ConDABench: Interactive Evaluation of Language Models for Data Analysis

Avik Dutta*
Microsoft
Bangalore, India
avikdutta772000@gmail.com

Priyanshu Gupta*
Microsoft
Bangalore, India
priyansgupta@microsoft.com

Hosein Hasanbeig*
Microsoft
Redmond, USA
hosein.hasanbeig@microsoft.com

Rahul Pratap Singh
Microsoft
Bangalore, India
t-rahulsingh@microsoft.com

Harshit Nigam
Microsoft
Bangalore, India
hnigam0@gmail.com

Sumit Gulwani
Microsoft
Redmond, USA
sumitg@microsoft.com

Arjun Radhakrishna
Microsoft
Redmond, USA
arradha@microsoft.com

Gustavo Soares
Microsoft
Redmond, USA
gsoares@microsoft.com

Ashish Tiwari
Microsoft
Redmond, USA
astiwar@microsoft.com

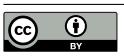
Abstract

Real-world data analysis tasks often come with under-specified goals and unclean data. User interaction is necessary to understand and disambiguate a user's intent, and hence, essential to solving these complex tasks. Existing benchmarks for evaluating LLMs on data analysis tasks do not capture these complexities or provide first-class support for interactivity. We introduce **ConDABench**, a framework for generating conversational data analysis (ConDA) benchmarks and evaluating external tools on the generated benchmarks. **ConDABench** consists of (a) a multi-agent workflow for generating realistic benchmarks from articles describing insights gained from public datasets, (b) 1,420 ConDA problems generated using this workflow, and (c) an evaluation harness that, for the first time, makes it possible to systematically evaluate conversational data analysis tools on the generated ConDA problems. Evaluation of state-of-the-art LLMs on the benchmarks reveals that while the new generation of models are better at solving more instances, they are not necessarily better at solving tasks that require sustained, long-form engagement. **ConDABench** is an avenue for model builders to measure progress towards truly collaborative models that can complete complex interactive tasks.

CCS Concepts

• **Computing methodologies** → **Discourse, dialogue and pragmatics**; • **Information systems** → *Data cleaning*; • **Human-centered computing** → **Empirical studies in interaction design**; **User centered design**; **Collaborative interaction**.

*Corresponding authors



SIGMOD Companion '26, Bengaluru, India

ACM ISBN 979-8-4007-2450-3/2026/05
https://doi.org/10.1145/3788853.3803099

Keywords

Conversational Data Analysis, Interactive Evaluation, Benchmark Generation, Multi-Agent Framework, Ambiguity Resolution

ACM Reference Format:
Avik Dutta, Priyanshu Gupta, Hosein Hasanbeig, Rahul Pratap Singh, Harshit Nigam, Sumit Gulwani, Arjun Radhakrishna, Gustavo Soares, and Ashish Tiwari. 2026. **ConDABench**: Interactive Evaluation of Language Models for Data Analysis. In *Companion of the International Conference on Management of Data (SIGMOD Companion '26), May 31-June 05, 2026, Bengaluru, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3788853.3803099

1 Introduction

LLM-powered conversational assistants such as ChatGPT and Gemini are rapidly gaining popularity for supporting a wide range of cognitive tasks. One area seeing especially notable growth is data analysis, with applications expanding across domains like business intelligence [31], healthcare [9], and scientific research [3, 21]. Despite their growing use, LLMs remain imperfect at performing data analysis due to the task's inherent complexity, which demands logical reasoning, code generation, procedural execution, and interactivity. These challenges make data analysis a particularly valuable test bed for evaluating the capabilities and limitations of LLMs. However, generating realistic benchmarks capturing the nuances of real-world data analysis is challenging. User queries are often *vague and underspecified*, particularly with respect to the underlying data context. Users frequently refine queries iteratively while *interacting* with a data analysis assistant. Furthermore, real-world datasets are often unclean, which needs to be reflected in the benchmark set.

We introduce **ConDABench**, a suite of tools to **generate** benchmark sets and to effectively **evaluate** Conversational Data Analysis (ConDA) capabilities of modern assistants. The first component of **ConDABench** is a **modular, multi-agent benchmark generation framework** for generating diverse and challenging data analysis problems (Fig. 1). This modular approach is essential for capturing the heterogeneity of real-world analytical workflows, allowing us to curate evaluation problems spanning: (a) **Open-ended**

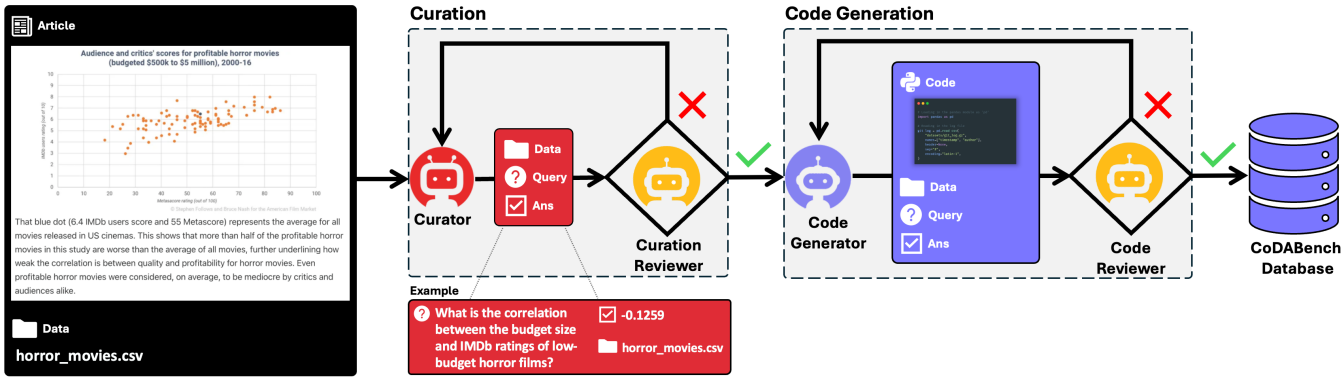


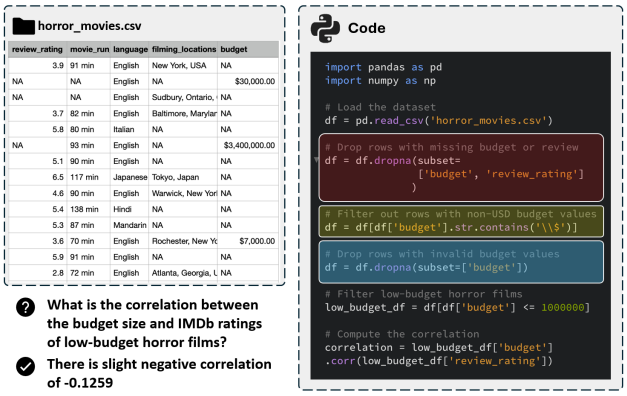
Figure 1: ConDABench Generation. From an article and data d, Curation pipeline extracts a query-answer pair (q, a) for which the code generation pipeline produces code c to support answer a. Details on agents' implementation and prompt outlines can be found on https://github.com/condabench/condabench_details.

queries, where models must explore multiple possible interpretations of an analysis problem; (b) **Projection** queries, where models must perform forecasting; (c) **Traditional question-answering** problems, where models must find a well-defined answer based on provided data. We use the above framework to generate a specific dataset, also called **ConDABench**, especially designed to assess LLMs in the context of conversational data analysis on real-world analysis problems. Finally, the third piece of the framework is a **benchmark evaluation harness** that can be used to evaluate interactive data analysis tools on **ConDABench**.

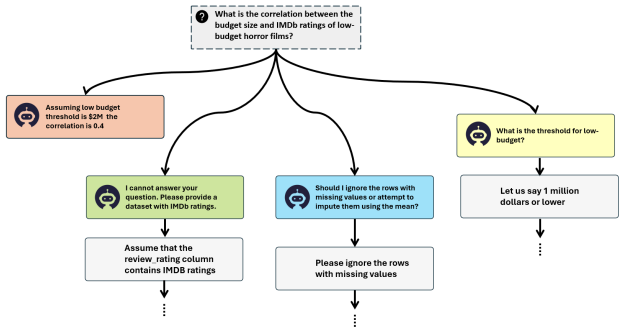
Key challenges: *How to create benchmarks that can support the automated evaluation of human-in-the-loop interactive tools?* In existing benchmarks for evaluating interactivity, such as in MT-Bench [1, 35], followup questions are initiated by the user and are static and part of the benchmark. This might not be realistic since data analysts condition their followups based on LLM's output. In addition, data analysts also *answer* questions that a conversational LLM agent may pose. How can we automate this conversation and evaluate it without human-in-the-loop? A second challenge when generating an interactive benchmark is ensuring the correctness and quality of the benchmark.

We address both of these challenges by including *code* along with the data files, the query, and the answer. A key component of our multiagent framework for benchmark generation is a code generator that iteratively finds the code that correctly explains the answer given the query and data files. The code represents the *latent reasoning* required to generate the answer from the query. The code serves two purposes. First, it provides grounding for the query-answer pair, thus ensuring the quality of the benchmark (Fig. 1). Second, it is used to implement the *User Proxy Agent*, which leverages the code to automatically answer queries that a data analysis tool may pose when given a vague or underspecified query. The User Proxy agent is a critical part of our **ConDABench** evaluation harness. The User Proxy answers questions (about the task) by referring to the code, but without revealing the code. The code allows the User Proxy to generate conversationally realistic answers for questions a data analysis tool might ask when analyzing the dataset.

We summarize the contributions of this paper as follows: (i) A **modular multi-agent architecture** for curation of *realisitc* data



(a) Running example: Data file (top-left), query-answer pair (bottom-left) and associated supporting code (right).



(b) A tree of possible conversations when a data analysis assistant is presented with the query from Figure 2a.

Figure 2: Running Example and a tree of conversation trajectories

analysis benchmarks. (ii) A **code-grounded benchmark set** generated using the above framework, with **1420** queries grounded in **338** real-world data-analysis articles, which reflects the complexities of an analyst working over the data containing a diverse

styles of queries. (iii) An **interactive evaluation harness** catered to automate evaluation of *conversational systems* using a carefully designed *User Proxy Agent*, and **systematic evaluation metrics** that measure both correctness and conversational quality of interactions. (iv) Detailed **analysis** on the conversationality and performance of various state-of-the-art LLMs (sample set on https://condabench.github.io).

2 Why is Evaluating Interactive Assistants Hard?

Supporting interactivity in the evaluation harness. Consider the query at the top of Fig. 2b. A data analysis assistant may respond to the query in one of several ways as depicted. For example, it may respond with: (a) a direct answer to the query along with a statement of assumptions, (b) a question about data cleaning steps, or (c) a question about an analysis parameter. Now, the evaluation harness must continue the conversation, but different choices can alter the assistant's final answer. For the question about "low-budget" threshold, if the threshold used to compute the ground truth answer does not match the threshold provided by the evaluation harness, the final response produced by the assistant will be incorrect despite the assistant doing "everything right". Generating follow-up responses compatible with the ground-truth answer is the primary challenge facing interactive benchmarking of data analysis assistants.

Our first key idea is to **use code that produces the expected answer** as the grounding to generate these compatible responses. We dub this component of the evaluation harness that produces responses the **user proxy**. The code that produces the expected answer is shown in Fig. 2a, and based on that code, the evaluation harness can answer question saying that the threshold is $\$1,000,000$. The example in Fig. 2a is a modified version of a benchmark task automatically generated using the **ConDABench** pipeline.

Sourcing realistic data-analysis tasks. The second major challenge in benchmarking interactive data analysis assistants is in sourcing realistic tasks where the answer is still verifiable. With recent advances in AI models, even complex multi-step tasks can be handled automatically [23]. Interactivity is relevant in specific situations, such as when the data is unclean, the task is ambiguous, or exploratory analysis is needed to fully define the scope of the analysis task. Previous attempts have been made to synthetically generate these types of tasks, for example, by artificially injecting ambiguity [16, 34], however, these still do not cover the complexity and the gamut of tasks requiring interaction.

Our second key idea is to **use real data analysis articles** (e.g., data journalism, documented notebooks, scientific literature, etc.) together with their source data to generate tasks, with the expected answers coming from the text inside the article (see Section 3.1). However, we are now left with another issue: these articles rarely provide the code that was used to produce the answer, nor do they describe the choices made for data cleaning, statistical tests, or analysis parameters. Given that these are needed to build the user proxy as described above, the second key challenge is to *reverse engineer* the analysis code from the data analysis article, query, and the ground-truth answer. In Fig. 2a, the code was automatically generated from the journalism article and the expected answer.

3 Benchmark Construction

A data analysis *problem* t in our setting is given by $t = (q, a, d, c)$, where q is a natural language *query*, a is the *answer* to the query, d is a collection of *data* files, and c is the supporting *code* used to generate the answer. The answer a can take various forms–an image plot, a dataframe, a single numerical value, or a natural language response. The code c is a (Python) program that performs data analysis, starting from the data files d, to produce artifacts such as numerical answers or plots to substantiate the answer a. As in Fig. 1, the benchmark construction workflow consists of two steps: (a) query-answer extraction, and (b) code generation. In the following, we elaborate on each step.

3.1 Query-Answer Curation

The first step in the construction of our benchmark dataset is the extraction of query-answer (q, a) pairs from various online sources. We start with sources that include both *articles* and public *datasets*, e.g., web pages, manuscripts, blogs, or Python notebooks that contain in-depth analyses of public datasets. Starting from these published articles and datasets ensures that we focus on both complex, real-world analysis tasks on un-processed datasets grounded in human-conducted analysis.

We introduce a *Curator*, an LLM-based agent, to process these articles (potentially consisting of text, code snippets, and visualizations), and to produce query-answer pairs (q, a). The Curator's task also includes generating different categories of queries for direct question-answering, open-ended, and projection tasks. We further use a *Reviewer* agent with a validation prompt to check if each query-answer pair is correctly supported by the original article.

Running Example. In a running example (Fig. 2a), the source article might have text similar to *"As opposed to other genres, the rating of low-budget horror movies have almost no relation to the budget itself with a very negligible correlation coefficient of* -0.1259*"*. From this, the Curator can generate the query-answer pair in the figure. Note that this query is inherently under-specified. Namely, the definition of "low-budget" is neither in the query nor in the article extract.

3.2 Reverse Engineering Data Analysis via Code Generation

The second step starts with the (q, a, d) tuple to generate the supporting code c. Note that this task is not the same as solving the data analysis problem since the answer a is already known while generating the code c. The purpose instead is to *reverse engineer* the data cleaning, analysis techniques, and parameters that the author of the data analysis article used to arrive at the answer. The generated code does not need to output the answer a verbatim, but only need to output sufficient numerical and visual artifacts to *support* the answer.

Running Example. In the running example from Fig. 2a, the code generation step crucially involves finding the threshold parameter for what movies are "low-budget". Picking the different values for the threshold will produce different correlation values.

The code generation pipeline consists of two interactive agents, the *Code Generator* and the *Reviewer*. The Code Generator is provided with only the query q and the dataset d (not the answer

a) and is tasked to produce code c. The Reviewer is additionally provided with the answer a and checks if the output of the code c matches a and if not, it generates feedback that is sent back to the Code Generator. This process continues iteratively until the code generator produces code which upon execution produces an answer that matches a. If this iterative process reaches a maximum bound without the appropriate code being generated, we deem that curated (q, a) pair as incorrect and filter them out. Note that both the Code Generator and the Reviewer are allowed to execute code. The Code Generator-Reviewer interaction eventually produces code c and hence, the full data analysis problem (q, a, d, c). Code generation not only is useful to produce the code c which can be used to support interactions discussed in Section 2, but also acts as a "proofing" step to ensure that the original data analysis article is correct.

Running Example. Fig. 3 shows a typical interaction between the Code Generator and Reviewer. Here, after addressing the missing values, the Code Generator could produce code with the wrong threshold for defining "low-budget". This code is executed and the Reviewer notices that the answer does not match the expected answer of -0.1259 and provides feedback that the correlation is higher than expected and that the Code Generator should try to vary the threshold. The Code Generator now regenerates the code with a different, correct threshold that produces the expected correlation value.

Answer Leakage and Audited Reviewer. Building a reviewer for the code generation step is not fully straight-forward. A naive reviewer is vulnerable to the problem of (partial) *answer leakage*. Consider a query "What is the average age in the department with the most faculty?" and an answer "The most populous department, history, has average age 49". If the code generator mistakenly determines psychology as the most populous department (potentially due to missing data cleaning steps), a naive reviewer might respond "The most populous department is history. Please revise your code". In the next iteration, the code generator may potentially hard-code "history" and compute the average age, i.e., `print(df[df.dept == "History"].age.mean())`, bypassing the actual task. We call this issue the answer leakage problem. The ideal feedback in the code generation step points the Generator in the right direction, but should not reveal the intermediate or final answer. To avoid this problem, we use a structured process using a series of tasks where: (a) the Reviewer agent first compares the output of the generated code against the answer a; (b) if they do not match, it produces feedback on the reason for the mismatch; (c) the it audits the feedback to check for answer leakage and the usefulness of the feedback before passing it back to the Code Generator.

3.3 ConDABench

We construct **ConDABench** using the procedure depicted in Sections 3.1 and 3.2, drawing from three diverse sources: TidyTuesday [30], Kaggle notebooks [15], and open-access articles from ScienceDirect [29]. These sources span informal, community-driven analyses (TidyTuesday), peer-reviewed scientific articles (ScienceDirect), and executable Python notebooks (Kaggle). Despite their differences in format, style, and domain expertise, our pipeline processes all three uniformly, without changes to prompts or artifact design, demonstrating its robustness, generalizability and extensibility. We started with 338 articles across all sources. In the curation step, we were able to curate 2398 query-answer pairs from 310 of the 338 articles (91.7%) with an average of 3.39 query-answer pairs generated per article.

The articles we were not able to generate query-answer pairs from were generally too short or contained only metadata. In the next step, the code generation pipeline was able to successfully generate 1420 of the 2398 query-answer pairs (59.2%). In a majority of the cases where code generation failed, either (a) the statement in the article did not directly follow from the data (i.e., it was potentially written based on knowledge outside the data files), or (b) the data files were updated after the articles was written (e.g., a new year's data was added after the article was written).

A detailed breakdown after each step of the workflow and summary statistics can be found in Table 2. Listed below are three categories of data analysis tasks generated using **ConDABench**.

Open-ended

Query: How does user behavior differ between new visitors and returning visitors, and how does this difference affect revenue generation?
Answer: Returning visitors have longer 'ProductRelated_Duration' (1289.42) and lower 'Bounce Rates', raising revenue to 1470, while new visitors show higher 'Bounce Rates' and 'Exit Rates', reducing revenue to 422.

Projection

Query: What is the projected total yield of the solar plants for the next year?
Answer: The projected total yield for next year is 2672270651.28 units for Plant 4135001 and 2555960912.01 units for Plant 4136001.

QA

Query: Which month has the highest number of visitors on this site?
Answer: May has the highest number of visitors on the site, with 3364 visitors.

We further split our benchmark into *shallow* and *deep* categories based on task depth, defined as the number of generator–reviewer iterations required to reach a correct solution. This reflects the number of choices made during data analysis. Tasks that converge in fewer than 3 iterations are labeled *shallow*, while those requiring 3 or more iterations are labeled *deep*.

To validate the benchmark correctness, we sampled 275 data points (approx. 20% of the entire set) and distributed them among human experts for verification. Based on their evaluations, we found that **92.73%** of samples did not require any correction. The error rate in our benchmark is on par with established datasets such as Imagenet [6], QuickDraw [4] and CIFAR [17], which have

conservatively reported error rates ranging between 4%-10% [25, 26].

Table 1: Summary statistics of ConDABench.

Metric	Value
size	1420
#data files	1855
avg. #data files per query	1.31
% of queries taking > 1 data files	17.25
avg. data file size per query	4.24MB
# viz. based queries	405
avg. code length	24.02
avg. code exe. time	2.01 sec
avg. # libraries per code	1.56
shallow (tasks converging in <3 iters)	1317
deep (tasks converging in ≥ 3 iters)	103

Table 2: Stepwise statistics for benchmark construction. *Curated* is the number of initial pairs, and *Code Gen. Passed* denotes the pairs with correct code that passed all checks.

Source	Articles	Type	Curated	Code Gen. Passed
TidyTuesday	283	qa	1004	551
		open-ended	899	560
		projection	6	4
Kaggle	30	qa	148	81
		projection	10	4
Open-Access	25	qa	120	75
		open-ended	85	47
		projection	7	3
Total	338		2398	**1420**

4 Interaction-Capable Evaluation Harness and Evaluation Metrics

The evaluation harness automates the evaluation of an external data analysis assistant. The harness measures the assistant's ability to provide accurate responses while engaging in meaningful dialogue. Since the goal is to evaluate conversational systems, the evaluation harness needs a user proxy that can interact with the system-under-test.

4.1 The User Proxy

The *User Proxy* agent simulates a real user to automate conversations between an external data analysis assistant and a user. This agent has access to the query q, dataset d, and the supporting code c, but not the answer a (to avoid answer leakage). The User Proxy agent responds to the system-under-test to provide the necessary information, disambiguation, or clarifications when *specifically asked*.

Running Example. Fig. 2b shows the interaction between the User Proxy and a model-under-test for the running example task from Fig. 1. Here, the model asks a few clarification questions before finally responding to the query. First, the model asks about what strategy should be used to handle missing data, to ignore

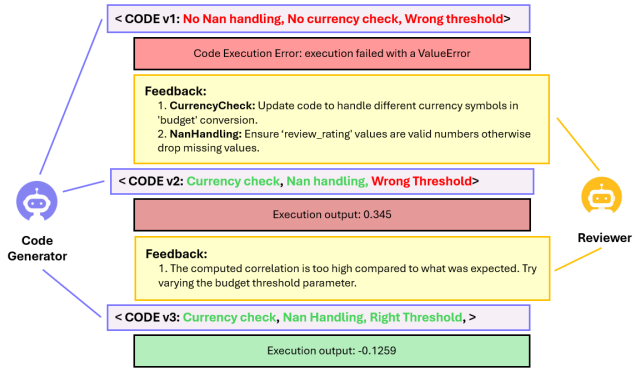


Figure 3: Code Generator-Reviewer.

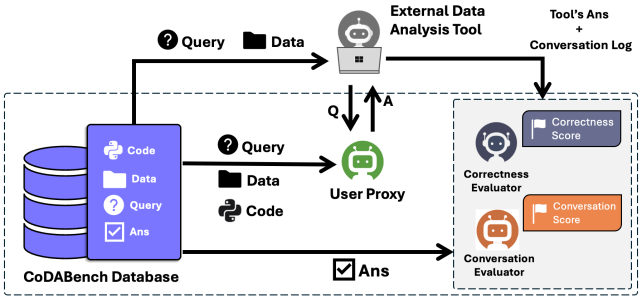


Figure 4: Evaluation Framework.

missing values or to impute them, and the User Proxy appropriately answers based on the supporting code. The next is about what "low-budget" means in this setting: here, the User Proxy can refer to the supporting code c generated by the code generation step to answer that low-budget is less than $1,000,000. Note that answering this question without the supporting code will very likely lead to diverse assumptions for a given ambiguity, making standardized evaluation difficult.

The User Proxy carries a risk similar to answer leakage. While the User Proxy does not directly have access to the answer a, it has access to the supporting code. Hence, it may proactively provide details of analysis techniques or parameters to use even if the model-under-test does not ask for them. For example, a naive User Proxy may provide the threshold for "low-budget" in the running example even when not asked for; this fails to test whether the system-under-test can interactively clarify and disambiguate the data analysis task. We again use multi-step tasking to avoid this leakage: (a) the User Proxy first classifies the model-under-test's utterance as either an answer, a clarification or a confirmation; (b) then a candidate response is generated using the code to provide any clarification if needed; The initial classification step ensures that the User Proxy does not forcefully correct the model-under-test when it responds using an incorrect analysis technique. Hence, the conversation ends when the external DA assistant provides a final answer irrespective of whether it is correct. The User Proxy thus standardizes the evaluation process to ensure consistency and fairness.

4.2 Evaluation Metrics: Correctness and Conversation

Building on previous work on conversational analysis [2, 19], we design a comprehensive evaluation framework that integrates metrics to assess both the correctness and the conversational capabilities of a data analysis (DA) assistant.

Response Correctness Assessment. Our evaluation methodology utilizes an Evaluator agent [20] to assess the overall correctness of the DA assistant's responses, i.e., if the assistant's response matches the expected answer a. This approach is essential for addressing the diverse nature of data analysis problems, which often include both structured and unstructured answers. The correctness assessment evaluates the relevance, coherence, informativeness, and exactness of the DA's responses. The correctness evaluator first identifies and extracts potential answers from the DA assistant's response, ensuring that the information provided is meaningful and aligned with the user's intent. This step is crucial, as verbose or ambiguous responses can compromise the evaluation process. The evaluator then compares the extracted answer with the correct solution across different modalities - including textual answers, numerical outputs, and visual representations (e.g., generated plots compared to textual descriptions). This ensures a robust correctness match by accounting for variations in how answers are expressed. To validate reliability, we compare the evaluator's judgments with a human-annotated reference set (section 5), observing strong correlation (Pearson correlation: **0.748**, Match accuracy: **88.11%**) with human evaluation.

Conversation Quality Evaluation. Beyond assessing answer correctness, we also evaluate the overall conversation quality of DA assistants. Taking inspiration from RUBICON [2], we design a metric that assigns scores based on Satisfiable (SAT) and Dissatisfiable (DSAT) rubrics. These rubrics capture key characteristics of effective dialogues, reflecting widely accepted notions of conversational effectiveness like the Gricean Maxims [7]. To build these rubrics, we conducted an exploratory study where human annotators provided reasons for accepting or rejecting specific conversations. After discussions, a consensus emerged on the essential factors that signify a well-conducted or poorly handled conversation in the domain of DA. Similar to RUBICON [2], we prompt the LLM to rate the conversation on each rubric on a three-point Likert scale.

SAT Rubrics: Did the assistant

- **[S.1]** Seek clarification regarding the user's query, dataset, or its planned approach?
- **[S.2]** Explain the steps taken to arrive at the solution?
- **[S.3]** Offer an analytical insight or a meaningful conclusion based on the obtained results?

DSAT Rubrics: Did the assistant

- **[D.1]** Repeat its questions or responses unnecessarily?
- **[D.2]** Fail to follow a direct instruction from the user or execute an unintended action?
- **[D.3]** Generate unnecessary responses (or computation steps) not required to reach the final answer?

To convert these multi-dimensional SAT and DSAT scores into a single binary judgement of conversation quality, we train an off-the-shelf regression model on a sample of data annotated by the same experts. We construct a balanced training set by sampling 39 good and 41 bad conversations along with their corresponding rubric-score vectors (each entry in $-1, 0, 1$ to represent *DisAgree*, *Neutral* and *Agree* respectively). We evaluate several regression models on this data. Logistic Regression achieves the best performance, attaining an F1-score of **0.75**. We therefore adopt this model for aggregating rubric scores during evaluation and producing the final good/bad conversation quality decision.

5 Validation by Humans

Before sharing key insights through our evaluation, we first describe the human annotation exercise that underpins the reliability of the evaluation methods used in this work. We enlisted three domain experts in Data Analysis to review conversation simulations between the user-proxy and the DA tool.. We randomly sample equal distribution of simulations from GPT-4o and GPT-3.5-Turbo in the Assistant's API framework, and distribute it among the annotators, keeping an overlap percentage of 50% out of the 50 instances that were rolled out. The annotators were required to scale the conversation between the DA tool and the user-proxy on a 3-point scale, where -1 represented "bad" and 1 represented "good" conversation quality. They were also required to annotate if the response from the DA tool was accurate in solving the query.

With this exercise, we performed the first round of inter-rater agreement and obtained a Pearson's correlation coefficient of 56.78. Due to a lower agreement, we invited the annotators on a discussion on the scenarios where they had a mismatch in opinion. After the first round of discussion, we re-iterated the annotation process with similar sample size. After the second round, the inter-rater agreement gave **79.24**, which was above our acceptable threshold. We finally obtained a hand-annotated set of 143 instances. These annotations were used as ground truth labels to validate and improve the quality of our evaluation metrics, attempting to aligning them with human judgment.

6 Evaluations over Data Analysis Frameworks

We evaluate various foundational models from OpenAI, DeepSeek, Mistral and Google over **ConDABench** across different Data Analysis Frameworks. Appendix **??** lists the models and checkpoints used. To support different frontier models with varying constraints and ensure fairness in evaluation, we evaluate the models using *Assistants API*, *Tool Call* and *InfiAgent-Conv*.

1. **Assistants API**: OpenAI's native code-interpreter support [27].
2. **Tool Call**: A bare-bones setup utilizing model's built-in function-calling implementation with a code-execution tool. The code-execution tool take the code as a single argument and return the execution output to the model after running it in a python sandbox.
3. **InfiAgent-Conv**: A ReAct [33] style data-analysis agent adapted from InfiAgent [11]. Specifically, we make minimal modifications to the InfiAgent setup to enable conversation history management. To enable parity, we used the same code-execution tool as the the Function Calling setup for code-execution.

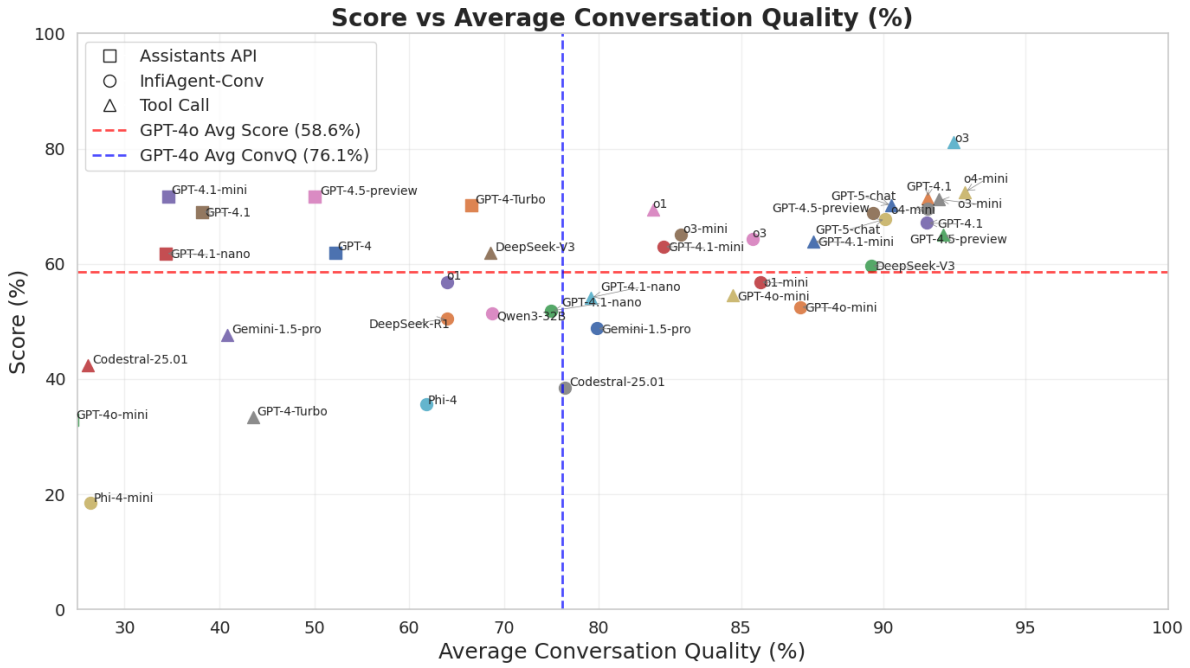


Figure 5: Performance vs Conversation Quality (Score vs ConvQ) Across Frameworks. Each point is a model–framework pair (squares: Assistants API; circles: InfiAgent-Conv; triangles: Tool Call). GPT-4o is not a contestant in our evaluations; it serves only as a fixed reference point. The dashed red and blue lines mark GPT-4o's average Score (58.6%) and ConvQ (76.1%), respectively. Models in the upper-right quadrant exceed this GPT-4o reference on both correctness and conversation quality, indicating they reach the correct solution while maintaining high conversational quality.

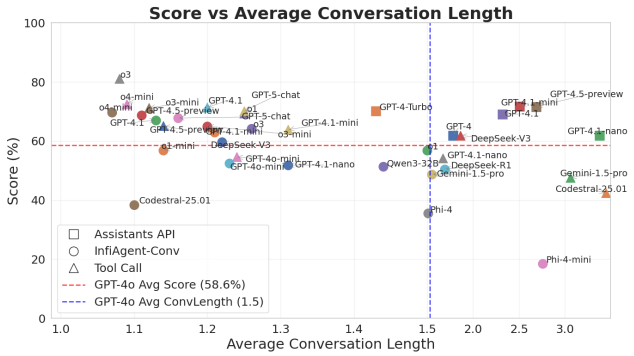


Figure 6: Performance vs Conversation Length.

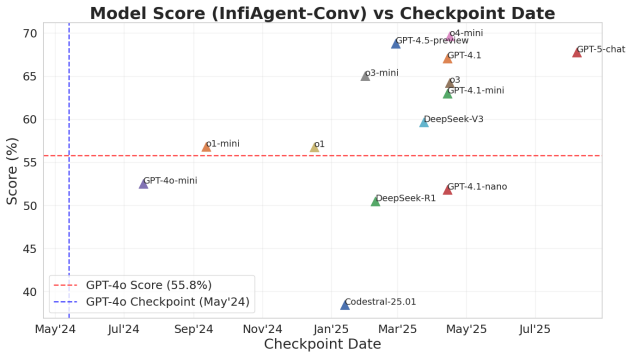


Figure 7: Performance vs Model Launch Dates.

The User Proxy agent and LLM-graded evaluation metrics are powered by the `GPT-4o-2024-05-13` model. To avoid model bias and circularity, GPT-4o is never included as a contestant in our evaluations and is only used to instantiate the user-proxy and the grader. We present three plots: (i) performance vs. conversation quality (Fig. 5), (ii) performance vs. conversation length (Fig. 6), and (iii) performance vs. model launch dates (Fig. 7), to assess these models (across frameworks) on ConDA problems.

6.1 Trends Across Models

We found **o3** Tool Calling to be the best performing model on **ConDABench**, beating the second-best configuration by 8.6% performance score. Even so, there is significant headway in the *deep* subset, with a 54.37 score%. Further analysis across model families reveals clear distinctions across model types and frameworks which has been discussed in details below.

Reasoning vs. Non-Reasoning Models. Reasoning models (o-family) showed stronger contextualization of open-ended queries than non-reasoning models (GPT-series, Gemini). For example, in a

medical query, o-family models linked complex medicines with orphan drugs, while Gemini flagged missing context. They also maintained better temporal consistency across extended interactions, handling tasks like imputations, pivots, and joins more effectively. In contrast, non-reasoning models often gave partial answers or misjudged granularity.

Open-Source vs. Proprietary Models. Open-source models (such as DeepSeekV3 and Codestral) had limited tool call capabilities, often encountering data handling errors. Proprietary models (like the OpenAI and Gemini models) were more robust, executing tool-assisted tasks reliably with fewer mistakes. Fine-tuning for tool use and error recovery was observed to be more mature in proprietary models, giving them an edge on a complex, multi-step benchmark tasks like ours.

Reasoning Effort. We observed a trade-off where lower reasoning effort settings yields more straightforward (but less precise) answers, while higher reasoning effort settings increase accuracy at the cost of verbosity and unnecessary complexity (e.g., extraneous code), hindering concise responses. In our experiments (Table 3), *medium* research effort tended to give the best performance.

Table 3: Evaluation of o4-mini on Tool Calling Framework across different reasoning efforts. Results demonstrate that even high reasoning efforts are unable to follow conversational approach towards solving such ambiguous and complex data analysis tasks.

o4-mini (Tool)	Overall		Shallow		Deep	
Reasoning effort	Score	ConvQ	Score	ConvQ	Score	ConvQ
low	65.02	87.73	66.51	87.85	**45.54**	86.00
medium	**72.46**	**92.87**	**74.79**	**92.62**	42.72	**96.12**
high	67.18	89.23	69.1	89.60	42.72	84.47

6.2 Trends Across Frameworks

To compare the performance of the different ADA frameworks, we analyze the differences in their performance over the models that are common between them.

Performance across frameworks. We observe that the Assistants API based implementation often performs better over the other implementations in terms of answer correctness but fared relatively poorly in conversational quality (Fig 5). Agents in the Assistants framework generated verbose explanations (characterized by a high S.3 score in Table 4) and often repeated their internal steps to the user, impeding conversational naturalness. In contrast to the Assistants API setup, the InfiAgent-conv and Tool calling framework yielded concise responses, presenting only the pertinent information, but often lacked in critical analysis and failed to perform as well on deep and open-ended queries. The only exception to this was observed when these frameworks were used with reasoning models, which assisted by their test-time compute capabilities fared especially well in contextualizing open ended queries.

Assistants API as a flavour of test time compute. The Assistants API implementations often mirrors how reasoning models behave in the tool calling with high answer correctness scores but poorer conversation quality. We argue that this can be explained by the design of the assistant API, which procedurally generates code, executes it and reflect upon it iteratively during inference - resembling the *test-time compute* paradigm observed to help reasoning models.

6.3 Conversational Analysis

One of the unique features of **ConDABench** is its ability to evaluate how well the models sustain long yet effective conversations. In this section we utilize this to draw detailed insights about the conversational capabilities of various LLMs.

Longer Conversations are not always better. Fig. 6 shows the trend in the number of interactions a model has with the User Proxy before reaching an answer. Models located toward the top-left (o3 and o4-mini) achieve high performance with fewer conversational turns, indicating efficient and effective reasoning capabilities. In contrast, models such as GPT-4.1 and GPT-4.5-preview achieve similarly high accuracies but with substantially longer conversations. This suggests that some models, particularly those not explicitly optimized for stepwise reasoning, may rely more on extended user engagement or clarification to arrive at the correct answer. While such engagement could be beneficial in certain applications, our results indicate some potential diminishing returns: excessive conversational length does not necessarily yield better accuracy, as evidenced by Gemini-1.5-pro, which features longer sessions but lower overall scores. These findings suggest an ongoing trend: improving LLMs is moving toward making them *do more with less interaction*, rather than making them *effective collaborators that can engage in fruitful long conversations*. In fact, we observe a clear "Pareto frontier" where higher success (in newer models) is strongly correlated with shorter interactions. Namely, with each new generation, models appear to move "up and left" on these dimensions, with the exception of the evolution from GPT-3.5 to GPT-4. This indicates that while models are improving at solving more problems, they are not necessarily getting better at sustaining long conversations. There remains significant work to be done in building models that can truly collaborate and complete lengthy tasks. More details can be found in Table 5.

Richer analysis using conversational rubrics. Our rubric-based conversational eval also enables an intricate diagnostic of evaluation trends across models (Table 4). For instance, Models like GPT-4o-mini (Asst API), Codestral-25.01 (Tool Call) and Gemini-1.5-pro (Tool Call) report high margins of S.1 values (30%), suggesting that they are quite proactive in understanding the user intent by asking for clarifications. While this helps them remove ambiguity a high D.3 values (exceeding trends by 20%) suggests that they often get lost in conversation and forget major nuances related to problem solving. Similarly for Codestral-25.01, we report low accuracy scores since these models prefer generating programs as solution instead of calling tools to execute them. The high S.2 value (93.02%) further confirms this behavior.

7 Related Works

Evaluating LLM-driven data analysis agents presents unique challenges due to the open-ended and interactive nature of real-world

Table 4: This figure presents a breakdown of the Conversation Quality Evaluation by reporting the hit rate (in %) for each SAT and DSAT rubric. Hit rates are computed by mapping 'Disagree', 'Neutral', and 'Agree' responses to 0, 0.5, and 1, respectively, for each rubric applied to a conversation. Rather than aggregating rubric scores within each conversation, we average the score for each rubric across all conversations to capture how often a model satisfies that rubric. Column notations correspond to the rubric definitions in section 4.2. Higher SAT hit rates and lower DSAT hit rates indicate better performance.

Framework	Model	SAT			DSAT		
		S.1	S.2	S.3	D.1	D.2	D.3
Asst API	GPT-4	45.76	89.61	86.86	0.57	29.96	53.43
	GPT-4-Turbo	25.93	92.78	89.32	0.53	30.23	48.80
	GPT-4o-mini	60.42	89.82	72.72	0.49	51.02	73.43
	GPT-4o	24.48	89.62	85.36	0.60	32.87	56.01
	GPT-4.1-nano	61.33	95.06	79.31	0.42	31.25	56.50
	GPT-4.1-mini	59.72	98.10	87.50	0.35	26.90	54.79
	GPT-4.1	56.62	97.67	82.24	0.42	27.31	51.13
	GPT-4.5-preview	27.11	93.48	85.35	0.75	27.29	46.38
Tool Call	GPT-4-Turbo	33.06	25.49	57.78	1.02	46.51	68.31
	GPT-4o-mini	6.44	14.86	74.61	0.99	33.49	49.01
	GPT-4o	5.70	13.56	79.58	0.85	38.13	52.36
	GPT-4.1-nano	17.69	16.84	76.91	1.13	34.64	55.12
	GPT-4.1-mini	11.95	10.89	76.63	1.49	30.83	50.35
	GPT-4.1	8.10	12.46	84.54	1.06	29.61	45.67
	GPT-4.5-preview	5.60	7.96	80.16	1.09	31.64	47.89
	Codestral-25.01	52.57	93.02	61.28	0.35	42.88	68.90
	Gemini-1.5-pro	69.32	24.02	74.34	1.43	40.13	64.40
	DeepSeek-V3	41.30	30.90	84.67	1.30	30.87	49.47
	o1	8.25	17.62	82.34	1.23	38.49	52.75
	o3-mini	5.50	42.10	91.11	0.49	32.19	49.01
	o4-mini	3.28	16.87	84.12	1.16	31.69	46.51
	o3	3.31	29.47	86.94	1.16	31.13	46.44
	GPT-5-chat	7.12	16.54	85.68	1.02	29.06	46.16
InfiAgent-Conv	GPT-4o-mini	1.73	8.15	75.95	0.95	35.90	47.47
	GPT-4o	1.83	13.35	75.30	0.60	37.77	49.12
	GPT-4.1-nano	4.79	10.04	73.38	0.70	42.82	53.94
	GPT-4.1-mini	9.63	16.09	82.04	1.48	35.43	50.35
	GPT-4.1	4.23	13.57	84.64	1.23	31.89	49.47
	GPT-4.5-preview	2.89	7.89	80.42	0.81	34.51	46.69
	Codestral-25.01	9.44	20.07	76.80	0.67	39.72	52.25
	Qwen3-32B	15.83	43.94	78.95	1.09	41.75	56.38
	Phi-4	13.67	38.91	75.92	0.85	43.47	58.51
	Phi-4-mini	47.95	52.16	51.91	2.08	61.20	81.17
	Gemini-1.5-pro	10.61	9.41	73.43	0.70	40.13	51.59
	DeepSeek-R1	21.15	55.32	80.21	0.62	36.80	53.08
	DeepSeek-V3	4.09	9.80	83.58	1.06	35.52	49.08
	o1-mini	5.58	8.72	82.82	0.67	39.59	51.45
	o1	15.00	43.05	75.76	1.02	37.16	55.01
	o3-mini	5.77	29.94	79.87	1.52	31.03	43.28
	o3	9.44	16.09	79.19	1.23	31.62	48.45
	o4-mini	3.45	11.65	81.48	1.09	32.39	45.85
	GPT-5-chat	3.98	13.77	87.82	1.55	35.88	50.85

tasks. While several benchmarks exist, they often fall short of addressing the full complexity of such scenarios, which include unclean data, multi-step processes, and the need for conversational interaction. Here, we review prior work to position our proposed benchmark within the broader landscape.

7.1 Table Question Answering

Table-based question answering has been a central focus of several benchmarks. WikiTableQuestions [28] evaluates semantic parsing over semi-structured tables, emphasizing challenges like open-ended relations and logical compositionality. FeTaQA [24] expands this domain by introducing free-form table question answering, which demands reasoning over structured sources and the ability to generate coherent and informative answers. These benchmarks highlight the foundational challenges of understanding and reasoning over tabular data. TabFact [5] shifts the focus to fact verification, testing both linguistic and symbolic reasoning over tables paired with natural language statements. TableBench [32] further bridges academic and real-world needs by evaluating tasks such as numerical reasoning and data visualization, emphasizing practical applications in industrial contexts. Collectively, these works underline the limitations of traditional table QA systems in addressing the dynamic and interactive nature of real-world data analysis.

Table 5: Compares models based on the average length of conversations they engage in with the user-proxy while attempting to solve a question. *Avg* represents the average conversation length on all queries in ConDABench and demonstrates a model's orientation towards engaging in lengthy or short conversations when attempting a solution. *Corr* represents the average conversation length for those queries which the model solved correctly. Lower values (good) demonstrates that achieved success with minimum number of interactions. *Incorr* represents the average conversation length over the subset where the model failed. Lower values (bad) suggests unsuccessful attempts at the solution without gaining clarity on user's preference.

Framework	Model	Overall			Shallow			Deep		
		Avg	Corr	Incorr	Avg	Corr	Incorr	Avg	Corr	Incorr
Asst API	GPT-4	1.78	1.60	2.08	1.75	1.58	2.04	2.29	2.08	2.42
	GPT-4-Turbo	1.43	1.32	1.71	1.40	1.28	1.71	1.82	1.90	1.75
	GPT-4o-mini	4.87	2.21	6.16	4.90	2.21	6.29	4.36	2.42	4.83
	GPT-4o	2.35	1.38	3.86	2.33	1.37	3.96	2.60	1.68	3.14
	GPT-4.1-nano	3.38	2.09	5.48	3.30	2.09	5.48	4.44	2.16	5.46
	GPT-4.1-mini	2.51	2.16	3.27	2.39	2.08	3.12	4.00	3.50	4.71
	GPT-4.1	2.32	1.90	3.25	2.27	1.89	3.22	2.86	2.13	3.43
	GPT-4.5-preview	2.69	1.81	4.92	2.66	1.80	5.03	3.02	2.00	4.17
Tool Call	GPT-4-Turbo	7.20	1.72	9.95	7.14	1.71	10.06	7.95	2.21	8.85
	GPT-4o-mini	1.24	1.16	1.34	1.23	1.16	1.33	1.39	1.35	1.41
	GPT-4o	1.16	1.13	1.21	1.15	1.10	1.21	1.37	1.80	1.19
	GPT-4.1-nano	1.67	1.37	2.02	1.64	1.35	2.01	2.06	1.88	2.14
	GPT-4.1-mini	1.31	1.22	1.46	1.30	1.21	1.46	1.48	1.43	1.50
	GPT-4.1	1.20	1.15	1.30	1.18	1.15	1.27	1.40	1.31	1.47
	GPT-4.5-preview	1.14	1.09	1.23	1.13	1.09	1.21	1.30	1.11	1.41
	Codestral-25.01	3.45	3.08	3.72	3.39	3.08	3.65	4.17	3.32	4.41
	Gemini-1.5-pro	3.06	2.56	3.51	2.99	2.54	3.43	3.95	3.26	4.15
	DeepSeek-V3	1.86	1.75	2.05	1.83	1.72	2.03	2.26	2.37	2.20
	o1	1.25	1.19	1.38	1.24	1.19	1.37	1.33	1.21	1.45
	o3-mini	1.12	1.09	1.18	1.10	1.08	1.15	1.32	1.28	1.36
	o4-mini	1.09	1.06	1.15	1.08	1.06	1.15	1.15	1.16	1.14
	o3	1.08	1.06	1.15	1.07	1.05	1.17	1.13	1.16	1.09
	GPT-5-chat	1.25	1.21	1.37	1.24	1.20	1.34	1.47	1.35	1.56
InfiAgent-Conv	GPT-4o-mini	1.23	1.02	1.47	1.19	1.02	1.40	1.77	1.00	2.04
	GPT-4o	1.07	1.05	1.10	1.06	1.04	1.08	1.17	1.14	1.19
	GPT-4.1-nano	1.31	1.13	1.50	1.23	1.12	1.37	2.27	1.43	2.59
	GPT-4.1-mini	1.21	1.19	1.26	1.21	1.19	1.26	1.24	1.20	1.28
	GPT-4.1	1.13	1.09	1.20	1.11	1.08	1.18	1.32	1.36	1.30
	GPT-4.5-preview	1.11	1.07	1.21	1.08	1.06	1.13	1.47	1.10	1.69
	Codestral-25.01	1.10	1.08	1.11	1.09	1.08	1.10	1.22	1.21	1.23
	Qwen3-32B	1.44	1.34	1.54	1.43	1.34	1.52	1.59	1.42	1.67
	Phi-4	1.51	1.44	1.55	1.50	1.43	1.55	1.59	1.64	1.58
	Phi-4-mini	2.76	2.50	2.81	2.73	2.48	2.79	3.09	2.92	3.11
	Gemini-1.5-pro	1.55	1.22	1.86	1.53	1.19	1.88	1.76	1.96	1.69
	DeepSeek-R1	1.69	1.61	1.78	1.69	1.61	1.77	1.75	1.57	1.82
	DeepSeek-V3	1.22	1.08	1.42	1.20	1.07	1.42	1.39	1.19	1.47
	o1-mini	1.14	1.11	1.18	1.13	1.09	1.17	1.28	1.47	1.21
	o1	1.50	1.43	1.60	1.49	1.42	1.57	1.71	1.50	1.81
	o3-mini	1.20	1.12	1.34	1.19	1.12	1.33	1.33	1.22	1.41
	o3	1.26	1.22	1.35	1.25	1.21	1.34	1.39	1.33	1.43
	o4-mini	1.07	1.03	1.16	1.05	1.02	1.12	1.27	1.05	1.41
	GPT-5-chat	1.16	1.12	1.24	1.13	1.09	1.22	1.54	1.74	1.36

7.2 Data Analysis Benchmarks

Benchmarks targeting data analysis extend beyond static question answering to encompass interactive and multi-step tasks. TAPILOT-CROSSING [18] employs a multi-agent system to simulate real-world data analysis scenarios, evaluating adaptability to ambiguous user intents and the ability to generate actionable code. This simulation-based approach provides valuable insights but relies on a fixed set of scenarios, limiting its generalizability.

InfiAgent-DABench [11] builds on this by focusing on end-to-end task completion, emphasizing interaction with execution environments to solve complex problems. DSBench [14] incorporates tasks that span both data analysis and modeling, challenging systems to

Table 6: Comparison to Other Benchmarks

Benchmark	# Datasets	# Queries	Conversational Eval	Open-Ended Queries	Free-form Answer Support	Multi Dataset Queries	Unclean Dataset
WikiTableQuestions	2108	22033	×	×	×	×	×
InfiAgentDABench	52	257	×	×	✓	×	✓
TAPilot-Crossing	5	1024	×	✓	✓	×	×
ConDABench (ours)	1855	1420	✓	✓	✓	✓	✓

handle long contexts, multi-table structures, and large datasets. Similarly, DA-Code [13] evaluates programming-intensive tasks, emphasizing step-by-step reasoning, data wrangling, and exploratory data analysis. These benchmarks address specific facets of data analysis but lack comprehensive coverage of conversational and iterative workflows.

7.3 Interactive and Conversational Benchmarks

The evaluation of interactive and conversational capabilities in benchmarks has gained traction in recent years. BLADE [8] examines agents' ability to decompose tasks, resolve ambiguities, and handle unclean data, emphasizing user engagement and feedback incorporation. MLAgentBench [12] focuses on machine learning engineering, evaluating systems on iterative refinement and adaptability to new tasks, which are crucial for long-term planning in complex workflows. SPREADSHEETBENCH [22] highlights challenges in spreadsheet manipulation, testing systems on complex instructions and flexible data organization. The Data Interpreter Benchmark [10] introduces hierarchical graph modeling to dynamically break down problems, emphasizing real-time adaptation and iterative refinement. These works collectively underscore the need for benchmarks that address conversationality, multi-step processes, and the integration of diverse data sources.

Table 6 provides a comparison with other popular benchmarks. Notably, **ConDABench** is the first benchmark that can simulate and evaluate conversations for data analysis tasks. Coupled with a highly diverse set of realistic queries and datasets, it provides a comprehensive framework for evaluation of conversational data analysis agents.

8 Discussion and Conclusion

We presented **ConDABench**, a modular multi-agent architecture designed for synthesizing benchmarks in Conversational Data Analysis (ConDA). By grounding each task in human-authored articles and validating them via synthesized code, **ConDABench** produces accurate, reliable benchmarks that closely mirror complex, real-world data analysis challenges.

A central challenge in synthetic benchmark generation with large language models is guaranteeing the correctness of reference answers. Our pipeline explicitly addresses this by requiring two unlikely errors for an incorrect query-answer pair to persist after the code validation step: (1) The original article contains inaccuracies or the curator agent hallucinates (with the curator reviewer failing to detect it); and (2) The code generation pipeline successfully produces valid data analysis code that outputs the erroneous answer. Empirically, the second failure is extremely rare, incorrect answers are almost always caught when running code. Manual inspection found only a small fraction of incorrect cases, mostly traceable to errors in the source articles. While **ConDABench** is a step forward in evaluating data assistants under more realistic conditions, it does not yet encompass the full breadth of real-world scenarios. For example, it does not consider cases where users change their intent mid-conversation or pose exploratory queries, such as "Give me some insights about this data." Addressing such scenarios remains an exciting avenue for future work.

Finally, our evaluation harness, powered by a code-grounded user proxy, systematically assesses models not just on correctness, but also on their ability to manage ambiguity, uncertainty, and extended interactions. While recent reasoning models show improved efficiency and contextual understanding, robust collaboration across long, complex dialogues continues to be an open research challenge.

References

[1] Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024. MT-Bench-101: A Fine-Grained Benchmark for Evaluating Large Language Models in Multi-Turn Dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 7421–7454. https://doi.org/10.18653/v1/2024.acl-long.401

[2] Param Biyani, Yasharth Bajpai, Arjun Radhakrishna, Gustavo Soares, and Sumit Gulwani. 2024. RUBICON: Rubric-Based Evaluation of Domain-Specific Human AI Conversations. In *Proceedings of the 1st ACM International Conference on AI-Powered Software*. 161–169.

[3] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature* 624, 7992 (2023), 570–578.

[4] Salman Cheema, Sumit Gulwani, and Joseph LaViola. 2012. QuickDraw: improving drawing experience for geometric diagrams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 1037–1064. https://doi.org/10.1145/2207676.2208550

[5] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164* (2019).

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255. https://doi.org/10.1109/CVPR.2009.5206848

[7] Paul Grice. 1991. *Studies in the Way of Words*. Harvard University Press.

[8] Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, et al. 2024. Blade: Benchmarking language model agents for data-driven science. *arXiv preprint arXiv:2408.09667* (2024).

[9] Stefan Harrer. 2023. Attention is not all you need: the complicated case of ethically using large language models in healthcare and medicine. *EBioMedicine* 90 (2023).

[10] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. 2024. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679* (2024).

[11] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. 2024. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507* (2024).

[12] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Benchmarking large language models as AI research agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

[13] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. 2024. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models. *arXiv preprint arXiv:2410.07331* (2024).

[14] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. DSBench: How Far Are Data Science Agents to Becoming Data Science Experts? *arXiv preprint arXiv:2409.07703* (2024).

[15] Kaggle. 2024. Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/

[16] Hyuhng Joon Kim, Youna Kim, Cheonbok Park, Junyeob Kim, Choonghyun Park, Kang Min Yoo, Sang-goo Lee, and Taeuk Kim. 2024. Aligning Language Models to Explicitly Handle Ambiguity. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1989–2007. https://doi.org/10.18653/v1/2024.emnlp-main.119

[17] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[18] Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. 2024. Tapilot-Crossing: Benchmarking and Evolving LLMs Towards Interactive Data Analysis Agents. *arXiv preprint arXiv:2403.05307* (2024).

[19] Ying-Chun Lin, Jennifer Neville, Jack Stokes, Longqi Yang, Tara Safavi, Mengting Wan, Scott Counts, Siddharth Suri, Reid Andersen, Xiaofeng Xu, Deepak Gupta, Sujay Kumar Jauhar, Xia Song, Georg Buscher, Saurabh Tiwary, Brent Hecht, and Jaime Teevan. 2024. Interpretable User Satisfaction Estimation for Conversational Systems with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 11100–11115. https://doi.org/10.18653/v1/2024.acl-long.598

[20] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv:2303.16634 [cs.CL] https://arxiv.org/abs/2303.16634

[21] Andrew Low and Z. Yasemin Kalender. 2023. Data Dialogue with ChatGPT: Using Code Interpreter to Simulate and Analyse Experimental Data. arXiv:2311.12415 [physics.ed-ph] https://arxiv.org/abs/2311.12415

[22] 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. *arXiv preprint arXiv:2406.14991* (2024).

[23] Friso Kingma Martin Iglesias, Alex Egg. 2025. Data Agent Benchmark for Multi-step Reasoning (DABstep). https://www.adyen.com/knowledge-hub/data-agent-benchmark-for-multi-step-reasoning-dabstep

[24] Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics* 10 (2022), 35–49.

[25] Curtis Northcutt, Lu Jiang, and Isaac Chuang. 2021. Confident Learning: Estimating Uncertainty in Dataset Labels. *J. Artif. Int. Res.* 70 (May 2021), 1373–1411. https://doi.org/10.1613/jair.1.12125

[26] Curtis G. Northcutt, Anish Athalye, and Jonas Mueller. 2021. Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks. arXiv:2103.14749 [stat.ML] https://arxiv.org/abs/2103.14749

[27] OpenAI. 2024. Assistants Api Overview. https://platform.openai.com/docs/assistants

[28] Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305* (2015).

[29] ScienceDirect. 2024. Science Direct: science, health and medical research. https://www.sciencedirect.com/

[30] TidyTuesday. 2024. Tidy Tuesday: A weekly social data project. https://tidytues.day

[31] Maxim Vidgof, Stefan Bachhofner, and Jan Mendling. 2023. Large language models for business process management: Opportunities and challenges. In *International Conference on Business Process Management*. Springer, 107–123.

[32] Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. 2024. TableBench: A Comprehensive and Complex Benchmark for Table Question Answering. *arXiv preprint arXiv:2408.09174* (2024).

[33] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

[34] Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. 2024. CLAMBER: A Benchmark of Identifying and Clarifying Ambiguous Information Needs in Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 10746–10766. https://doi.org/10.18653/v1/2024.acl-long.578

[35] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. arXiv:2306.05685 [cs.CL] https://arxiv.org/abs/2306.05685