preparation with Language Models: Why and How? *arXiv preprint arXiv:2508.01556* (2025).
- [3] Sibe Chen, Nan Tang, Ju Fan, Xuemi Yan, Chengliang Chai, Guoliang Li, and Xiaoyong Du. 2023. Haipipe: Combining human-generated and machine-generated pipelines for data preparation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.
- [4] Wenhui Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyu Zhou, and William Yang Wang. 2020. TabFact: A Large-scale Dataset for Table-based Fact Verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net. <https://openreview.net/forum?id=rkeJRhNYDH> Accessed Jan. 11, 2026.
- [5] Wei-Hao Chen, Weixi Tong, Amanda Case, and Tianyi Zhang. 2025. Dango: A Mixed-Initiative Data Wrangling System using Large Language Model. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–28.
- [6] Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. 2025. SheetAgent: towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025*. 158–177.
- [7] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. SEED: Domain-specific data curation with large language models. *arXiv preprint arXiv:2310.00749* (2023).
- [8] Michele Dallachiesa, Amr Ebad, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 541–552.
- [9] Mingwen Dong, Nischal Ashok Kumar, Yiqun Hu, Anuj Chauhan, Chung-Wei Hang, Shuaichen Chang, Lin Pan, Wuwei Lan, Henghui Zhu, Jiarong Jiang, et al. 2025. PRACTIQ: A practical conversational text-to-SQL dataset with ambiguous and unanswerable queries. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 255–273.
- [10] Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. 2025. DABstep: Data Agent Benchmark for Multi-step Reasoning. *arXiv preprint arXiv:2506.23719* (2025).
- [11] Meihao Fan, Ju Fan, Nan Tang, Lei Cao, Guoliang Li, and Xiaoyong Du. 2025. AutoPrep: Natural Language Question-Aware Data Preparation with a Multi-Agent Framework. *Proc. VLDB Endow.* 18, 10 (2025), 3504–3517. <https://doi.org/10.14778/3748191.3748211>
- [12] Yuhang Ge, Yachuan Liu, Zhangyan Ye, Yuren Mao, and Yunjun Gao. 2025. Text-to-pipeline: Bridging natural language and data preparation pipelines. *arXiv preprint arXiv:2505.15874* (2025).
- [13] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2024. Automated data cleaning can hurt fairness in machine learning-based decision making. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (2024), 7368–7379.
- [14] Vivek Gupta, Pranshu Kandoi, Mahek Vora, Shuo Zhang, Yujie He, Ridho Reinanda, and Vivek Srikrumar. 2023. TempTabQA: Temporal question answering for semi-structured tables. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2431–2453.
- [15] Mazhar Hameed and Felix Naumann. 2020. Data preparation: A survey of commercial tools. *ACM sigmod record* 49, 3 (2020), 18–29.
- [16] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, et al. 2025. Data interpreter: An LLM agent for data science. In *Findings of the Association for Computational Linguistics: ACL 2025*. 19796–19821.
- [17] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21–27, 2024*. OpenReview.net. <https://openreview.net/forum?id=d5LURMSFTx> Accessed Jan. 11, 2026.
- [18] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 2024. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12–16, 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 13487–13521. <https://doi.org/10.18653/V1/2024.EMNLP-MAIN.748>
- [19] 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).
- [20] Zezhou Huang and Eugene Wu. 2024. Cocoon: Semantic table profiling using large language models. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*. 1–7.
- [21] Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Özcan, Yannis Papakonstantinou, and Reynold Cheng. 2025. BIRD-INTERACT: Re-imagining Text-to-SQL Evaluation for Large Language Models via Lens of Dynamic Interactions. *CoRR abs/2510.05318* (2025). <https://doi.org/10.48550/ARXIV.2510.05318> arXiv:2510.05318
- [22] Tengjun Jin, Yuxuan Zhu, and Daniel Kang. 2025. ELT-Bench: An End-to-End Benchmark for Evaluating AI Agents on ELT Pipelines. *arXiv preprint arXiv:2504.04808* (2025).
- [23] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*. 3363–3372.
- [24] Moe Kayali, Anton Lykov, Ilias Fountalis, Nikolaos Vasiloglou, Dan Olteanu, and Dan Suciu. 2024. CHORUS: Foundation Models for Unified Data Discovery and Exploration. *Proc. VLDB Endow.* 17, 8 (2024), 2104–2114. <https://doi.org/10.14778/3659437.3659461>
- [25] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton A Zabreyko, Chenning Li, Ferdi Kosmann, Jialin Ding, et al. 2025. KramaBench: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. *arXiv preprint arXiv:2506.06541* (2025).
- [26] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*. PMLR, 18319–18345.
- [27] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24–28, 2025*. OpenReview.net. <https://openreview.net/forum?id=XmProj9cPs> Accessed Jan. 11, 2026.
- [28] Hongxin Li, Jingran Su, Yuntao Chen, Qing Li, and Zhao-Xiang Zhang. 2023. SheetCopilot: Bringing software productivity to the next level through large language models. *Advances in Neural Information Processing Systems* 36 (2023), 4952–4984.
- [29] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can LLM already serve as A database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.
- [30] Meng Li, Timothy M McPhillips, Dingmin Wang, Shin-Rong Tsai, and Bertram Ludäscher. 2025. CRABS: A syntactic-semantic pincer strategy for bounding LLM interpretation of Python notebooks. *arXiv preprint arXiv:2507.11742* (2025).
- [31] Luyi Ma, Nikhil Thakurdesai, Jiao Chen, Jianpeng Xu, Evren Korpoglu, Sushant Kumar, and Kannan Achan. 2023. LLMs with User-defined prompts as generic data operators for reliable data processing. In *2023 IEEE International Conference on big data (BigData)*. IEEE, 3144–3148.
- [32] Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems* 37 (2024), 94871–94908.
- [33] Avanika Narayan, Ines Chami, Laurel J. Orr, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? *Proc. VLDB Endow.* 16, 4 (2022), 738–746. <https://doi.org/10.14778/3574245.3574258>
- [34] OpenAI. 2024. ChatGPT. <https://chatgpt.com> Accessed Jan. 11, 2026.
- [35] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. 2024. Schema matching with large language models: an experimental study. *arXiv preprint arXiv:2407.11852* (2024).
- [36] Ralph Peeters, Aaron Steiner, and Christian Bizer. 2025. Entity Matching using Large Language Models. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25–28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queral, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 529–541. <https://doi.org/10.48786/EDBT.2025.42>
- [37] Preppin’ Data. 2024. Preppin’ Data Challenges. <https://www.preppindata.com/challenges> Accessed Jan. 11, 2026.
- [38] Danrui Qi, Zhengjie Miao, and Jiannan Wang. 2025. CleanAgent: Automating Data Standardization with LLM-based Agents. In *VLDB 2025 Workshop: Data Driven AI*. https://www.vldb.org/2025/Workshops/VLDB-Workshops-2025/DATAI/DATAI25_8.pdf Accessed Jan. 11, 2026. Artifact available at <https://github.com/sfu-db/CleanAgent>, accessed Jan. 11, 2026.
- [39] Zhangcheng Qiang, Weiqing Wang, and Kerry Taylor. 2024. Agent-OM: Leveraging LLM Agents for Ontology Matching. *Proc. VLDB Endow.* 18, 3 (2024), 516–529. <https://doi.org/10.14778/3712221.3712222>
- [40] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter’s Wheel: An Interactive Data Cleaning System. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11–14, 2001, Roma, Italy*, Peter M. G. Apers, Paolo Atzeni, Stefano Ceri, Stefano Paraboschi, Kotagiri Ramamohanarao, and Richard T. Snodgrass (Eds.). Morgan Kaufmann, 381–390.
- [41] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201. <https://doi.org/10.14778/3137628.3137631>
- [42] El Kindi Rezig, Lei Cao, Michael Stonebraker, Giovanni Simonini, Wenbo Tao, Samuel Madden, Mourad Ouzzani, Nan Tang, and Ahmed K. Elmagarmid. 2019. Data Civilizer 2.0: A Holistic Framework for Data Preparation and Analytics. *Proc.*

- VLDB Endow. 12, 12 (2019), 1954–1957. <https://doi.org/10.14778/3352063.3352108>
- [43] Aécio Santos, Eduardo HM Pena, Roque Lopez, and Juliana Freire. 2025. Interactive Data Harmonization with LLM Agents: Opportunities and Challenges. *arXiv preprint arXiv:2502.07132* (2025).
 - [44] SAS Institute. 2024. SAS Data Preparation. https://www.sas.com/en_us/software/data-preparation.html Accessed Jan. 11, 2026.
 - [45] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1781–1794.
 - [46] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 645–654.
 - [47] Ji Sun, Guoliang Li, Peiyao Zhou, Yihui Ma, Jingzhe Xu, and Yuan Li. 2025. AgenticData: An Agentic Data Analytics System for Heterogeneous Data. *CoRR abs/2508.05002* (2025). <https://doi.org/10.48550/ARXIV.2508.05002>
 - [48] Tableau Software. 2024. Tableau Prep. <https://www.tableau.com/products/prep> Accessed Jan. 11, 2026.
 - [49] Enzo Veltri, Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Data ambiguity profiling for the generation of training examples. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 450–463.
 - [50] Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xuanang Chen, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025. Match, compare, or select? an investigation of large language models for entity matching. In *Proceedings of the 31st International Conference on Computational Linguistics*. 96–109.
 - [51] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024. MINT: Evaluating LLMs in Multi-turn Interaction with Tools and Language Feedback. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. <https://openreview.net/forum?id=jp3gWrMulZ> Accessed Jan. 11, 2026.
 - [52] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, et al. 2023. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 126–173.
 - [53] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. <https://doi.org/10.18653/V1/D18-1425>
 - [54] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2024. Large Language Models as Data Preprocessors. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. <https://vldb.org/workshops/2024/proceedings/TaDA/TaDA.11.pdf> Accessed Jan. 11, 2026.
 - [55] Michael J. Q. Zhang, W. Bradley Knox, and Eunsol Choi. 2025. Modeling Future Conversation Turns to Teach LLMs to Ask Clarifying Questions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. <https://openreview.net/forum?id=cwuSAR7EKd> Accessed Jan. 11, 2026.
 - [56] Yuge Zhang, Qiyang Jiang, Xingyu Han, Nan Chen, Yuqing Yang, and Kan Ren. 2024. Benchmarking Data Science Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 5677–5700. <https://doi.org/10.18653/V1/2024.ACL-LONG.308>
 - [57] 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).
 - [58] Xuanhe Zhou, Junxuan He, Wei Zhou, Haodong Chen, Zirui Tang, Haoyu Zhao, Xin Tong, Guoliang Li, Youmin Chen, Jun Zhou, et al. 2025. A Survey of LLM× DATA. *arXiv preprint arXiv:2505.18458* (2025).
 - [59] Yizhang Zhu, Liangwei Wang, Chenyu Yang, Xiaotian Lin, Boyan Li, Wei Zhou, Xinyu Liu, Zhangyang Peng, Tianqi Luo, Yu Li, et al. 2025. A Survey of Data Agents: Emerging Paradigm or Overstated Hype? *arXiv preprint arXiv:2510.23587* (2025).