DABstep: Data Agent Benchmark for Multi-step Reasoning

Alex Egg[1,♣,*], **Martin Iglesias Goyanes**[1,♣], **Friso Kingma**[1,♣], **Andreu Mora**[1,♣], **Leandro von Werra**[2], **Thomas Wolf**[2]
[1]Adyen [2]Hugging Face
{alex.egg,martin.iglesiasgoyanes,friso.kingma,andreu.mora}@adyen.com

Abstract

We introduce **DABstep**, a novel benchmark for evaluating AI agents on realistic multi-step data analysis tasks. DABstep comprises over 450 real-world challenges derived from a financial analytics platform, requiring models to combine code-based data processing with contextual reasoning over heterogeneous documentation. Each task demands an iterative, multi-step problem-solving approach, testing capabilities in data manipulation, cross-referencing multiple sources, and precise result reporting. The benchmark provides a factoid-style answer format with automatic correctness checks for objective scoring at scale. We evaluate leading LLM-based agents, revealing a substantial performance gap: even the best agent achieves only 14.55% accuracy on the hardest tasks. We detail our benchmark's design, dataset composition, task formulation, evaluation protocol, report baseline results and analyze failure modes. DABstep is released with a public leaderboard and toolkit to accelerate research in autonomous data analysis.

Data & Code: huggingface.co/spaces/adyen/DABstep

Data & Dataset Card: huggingface.co/datasets/adyen/dabstep

1 Introduction

Recent advances in large language models (LLMs) have enabled the development of autonomous agentic workflows, particularly showing promise for automating complex, multi-step tasks within domains like data science and software engineering. However, the evaluation of such agents, especially for data analysis, faces significant hurdles. Many existing benchmarks rely on synthetic tasks, overly simplistic evaluations, or subjective assessment methods (such as LLM-as-a-judge, known for biases), limiting their ability to accurately reflect the challenges encountered in real-world analytical scenarios and gauge true agent capabilities.

To address these limitations, we introduce the Data Agent Benchmark for Multi-step Reasoning (DABstep) which comprises over 450 authentic data analysis tasks derived directly from operational workloads at Adyen. Distinctly, these tasks combine structured (e.g., CSV, JSON) and unstructured data (e.g., text, domain-specific documentation or complicated manuals), requiring agents to demonstrate technical data manipulation skills (spanning SQL, statistical analysis, coding), a deep understanding of contextual instructions, and the ability to plan hierarchically.

A core design principle of DABstep is its focus on multi-step reasoning complexity. Unlike benchmarks where tasks might be solvable via single-shot generation [47, 24, 21, 20], *hard tasks in DABstep are designed to require multi-step reasoning*. Agents must decompose problems into sequential, iterative steps—such as filtering data, computing aggregates, consulting reference tables, and handling intermediate results—often requiring interaction with a code execution environment as shown

♣ Equal contribution.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

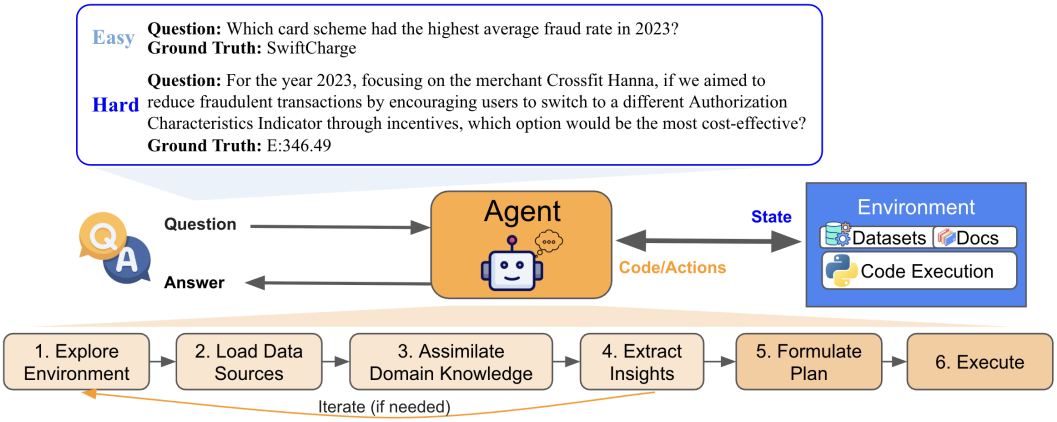

Figure 1: System overview of DABstep's agent-task interaction. The figure illustrates the core components: task input (questions), agent, execution environment, and output (answers). Two representative questions are shown: a Risk/Fraud question from the Easy set (top), requiring 2+ data sources and at least 3 execution steps; and a Scheme Fees question from the Hard set (bottom), requiring 3+ sources and more complex reasoning over at least 6 steps. Agents must combine contextual understanding, code execution, and iterative refinement to produce a correct answer. See full trace example in section A.4

in Figure 1. Furthermore, DABstep is designed for accessibility with a low barrier to entry, avoiding complex scaffolding or environments. An automated online leaderboard facilitates easy submission and *standardized* evaluation while also fostering community participation.

Our evaluations underscore DABstep's significant challenge to current state-of-the-art LLMs. Top-performing agents, such as o4-mini [34], at time of writing, achieve only 14.55% accuracy (Table 1), highlighting substantial gaps between current agent capabilities and the demands of rigorous, practical data analysis.

This paper makes the following key contributions:

- **Data**: A novel benchmark featuring **over 450 real-world data analysis tasks** designed to test complex, multi-step reasoning and planning while leveraging diverse data sources including a large (+100k) payments dataset among others.
- **Factoid Evaluation Framework**: An objective and standardized evaluation methodology centered on factoid answers with binary (right/wrong) outcomes, supported by a flexible scoring mechanism to handle formatting variations *fairly*.
- **Baselines**: Performance results and failure modes for leading open and closed LLM agents, identifying current limitations and key areas for future research.
- **Community Platform**: An accessible setup including a developer set, a quick-start notebook, an open-source baseline code, and a centralized live leaderboard to track progress and foster collaboration.

Through DABstep, we aim to drive progress in developing AI agents capable of rigorous, practical, multi-step data analysis, better aligned with real-world analytical needs.

2 Design Principles & Related Works

The guiding philosophy behind DABstep is grounded in four key principles that collectively emphasize realism, complexity, objectivity, and accessibility. In Table 2 we provide a high-level comparison to other related benchmarks on the main characteristics that we consider required to evaluate real-world data analysis tasks.

Table 1: Performance of baseline models on the DABstep benchmark (Hidden Test Set). Scores reflect accuracy (%) on Hard and Easy splits. Costs are estimates based on public pricing at the time of evaluation (Q1 2025) and token usage; open models run locally are considered free ('-'). All baselines run for a maximum of 10 steps per task with a ReAct style prompt except for the reasoning models. See Section 4 for methodology.

Name	Hard (%)	Easy (%)	Total Cost ($)
o4-mini [34]	14.55	76.39	93
Claude 3.7 Sonnet [4]	13.76	75.00	139
o3-mini [34]	13.76	72.22	85
Gemini 2.5 Pro [10]	12.70	66.67	270
GPT 4.1 [33]	12.43	80.56	155
o1 [32]	11.11	69.44	435
Deepseek R1 [12]	11.04	68.21	3
Claude 3.5 Sonnet [3]	9.26	77.78	97
Llama 4 Maverick [2]	8.73	75.00	-
GPT-4o [30]	6.08	66.67	53
Deepseek V3 [26]	5.56	66.67	2
Claude 3.5 Haiku [3]	5.03	77.78	35
Llama 3.3 70B [11]	3.70	68.06	-
GPT-4o-mini [31]	3.44	69.44	3
Llama 4 Scout [2]	1.85	52.78	-
Llama 3.2 1B [1]	0.00	1.39	-

2.1 Real-World and Multi-Step Complexity

A central design principle of DABstep is its emphasis on realistic analytical challenges that require multi-step reasoning over heterogeneous data sources. Unlike benchmarks that focus on abstract math problems [14], isolated code snippets [7], or synthetic QA tasks [8], DABstep's over 450 tasks are derived directly from operational workloads at Adyen, reflecting the complex, iterative problem-solving scenarios faced by professional data analysts.

These tasks are grounded in real-world financial analysis and integrate both structured data (e.g., CSV tables like `payments.csv`, JSON files like `fees.json`) and unstructured documentation (e.g., Markdown files like `manual.md`). Solving them demands technical proficiency in data manipulation (e.g., filtering, aggregation, joins), statistical reasoning, and the ability to extract and apply domain-specific rules from documentation. For example, agents must answer questions such as "Which card scheme had the highest average fraud rate in 2023?" or perform scenario analysis like "If merchant X changed its business category, how would that affect fees?"

Crucially, DABstep tasks are explicitly designed to resist one-shot code generation strategies [47, 24]. Especially in the Hard split (84% of tasks), no question can be answered through a single direct execution. Hard tasks require iterative data filtering and cross-referencing, which single-shot code cannot handle. Instead, agents must follow a multi-step reasoning process that involves identifying relevant context, synthesizing data across files, computing intermediate results, and validating outputs. Many tasks require cross-referencing structured sources such as tables (CSV or otherwise) or JSON datasets with unstructured content such as documentation or technical manuals, and executing multi-stage plans within a Python environment (see Figure 1). Previous code generation benchmarks [21, 7, 25] focus on isolated coding skills but lack the requirement for multi-step reasoning grounded on domain knowledge across structured and unstructured sources. Other benchmarks moved towards iterative analysis via interactive environments with multiple code executions inspired by Intercode [44], but lacked the focus on domain-specific knowledge integration during planning [23, 17], are limited to text-to-Pandas within Jupyter environments [46] or do not require integrating with heterogeneous data sources [15].

This emphasis on sequential, tool-augmented reasoning distinguishes DABstep from benchmarks focused solely on Text-to-SQL benchmarks [47, 24, 40, 13, 22, 50] or closed-domain QA [18], and better reflects the iterative nature of real-world data workflows. As evidenced in Section 4, even state-of-the-art LLM agents struggle with these challenges, especially when planning, tool use, or implicit instruction following is required.

Table 2: Comparison with existing related benchmarks. Columns include the benchmark topic (Topic), the number of tasks (# Tasks) and whether the tasks in the benchmark involve: integrating heterogeneous data sources (Hetero.), come from real-world scenarios driving business value and not from just educational, synthetic or online resources (Real World), require following domain-specific business knowledge and/or rules to arrive to a solution (Domain Knowledge), require multi-model input handling (Multi-modal), require multiple steps of reasoning to arrive to a solution (Multi-step), require agents to perform analysis via code (Code), can be objectively evaluated (Objective Evals). DABstep v1 focuses on structured/unstructured text only.

Benchmark	Topic	Hetero.	Real World	Domain Knowledge	Multi-modal	Multi-step	Code	Objective Evals	#Tasks
FinanceBench [18]	Finance QA	✓	✓	✓	✗	✗	✗	✗	10,231
GAIA [28]	General QA	✓	✓	✗	✓	✗	✓	✓	466
MATH [14]	Math QA	✗	✗	✗	✗	✗	✗	✓	12,500
GSM8K [29]	Math QA	✗	✗	✗	✗	✗	✗	✓	8,500
Spider [47]	Text-to-SQL	✗	✗	✗	✗	✗	✗	✓	1,181
Spider 2 [23]	Text-to-SQL	✓	✗	✓	✗	✓	✓	✓	632
BIRD [24]	Text-to-SQL	✗	✗	✓	✗	✗	✗	✓	12,751
KaggleDBQA [22]	Text-to-SQL	✓	✗	✗	✗	✗	✓	✓	272
WikiSQL [50]	Text-to-SQL	✗	✓	✗	✗	✗	✗	✓	80,654
HumanEval [7]	Text-to-Python	✗	✗	✗	✗	✗	✓	✗	164
NL2Bash [25]	Text-to-Bash	✗	✓	✗	✗	✗	✗	✗	9,305
Arcade [46]	Text-to-Pandas	✗	✗	✗	✗	✓	✓	✓	10,082
SWE-Bench [19]	Software	✗	✓	✗	✓	✗	✗	✓	2,294
WebArena [51]	Web	✓	✗	✗	✓	✓	✗	✓	812
OSWorld [42]	Computer Control	✓	✓	✗	✓	✓	✗	✓	369
Intercode [44]	Iterative code	✗	✗	✗	✗	✓	✓	✓	1351
MLAgentBench [16]	Machine Learning	✓	✓	✗	✓	✓	✓	✓	13
DABench [15]	Data Analysis	✗	✓	✗	✗	✓	✓	✓	257
DA-Code [17]	Data Science	✓	✓	✗	✗	✓	✓	✓	500
DS-1000 [21]	Data Science	✗	✓	✗	✗	✗	✓	✓	1,000
Spider2-V [5]	Data Science	✓	✓	✓	✗	✗	✓	✓	494
DSEval [48]	Data Science	✗	✗	✗	✗	✓	✓	✓	825
DSBench [20]	Data Science	✓	✗	✗	✓	✓	✓	✗	540
DABstep (ours)	Data Analysis	✓	✓	✓	✗	✓	✓	✓	450

2.2 Objective Evaluation

DABstep was designed with a guiding principle of verifiable answers to ensure robust and straightforward evaluations. Unlike many benchmarks that include tasks validated by LLM-as-a-judge approaches [20, 43, 28], we selectively curated questions to produce objective, factoid answers—such as numbers, lists, or concise strings—amenable to automated scoring. This choice reflects a deliberate trade-off: while many impactful use-cases often involve free-form outputs (e.g., analytical reports, narrative summaries), their evaluation often requires subjective, resource-intensive methods like LLM-as-a-judge, introducing potential bias and infrastructure costs [49]. By focusing on verifiable answers, DABstep sacrifices some task diversity (like more open-ended tasks with free-form output) but gains in advantages: high evaluation reliability, scalability without LLM dependency, and a streamlined experience for developers and users. As a concrete example, factoid task answers may be numerical (e.g., 42) or a list of items (e.g., 'CardA, CardB'). This curation aligns with our goal of creating a benchmark that prioritizes reliability and accessibility, enabling fair comparisons of LLM agents on data analysis tasks without the overhead of complex evaluation pipelines. However, even factoid answers exhibit variability (e.g., '42' vs. 'forty-two' vs. '42.00'), necessitating a flexible and deterministic scoring mechanism (detailed in Section A.2) to maintain fairness without reverting to rigid exact matching or LLM-based subjectivity.

2.3 Simple accessible setup

A key design philosophy underpinning DABstep is ensuring a low barrier to entry for researchers and practitioners. We deliberately avoided requiring complex environments or specialized infrastructure often associated with other benchmarks, like general agent platforms [27, 5, 9, 42, 41], software and

ML engineering development benchmarks [19, 6, 16] and dedicated SQL benchmarks [47]. Instead, DABstep is designed for ease of use, requiring only a standard Python runtime for task execution.

The evaluation process is streamlined through an automated online leaderboard. Participants can submit their results easily, receiving *standardized* evaluations without needing to manage complex local evaluation setups and avoids biased cherry-picked baselining [35]. To further support accessibility, baseline implementations and prompts are provided openly. This simplicity in setup and evaluation is intended to encourage broad participation from the research community, mirroring the accessibility that has spurred progress in widely adopted NLP benchmarks [39]. By minimizing setup complexity, DABstep allows researchers to focus directly on the core challenges of multi-step reasoning and data analysis for LLM agents.

3 Benchmark

3.1 Task Curation

DABstep consists of over 450 tasks derived from real-world analytical challenges at Adyen, curated to evaluate agent capabilities through factoid-style question answering with objective scoring. Each task mirrors typical workflows and queries faced by professional data analysts. The tasks were selected from real but anonymized internal queries to ensure a diverse range of data manipulation, reasoning steps, and contextual understanding. Each task (see example in Section A.4) is presented with a natural language prompt, which includes:

- **Question**: A specific question (e.g., "Which card scheme had the highest average fraud rate in 2023?") or an analytical scenario (e.g., assessing the fee impact of a merchant changing business categories).
- **Guidance**: Clear guidance on the required answer format (e.g., "Provide the name of the scheme" or "Provide the result as `scheme:fee` where fee is rounded to 2 decimal places"). This minimizes penalties due to trivial formatting errors.
- **Context**: A set of context files (datasets and documentation) necessary to solve the task.
- **Level**: A difficulty tag (Easy or Hard).

The tasks require agents to integrate information from heterogeneous data sources, demanding both technical data analysis skills (using code) and domain-specific understanding learned from the provided context.

To provide the necessary context or domain knowledge for agents to reason over while solving tasks, we release several datasets, including a large payments dataset with over 100,000 anonymized transactions and various industry-specific manuals and documentation represented in simplified formats. For instance, in the finance industry, business context is often outlined in extensive handbooks from payment networks, regulators, and processors. For this benchmark version, we have created markdown documentation distilling essential business knowledge (e.g., concepts like Merchant Category Codes (MCC), Authorization Characteristics Indicators (ACI), scheme fee structures) into a precise yet accessible format crucial for solving many tasks accurately.

Symbolic Reasoning through Parameterization In the spirit of GSM-Symbolic [29], many base task types have been expanded into multiple concrete instances by systematically varying parameters like time ranges, merchant names, or specific thresholds. This parameterization significantly increases the number of unique task instances derived from a smaller set of core analytical workflows. *Concretely, out of the 450 total questions, 95 of those are core questions.* The rationale is twofold: first, to minimize the possibility of agents succeeding through 'lucky guesses' or memorization of answers potentially seen during pre-training; second, to rigorously validate the core reasoning capabilities of the agents – their ability to apply the same logical steps consistently across different input values, thereby testing generalization. This approach emphasizes evaluating the underlying problem-solving process rather than just retrieving specific facts. A concrete example: A task asking for fraud rates in September is varied by changing the month to July or February, or by altering merchant names.

3.2 Task Distribution

DABstep comprises over 450 data analysis tasks, permuted from a core of 95 (see 3.1), combining structured datasets (like CSV tables and JSON files) with unstructured text (Markdown documentation). The tasks are categorized by difficulty:

- **Easy Tasks:** 72 tasks (approximately 16% of the total). These generally involve querying or processing a single primary dataset with minimal reliance on complex contextual information from documentation. They serve as basic sanity checks or warm-ups.
- **Hard Tasks:** 378 tasks (approximately 84% of the total and permuted from 23 core questions). These tasks form the core challenge of the benchmark. They typically require cross-referencing multiple data sources, understanding domain-specific concepts explained in manuals/documentation and executing a multi-step reasoning process involving several stages of data manipulation and interpretation. Section A.4 shows that solving these tasks goes significantly beyond simple single-shot code generation capabilities.

This distribution, heavily weighted towards Hard tasks, intentionally reflects the complexity of real-world data analysis challenges faced by professional data analysts, demanding robust technical skills combined with planning and reasoning capabilities. From our baselines in Section 4, there is a 49% correlation with performance on the easy set to performance on the hard set. In Section A.3 we provide a snapshot of the key data sources included in the benchmark's context, illustrating the mix of structured formats and unstructured documentation.

3.3 Evaluation Protocol

DABstep is designed for automated, fast, and factual evaluation. Echoing the principles of benchmarks like GAIA [28], each task requires a specific factoid answer: typically a string (one or a few words), a number, or a list of strings/numbers (comma-separated unless otherwise specified in the guidance). There is only one correct ground truth answer for each task.

Evaluation is performed via automated comparison between the agent's final answer and the corresponding ground truth answer, using a flexible scoring algorithm detailed in Section A.2 (and Algorithm 1 in the Appendix). This quasi-exact match approach, with type-specific normalization and tolerance, ensures objective and scalable scoring.

Hidden Test Set for Zero-Shot Generalization To robustly evaluate the zero-shot generalization capabilities of agents, DABstep employs a single, held-out hidden test set. This set contains the majority of the benchmark tasks and is used exclusively for the official evaluation conducted via our public leaderboard. We do not release separate public validation or test sets derived from this hidden data.

This design choice serves several purposes. First, it encourages the development of agents that generalize well across the diverse range of data analysis tasks represented in the benchmark, rather than overfitting to a specific public subset. Second, by hiding the test ground truths, we maintain the long-term integrity of the benchmark and reduce the risk of leakage into the training corpora of future models, i.e *saturation*. This safeguard ensures that DABstep remains a reliable measure of true generalization capabilities over time. Finally, our design aligns with best practices for rigorous benchmarking [35], supporting fair and standardized evaluation across all participants.

To facilitate development, local testing, and environment setup without requiring interaction with the official leaderboard for every iteration, we release a smaller public developer set. This set includes a representative sample of tasks with their ground truth answers, allowing researchers to verify their agent implementations and scoring logic end-to-end before submitting to the leaderboard for evaluation on the hidden test set.

4 Baselines

To establish initial performance levels on DABstep and highlight the gap between current LLM agent capabilities and the demands of complex, real-world data analysis tasks, we evaluated a range

of state-of-the-art open and closed-source language models available at the time of evaluation (Q1 2025).

4.1 Baseline Setup

To ensure fair and scalable evaluation across a diverse range of models, we adopted a standardized setup with minimal model-specific tuning. Most models were prompted using a generic ReAct-style approach [45], framing the LLM as a data analyst that reasons step by step, invokes a Python execution tool when needed, and formats its final answer in a structured output. The prompt included abstract demonstrations of the desired reasoning-action-observation loop. We intentionally avoided heavily engineered or model-specific prompts to keep comparisons consistent across different architectures.

For a few models—such as o4-mini, o3-mini, o1, R1 and Gemini 2.5 Pro—we employed a "Reasoning Prompt," a slightly adapted variant better aligned with their internal reasoning paradigms. However, these were still standardized across those models to ensure internal consistency.

Open-source models were run on a dedicated cluster with 4× Nvidia A100 GPUs (80GB each). All tasks were executed in isolated environments where the agent had access to a Python kernel, enabling dynamic code execution for data loading, manipulation, and statistical analysis via standard libraries. Each task environment also included mounted context files containing relevant datasets and documentation.

Notably, we avoided the use of complex agent frameworks or external orchestration layers. The entire setup consisted of a lightweight wrapper [36] around the LLM API, providing Python execution and I/O, but leaving all decision-making—including reasoning, code generation, and result interpretation—to the model itself. This minimal infrastructure ensures that observed performance reflects the model's inherent capabilities, rather than the sophistication of external tooling.

The full baseline implementation, including prompts and evaluation code, is available in the benchmark repository.[*]

4.2 Results

Table 1 presents accuracy results, broken down by the Easy (16% of tasks) and Hard (84% of tasks) splits of the benchmark. These scores demonstrate the significant challenge posed by DABstep, particularly in the Hard split. Even the top-performing model, o4-mini (using the reasoning prompt) achieves only 14.55% accuracy on the Hard tasks. Several other leading models—including proprietary models (e.g., Claude 3.7 Sonnet and o3-mini), as well as strong open models— scored below 14% on the Hard set.

In contrast, performance on the Easy split is considerably higher. For instance, o4-mini reaches 76.39% accuracy, suggesting that many LLMs can already handle one-shot analysis tasks with high effectiveness. The stark drop in scores on the Hard split reveals that tasks requiring multiple steps of reasoning remain largely unsolved. Notably, smaller or less specialized models, such as Llama 3.2 1B, struggle across the board, particularly on the Hard tasks, further highlighting the gap in current capabilities.

The evaluation also considered the approximate cost of running the benchmark for proprietary API-based models (Table 1), revealing significant trade-offs between performance and cost. Costs per full benchmark run varied widely, highlighting the economic factors involved in deploying these agents.

4.3 Failure Modes

Agent performance often degrades when deviating from the ideal iterative trajectory outlined in Figure 1, particularly during planning and execution phases. Analysis of incorrect agent trajectories revealed common failure patterns frequently linked to DABstep's design principles emphasizing multi-step complexity and real-world nuances. Section A.4 illustrates a real agent trace example with some of the failure modes we introduce below.

[*] https://huggingface.co/spaces/adyen/DABstep/tree/main/baseline

Planning and Instruction Following Deficiencies. Agents often struggled to correctly decompose complex Hard tasks into a viable sequence of sub-steps. They might miss necessary intermediate calculations, fail to consult required documentation at the right point, or hallucinate incorrect analysis plans. A common issue was attempting calculations before reading the relevant sections of documentation that defined key domain terms or logic. In addition, we observed that agents tend to perform well at following instructions which are **explicitly** stated in the context (i.e domain-specific formula). However, agents are considerably more prone to fail when they face a task in which they need to follow rules **implicitly** mentioned in the context (i.e domain-specific rule with multiple downstream implications) or composite rules which are linked together implicitly. While our observations are preliminary, one plausible explanation for these failure modes lies in the nature of the current retrieval systems that can be scaffolded around models and the inductive biases of LLMs. Specifically, the self-attention mechanism [38], central to the LLM architecture and retrieval systems, primarily captures semantic similarity (i.e., relationships based on token co-occurrence and contextual proximity) rather than abstract conceptual similarity (i.e., relationships between underlying ideas irrespective of their surface-level expression). This might result in agents missing to link together pieces of information, not explicitly mentioned, which are crucial for the task at hand. In contrast, human analysts excel at identifying and linking such implicit cues, highlighting a potential area where current models fall short.

Inefficient Code. We observe that the code generated by the agent becomes increasingly inefficient as the complexity of the reasoning required by a task also increases. In particular, agents tend to default to low-level constructs such as explicit for-loops for tasks like computing group averages, even when high-level abstractions or idioms such as group-by or filter are available. This suggests that while the models can produce functionally correct code, their reasoning process fails to generalize to more abstract or idiomatic programming patterns as task difficulty grows.

Multi-step Instruction Following. In addition to the main, user-provided agent prompt, answers to tasks must also follow a formatting guidance prompt (Section 3.1). Potentially resulting in a complex, but realistic set of instructions to follow. Studies have shown [37] that enforcing multiple instructions in a single prompt, such as, in this case, formatting guidelines, significantly reduces LLM reasoning abilities. We did observe a non-trivial amount of errors resulted from failing to follow the specified output format guidance, for example, agents will often provide conversational text instead of outputting a number or apply incorrect rounding, wrong list delimiters or order, suggesting decomposing instructions into independent units as in [37].

Prompt Sensitivity. Certain models known for strong reasoning capabilities (R1, o1, etc) performed poorly with the standardized ReAct prompt (some scoring near 0% initially before trying a reasoning-specific standard prompt), indicating a high sensitivity to prompt structure and a potential reliance on specific custom prompting techniques not used in our fair, standardized baseline evaluation.

While these baseline results represent performance with non-optimized, standardized prompts and **should be considered a lower bound**, they effectively demonstrate the significant challenges DABstep presents in areas crucial for practical data analysis: robust multi-step reasoning, accurate handling of diverse data sources and domain knowledge, reliable tool use, and precise instruction following. These findings establish a clear benchmark and highlight key areas for future research aimed at improving agent capabilities.

5 Conclusion, Limitations & Future Work

DABstep is a large-scale benchmark designed to rigorously evaluate autonomous agents on realistic, multi-step data analysis tasks. With over 450 grounded challenges derived from financial workloads, DABstep uniquely combines code execution, contextual reasoning over structured and unstructured data, and an objective factoid-based evaluation protocol. Our baseline results show a stark capability gap: state-of-the-art LLM agents achieve only 14.55% accuracy on Hard tasks, underscoring critical limitations in reasoning, planning, tool use, and instruction following. By releasing data, evaluation tools, baseline code, and a public leaderboard, we aim to foster community engagement and accelerate research on practical, agentic data analysis.

However, this is the first iteration of the DABstep benchmark. It currently supports only text-based inputs and factoid-style outputs, limiting its ability to assess critical skills like visual reasoning (e.g., from charts or PDFs) or open-ended analysis (e.g., narratives, recommendations). These constraints reflect deliberate trade-offs to favor objective, scalable evaluation, but future iterations will address them by (i) expanding financial tasks (e.g., approval rate and temporal analysis), (ii) incorporating other domains (e.g., healthcare, e-commerce), (iii) increasing data scale and heterogeneity (e.g., longer documents, PDF manuals), (iv) introducing multimodal tasks, and (v) pushing toward agents capable of interactive clarification, exploratory analysis, and synthesis. These directions aim to close the gap between benchmark and real-world analytical workflows, while continuing to emphasize transparency, rigor, and accessibility.

References

[1] Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, 2024.

[2] Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2025.

[3] Anthropic. Claude 3.5: Next-generation language models. https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

[4] Anthropic. Claude 3.7: Advancements in language understanding. https://www.anthropic.com/news/claude-3-7-sonnet, 2025.

[5] Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Wenjing Hu, Yuchen Mao, et al. Spider2-v: How far are multimodal agents from automating data science and engineering workflows? In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, volume 37, pages 107703–107744, 2024.

[6] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[9] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: how capable are web agents at solving common knowledge work tasks? ICML'24, 2024.

[10] Google. Gemini 2.5: Our most intelligent ai model, 2025. URL https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/. Google Blog.

[11] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[13] Moshe Hazoom, Vibhor Malik, and Ben Bogin. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87, 2021.

[14] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[15] 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. InfiAgent-DABench: Evaluating agents on data analysis tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 19544–19572, 2024.

[16] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

[17] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, and Kang Liu. 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*, pages 13487–13521, 2024.

[18] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. Financebench: A new benchmark for financial question answering, 2023.

[19] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[20] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents from becoming data science experts? In *The Thirteenth International Conference on Learning Representations*, 2025.

[21] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.

[22] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, August 2021.

[23] 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. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*, 2025.

[24] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big benchmark for large-scale database-grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*, volume 36, pages 42330–42357, 2023.

[25] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, May 2018.

[26] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[27] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*, 2024.

[28] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*, 2023.

[29] Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[30] OpenAI. Introducing gpt-4o: Multimodal capabilities and efficiency. https://openai.com/index/hello-gpt-4o, 2024.

[31] OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

[32] OpenAI. Openai o1 and new tools for developers. https://openai.com/index/o1-and-new-tools-for-developers, 2024.

[33] OpenAI. Gpt-4.1 technical overview. https://openai.com/index/gpt-4-1, 2025.

[34] OpenAI. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini, 2025.

[35] Steffen Rendle, Li Zhang, and Yehuda Koren. On the difficulty of evaluating baselines: A study on recommender systems. *arXiv preprint arXiv:1905.01395*, 2019.

[36] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. smolagents: A smol library to build great agentic systems, 2025. URL https://github.com/huggingface/smolagents. GitHub repository.

[37] Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on large language model performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1218–1236, November 2024.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[39] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

[40] Ping Wang, Tian Shi, and Chandan K Reddy. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361, 2020.

[41] Michael Wornow, Avanika Narayan, Ben Viggiano, Ishan Khare, Tathagat Verma, Tibor Thompson, Miguel Hernandez, Sudharsan Sundar, Chloe Trujillo, Krrish Chawla, et al. Wonderbread: A benchmark for evaluating multimodal foundation models on business process management tasks. In *Advances in Neural Information Processing Systems*, volume 37, pages 115963–116021, 2024.

[42] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094, 2024.

[43] Tinghao Xie, Xiangyu Qi, Yi Zeng, Yangsibo Huang, Udari Madhushani Sehwag, Kaixuan Huang, Luxi He, Boyi Wei, Dacheng Li, Ying Sheng, et al. Sorry-bench: Systematically evaluating large language model safety refusal behaviors. *arXiv preprint arXiv:2406.14598*, 2024.

[44] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. Intercode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 23826–23854, 2023.

[45] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[46] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. 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)*, pages 126–173, 2023.

[47] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 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*, oct 2018.

[48] Yuge Zhang, Qiyang Jiang, XingyuHan XingyuHan, Nan Chen, Yuqing Yang, and Kan Ren. Benchmarking data science agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5677–5700, 2024.

[49] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623, 2023.

[50] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[51] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2024.

A Technical Appendices and Supplementary Material

A.1 Usage and Accessibility

DABstep is designed for broad accessibility. The benchmark platform, including baseline code, documentation, and the live leaderboard, is publicly hosted on **Hugging Face Spaces**. The core dataset, anonymized and released under an open license [Creative Commons Attribution 4.0 International], is available on the **Hugging Face Hub** with a detailed dataset card. The platform provides a standardized environment for evaluation via leaderboard submissions. [†]

[†] https://creativecommons.org/licenses/by/4.0/

A.2 Hybrid Scoring Algorithm

To ensure objective and robust evaluation while accommodating minor, semantically irrelevant variations in agent outputs, DABstep employs a hybrid scoring algorithm. This approach avoids the brittleness of pure exact string matching, which would unfairly penalize correct answers with trivial differences (e.g., "$42.00" vs. "42"), and sidesteps the potential biases, costs, and complexities associated with LLM-as-a-judge methods [49].

The algorithm first normalizes both the agent's predicted answer and the ground truth answer (e.g., converting to lowercase, trimming whitespace). It then applies type-specific comparison logic:

- **Numeric Comparison**: If both inputs can be interpreted as numbers, it extracts the numeric values, ignoring formatting like currency symbols or thousands separators (e.g., '$1,234.56' becomes 1234.56). Comparison allows for a small tolerance (e.g., 10^{-4}) to handle potential floating-point rounding differences.
- **List Comparison**: If inputs appear to be lists (based on delimiters like commas or semicolons), they are split into elements. Each element is normalized (e.g., whitespace trimmed, converted to lowercase). The lists are then compared, typically ignoring the order of elements unless the task specifically requires ordered output. Normalization handles variations like 'uber, spotify, nike' matching 'Nike, Uber, Spotify'. Recursive calls to the scoring function handle comparisons of elements within the lists if they are complex (e.g., lists of numbers).
- **String Comparison**: For general string answers, basic cleaning is applied (e.g., removing punctuation, extra whitespace). If the cleaned strings match exactly, the answer is correct. If not, fuzzy string matching (e.g., using Levenshtein distance or a similar metric) is employed, accepting answers with a high similarity score (e.g., > 0.95) to the ground truth which handles minor typos or variations. Special handling might apply for single-word vs. multi-word comparisons.

This tiered approach ensures that the evaluation focuses on the semantic correctness of the factoid answer, providing a fair yet rigorous assessment aligned with DABstep's goal of objective, automated scoring. Pseudocode for the algorithm is provided in Algorithm 1 in the Appendix (Figure A.2).

To validate our automated scoring algorithm, we collected 75 model-generated answers across Easy and Hard tasks from a diverse set of LLMs (GPT-4, Claude 3.5, LLaMA 3). Each answer was manually judged by two independent annotators, who labeled the response as correct or incorrect according to the task instructions and gold reference. Inter-annotator agreement was 97.3% (Cohen's k = 0.94), with disagreements resolved via discussion.

Our scoring function matched human judgment on all 75 examples, achieving 100% accuracy. Using binomial confidence intervals, this yields a 95% CI of [96.2%, 100%], indicating strong reliability. Crucially, many examples required tolerance to numeric rounding, flexible list ordering, or fuzzy string similarity—highlighting the importance of a robust scoring mechanism.

A.3 Dataset Snapshot

In Table 3 we provide a concise overview of each file in the DABstep benchmark's dataset bundle. This snapshot highlights file names, formats, and a brief description of their contents, which together capture the key aspects of our finance use cases.

Algorithm 1 Hybrid Answer Scoring Algorithm (Pseudocode)

procedure SCOREANSWER(predicted_answer, ground_truth)
    $pred \leftarrow$ Normalize(`predicted_answer`) ▷ e.g., lowercase, trim whitespace
    $gt \leftarrow$ Normalize(`ground_truth`)
    if IsNumeric($pred$) **and** IsNumeric(gt) **then** ▷ Check if both are numeric
        $n_{pred} \leftarrow$ ExtractNumeric($pred$) ▷ Handles , etc.
        $n_{gt} \leftarrow$ ExtractNumeric(gt)
        return CompareNumeric(n_{pred}, n_{gt}, tolerance=10^{-2}) ▷ Allows small float diff
    end if
    if IsList($pred$) **and** IsList(gt) **then** ▷ Check if both look like lists
        $l_{pred} \leftarrow$ SplitSortNormalizeList($pred$) ▷ Split, normalize elements, sort
        $l_{gt} \leftarrow$ SplitSortNormalizeList(gt)
        if length(l_{pred}) $\neq$ length(l_{gt}) **then**
            return `false`
        end if
        all_match $\leftarrow$ `true`
        for i from 0 to length(l_{pred}) - 1 **do**
            if not ScoreAnswer($l_{pred}[i]$, $l_{gt}[i]$) **then** ▷ Recursive call for elements
                all_match $\leftarrow$ `false`
                break
            end if
        end for
        return all_match
    end if
    ▷ Default to string comparison
    $pred_{clean} \leftarrow$ CleanString($pred$) ▷ Remove punctuation, extra spaces
    $gt_{clean} \leftarrow$ CleanString(gt)
    if $pred_{clean} = gt_{clean}$ **then**
        return `true` ▷ Exact match after cleaning
    end if
    similarity_score $\leftarrow$ CalculateStringSimilarity($pred_{clean}$, gt_{clean}) ▷ e.g., Levenshtein ratio
    if similarity_score > 0.95 **then** ▷ Threshold for fuzzy match
        return `true`
    end if
    return `false` ▷ Default to false if no match
end procedure

▷ *Helper functions like Normalize, IsNumeric, ExtractNumeric, CompareNumeric, IsList, SplitSortNormalizeList, CleanString, CalculateStringSimilarity are assumed.*

Table 3: Snapshot of key datasets provided as context within the DABstep benchmark, covering aspects of the financial payments sector.

Name	Description
`payments.csv`	Anonymized payments dataset containing over 138,000 transactions with features relevant to fraud detection and risk analysis use-cases.
`payments-readme.md`	Human-readable documentation explaining the columns and content of the `payments.csv` dataset.
`acquirer_countries.csv`	Table mapping acquiring bank identifiers to their respective countries.
`fees.json`	Dataset detailing various scheme fee structures (over 1000 entries), often dependent on transaction and merchant attributes.
`merchant_category_codes.csv`	Table listing Merchant Category Codes (MCCs) and their descriptions.
`merchant_data.json`	Table containing descriptive information about various merchants (anonymized).
`manual.md`	A comprehensive guide (distilled for the benchmark) explaining core payment processing concepts (e.g., Account Types, MCC, ACI), detailing fee calculation logic based on merchant and transaction attributes, and outlining best practices for minimizing costs and fraud risk. Essential for solving many Hard tasks.

A.4 Failure Mode Example

Figures 2–11 illustrate a trace from a real task execution by a Claude 3.7 Sonnet agent, one of the best performing baselines, on the DABstep benchmark. For clarity, we omit action outputs that are excessively verbose (e.g., full documentation dumps, programming error traces).

The task requires identifying all applicable fees for a specific merchant on a given date, with the expected output being a comma-separated list as defined in the task guidelines. To address this, the agent first explores the data sources available in the `data/context` environment. In steps 0 and 1 (Figures 2 and 3), the agent succeeds in identifying and retrieving the relevant context files.

In steps 2 and 3, the agent parses these data to extract merchant business attributes (Figures 4 and 5). Next, it returns to the context in step 4 (Figure 6) to refine its understanding of fee scheme rules. After refining its understanding, at step 5 (Figure 7), the agent recognizes a missing piece of the puzzle: merchant payment traffic characteristics, which it attempts to find next.

Step 6 (Figures 8–10) represents the synthesis stage, where the agent combines insights from merchant's business and payment traffic with domain-specific rules to discriminate the applicable fees. Finally, in step 7 (Figure 11), it composes an answer conforming to the task output format.

Despite the apparently coherent strategy, the agent ultimately fails to produce the correct result. This is because the agent misses a critical piece of information from the context (in step 6, Figure 9), although it has ingested into its memory in step 1 (Figure 3). The overlooked information is that merchant monthly fraud rates and other monthly payment traffic statistics discriminate between fees, and these statistics need to be computed to then be filtered on. Therefore the provided list of fees in the answer is a superset of the actual applicable fees. This behavior is an example of *Planning and Instruction Following Deficiencies* described in Section 4.3. Step 6 (Figure 9) also shows some inefficient code where the agent iterates transactions and fees using nested for-loops instead of using libraries built-in operations like group-by and filters.

LLM: Claude 3.7 Sonnet - Input Tokens: 81k - Output Tokens: 8k

Question: For the 10th day of the year 2023, what are the Fee IDs applicable to Belles_cookbook_store?

Guidelines: Answer must be a list of values in comma separated list, eg: A, B, C. If the answer is an empty list, reply with an empty string. If a question does not have a relevant or applicable answer for the task, please respond with 'Not Applicable'
Before answering the question, reference any documentation in the data dir and leverage its information in your reasoning / planning.

Step 0

Agent

I'll solve this task by exploring the data in the `data/context` directory, understanding the structure, and then finding the Fee IDs applicable to Belles_cookbook_store for the 10th of the year 2023.

Thought: First, I need to explore the `data/context` directory to see what files are available and understand the data structure and documentation.

Code:
```py
import os

# List files in the data/context directory
files = os.listdir('data/context')
print("Files in data/context directory:")
for file in files:
    print(file)
```

Execution Logs

```
Files in data/context directory:
fees.json
merchant_data.json
merchant_category_codes.csv
payments.csv
payments-readme.md
acquirer_countries.csv
manual.md
```

Figure 2: Agent step 0 where environment exploration happens.

Step 1

Agent

Thought: Great, I can see several files that might be useful for answering this question. I should first check if there's a manual or documentation file that could help me understand the data structure. I see a `manual.md` file which could be useful.

Code:

```py
# Read the manual.md file to understand the data structure
with open('data/context/manual.md', 'r') as file:
    manual_content = file.read()
    print(manual_content)
```

Execution Logs

```
# Merchant Guide to Optimizing Payment Processing and Minimizing Fees

Version 2.1 | Last Updated: November 1, 2024

## Table of Contents
1. Introduction
2. Account Type
3. Merchant Category Code
...
[Excluded content]
```

Figure 3: Agent step 1 where the agent loads domain-specific documentation into its working memory. Full documentation dump is omitted.

Step 2

Agent

Thought: The manual provides a lot of useful information. I understand that there are different files that might contain the information I need. Let's first check the `merchant_data.json` file to find information about Belles_cookbook_store.

Code:

```py
import json

# Read the merchant_data.json file
with open('data/context/merchant_data.json', 'r') as file:
    merchant_data = json.load(file)

# Check if Belles_cookbook_store is in the data
if 'Belles_cookbook_store' in merchant_data:
    print("Found Belles_cookbook_store in merchant_data.json")
    print(merchant_data['Belles_cookbook_store'])
else:
    print("Belles_cookbook_store not found in merchant_data.json")

# Let's list the keys in the merchant_data to see if the name might be formatted differently
print("\nMerchant names in the data:")
for merchant_name in merchant_data.keys():
    print(merchant_name)
```

Execution Logs

Belles_cookbook_store not found in merchant_data.json

Merchant names in the data:
ERROR: Code execution failed at line 'for merchant_name in merchant_data.keys():
    print(merchant_name)' because of the following error:
Object [{'merchant': 'Crossfit_Hanna', 'capture_delay': 'manual', 'acquirer': ['gringotts', 'the_savings_and_loan_bank', 'bank_of_springfield', 'dagoberts_vault'], 'merchant_category_code': 7997,
'account_type': 'F'}, {'merchant': 'Martinis_Fine_ ...] has no attribute keys

Figure 4: Agent step 2 where the agent tries to find more about the merchant but it fails to produce valid code. The full error trace is omitted.

Step 3

Agent

Thought: I see the issue. The merchant_data.json file contains a list of dictionaries, not a dictionary with keys. Let's try a different approach to find information about Belles_cookbook_store.

Code:

```py
import json

# Read the merchant_data.json file
with open('data/context/merchant_data.json', 'r') as file:
    merchant_data = json.load(file)

# Find data for Belles_cookbook_store
belles_data = None
for merchant in merchant_data:
    if merchant.get('merchant') == 'Belles_cookbook_store':
        belles_data = merchant
        print("Found Belles_cookbook_store data:")
        print(belles_data)
        break

if belles_data is None:
    print("Belles_cookbook_store not found in merchant_data.json")
```

Execution Logs

Found Belles_cookbook_store data:
{'merchant': 'Belles_cookbook_store', 'capture_delay': '1', 'acquirer': ['lehman_brothers'], 'merchant_category_code': 5942, 'account_type'

Figure 5: Agent step 3 where the agent successfully finds merchant business related data.

Step 4

Agent

Thought: Great! Now I have information about Belles_cookbook_store. According to the data, it's a merchant with category code 5942, account type 'R' (Enterprise - Retail), and a capture delay of '1'. Next, I need to look at the fees.json file to understand the fee structure and then check what fees are applicable on the 10th of the year 2023.

Code:

```py
import json

# Read the fees.json file
with open('data/context/fees.json', 'r') as file:
    fees_data = json.load(file)

# Check the structure of the fees data
print("Fees data structure:")
print(type(fees_data))
if isinstance(fees_data, list) and len(fees_data) > 0:
    print("Sample fee entry:")
    print(fees_data[0])
else:
    print("No fee data found or unexpected format")
```

Execution Logs

Fees data structure:
<class 'list'>
Sample fee entry:
{'ID': 1, 'card_scheme': 'TransactPlus', 'account_type': [], 'capture_delay': None, 'monthly_fraud_level': None, 'monthly_volume': None, 'merchant_category_code': [8000, 8011, 8021, 8031, 8041, 7299, 9399, 8742], 'is_credit': False, 'aci': ['C', 'B'], 'fixed_amount': 0.1, 'rate': 19, 'intracountry': None}

Figure 6: Agent step 4 where the agent refines its understanding about fee calculations.

Step 5

Agent

Thought: Now I understand the structure of the fees.json file. I need to find the fees that are applicable to Belles_cookbook_store on the 10th of the year 2023. Let me first check if there's a payments.csv file that might contain payment information for the date in question.

Code:

```py
import pandas as pd
# Read payments.csv
try:
    payments_df = pd.read_csv('data/context/payments.csv')
    print("Payments.csv columns:")
    print(payments_df.columns.tolist())

    # Let's check if there are any records for Belles_cookbook_store on the 10th of the year 2023
    if 'merchant' in payments_df.columns and 'date' in payments_df.columns:
        belles_payments = payments_df[(payments_df['merchant'] == 'Belles_cookbook_store') &
                                      (payments_df['date'].str.contains('2023') &
                                       payments_df['date'].str.contains('10'))]
        if not belles_payments.empty:
            print("\nPayments for Belles_cookbook_store on the 10th of 2023:")
            print(belles_payments.head())
        else:
            print("\nNo payments found for Belles_cookbook_store on the 10th of 2023")
    else:
        print("\nRequired columns 'merchant' or 'date' not found in payments.csv")
except Exception as e:
    print(f"Error reading payments.csv: {e}")

# Let's also check the payments-readme.md
try:
    with open('data/context/payments-readme.md', 'r') as file:
        readme_content = file.read()
        print("\nPayments-readme.md content:")
        print(readme_content)
except Exception as e:
    print(f"Error reading payments-readme.md: {e}")
```

Execution Logs

```
Payments.csv columns:
['psp_reference', 'merchant',  .. [Excluded]]
Required columns 'merchant' or 'date' not found in payments.csv

Payments-readme.md content:
This is documentation for the payments.csv dataset [Excluded] …
```

Figure 7: Agent step 5 where the agent is looking for relevant information in the merchant's payment traffic.

Step 6

Agent

Thought: Now I understand that the payments.csv file contains information about payments but with a different column structure than I initially expected. Instead of a 'date' column, it has 'year' and 'day_of_year' columns, which is actually helpful for our question. The 10th day of the year 2023 would be corresponding to 'year' = 2023 and 'day_of_year' = 10.

Let's filter the payments data to find transactions for Belles_cookbook_store on the 10th day of 2023, and then identify which Fee IDs would apply to those transactions based on the fees.json file and the merchant characteristics.

Code:

```py
import pandas as pd
import json

# Read the necessary files
with open('data/context/merchant_data.json', 'r') as file:
    merchant_data = json.load(file)

with open('data/context/fees.json', 'r') as file:
    fees_data = json.load(file)

payments_df = pd.read_csv('data/context/payments.csv')

# Find Belles_cookbook_store data
belles_data = None
for merchant in merchant_data:
    if merchant.get('merchant') == 'Belles_cookbook_store':
        belles_data = merchant
        break

if belles_data:
    print("Belles_cookbook_store characteristics:")
    print(f"Account Type: {belles_data['account_type']}")
    print(f"Merchant Category Code: {belles_data['merchant_category_code']}")
    print(f"Capture Delay: {belles_data['capture_delay']}")

    # Filter payments for Belles_cookbook_store on day 10 of 2023
    belles_payments_day10 = payments_df[
        (payments_df['merchant'] == 'Belles_cookbook_store') &
        (payments_df['year'] == 2023) &
        (payments_df['day_of_year'] == 10)
    ]
```

Continued

Figure 8: Agent step 6, part 0, where agent is retrieving payment traffic from the merchant. The full code generated by the agent in step 6 is continued in 9.

Step 6

Agent

```
    if not belles_payments_day10.empty:
        print(f"\nFound {len(belles_payments_day10)} payments for Belles_cookbook_store on day 10
of 2023")

        # Get unique combinations of relevant fields for checking fee applicability
        unique_txn_types = belles_payments_day10[['card_scheme', 'is_credit', 'aci']].drop_duplicates()
        print("\nUnique transaction types:")
        print(unique_txn_types)

        # Find applicable fee IDs
        applicable_fees = []

        for _, txn_type in unique_txn_types.iterrows():
            card_scheme = txn_type['card_scheme']
            is_credit = txn_type['is_credit']
            aci = txn_type['aci']

            for fee in fees_data:
                # Check if fee applies to this merchant and transaction type
                scheme_match = fee['card_scheme'] == card_scheme or fee['card_scheme'] is None
                account_type_match = not fee['account_type'] or belles_data['account_type'] in
fee['account_type']
                mcc_match = not fee['merchant_category_code'] or belles_data['merchant_category_code']
in fee['merchant_category_code']
                credit_match = fee['is_credit'] == is_credit or fee['is_credit'] is None
                aci_match = not fee['aci'] or aci in fee['aci']

                if scheme_match and account_type_match and mcc_match and credit_match and aci_match:
                    if fee['ID'] not in applicable_fees:
                        applicable_fees.append(fee['ID'])

        applicable_fees.sort()
        print("\nApplicable Fee IDs:")
        print(applicable_fees)
    else:
        print("\nNo payments found for Belles_cookbook_store on day 10 of 2023")
else:
    print("Belles_cookbook_store not found in merchant_data.json")
```

Figure 9: Agent step 6, part 1, where the agent joins the payment traffic information with the rules about fee calculations it has learned about. Step 6 is continued in 10.

Step 6

Execution Logs

Belles_cookbook_store characteristics:
Account Type: R
Merchant Category Code: 5942
Capture Delay: 1

Found 37 payments for Belles_cookbook_store on day 10 of 2023

Unique transaction types:

	card_scheme	is_credit	aci
6042	NexPay	False	F
7109	GlobalCard	False	A
7961	GlobalCard	True	D
10061	GlobalCard	True	G
23853	NexPay	True	D
35807	TransactPlus	True	G
36239	TransactPlus	True	A
45238	GlobalCard	True	C
58584	GlobalCard	False	F
64920	GlobalCard	False	D
78788	TransactPlus	True	F
79891	TransactPlus	True	D
96367	NexPay	True	F
98810	SwiftCharge	False	F
108163	GlobalCard	True	F
111835	NexPay	False	G
131861	TransactPlus	False	F

Applicable Fee IDs:
[29, 78, 121, 286, 304, 357, 359, 381, 427, 454, 456, 473, 477, 484, 502, 536, 572, 596, 602, 606, 642, 704, 709, 741, 792, 813, 861, 871, 895, 917, 944, 968]

Figure 10: Agent step 6, part 2, where the execution logs show the found fees by the agent.

Step 7

Agent

Thought: Now I have the information I need to provide the answer. I've found the applicable Fee IDs for Belles_cookbook_store on the 10th day of the year 2023. According to the analysis, there are 32 fee IDs that apply:
[29, 78, 121, 286, 304, 357, 359, 381, 427, 454, 456, 473, 477, 484, 502, 536, 572, 596, 602, 606, 642, 704, 709, 741, 792, 813, 861, 871, 895, 917, 944, 968]

I should format them according to the guidelines provided: "Answer must be a list of values in comma separated list, eg: A, B, C."

Code:
```py
# Format the answer as a comma-separated list
fee_ids = [29, 78, 121, 286, 304, 357, 359, 381, 427, 454, 456, 473, 477, 484, 502, 536, 572, 596, 602, 606, 642, 704, 709, 741, 792, 813, 861, 871, 895, 917, 944, 968]

# Convert integers to strings and join with commas
formatted_answer = ", ".join(str(fee_id) for fee_id in fee_ids)

# Provide the final answer
final_answer(formatted_answer)
```

Execution Logs

Final answer: 29, 78, 121, 286, 304, 357, 359, 381, 427, 454, 456, 473, 477, 484, 502, 536, 572, 596, 602, 606, 642, 704, 709, 741, 792, 813, 861, 871, 895, 917, 944, 968

Figure 11: Agent step 7 where agent produces a final answer compliant with the task guidelines.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We describe throughout the paper our benchmark design, dataset composition, evaluation strategy and task formulations. Furthermore, we report baselines results across multiple state-of-the-art LLMs and analyze failure modes we discovered. Finally, we release data and code for reproducibility and accessibility under HuggingFace spaces.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are described in Section 5. We discuss how our design choice of objective evaluation has the drawback that we can not evaluate some impactful use-cases where free-form outputs are involved. However, with our choice we avoid potential biases and infrastructure costs of LLMs-as-a-judge approaches. Additionally, we acknowledge that in order to stand as a benchmark that evaluates generalization in data analysis we must diversify the task domains (not just finance) as well as increasing data scale and complexity.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: Our work does not entail theoretical work that would require a proof.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We release full code for reproducing the baselines and running the evaluations in https://huggingface.co/spaces/adyen/DABstep/tree/main . Additionally, all benchmark data is publicly available here https://huggingface.co/datasets/adyen/DABstep (we only hide the ground truth answers to preserve the benchmark integrity). On top of this, in Section A.2, we also describe with pseudocode the evaluation algorithm, and, in Section 4, we detail the main high level steps involved to generate the baselines minimal scaffolding.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We release full code for reproducing the baselines and running the evaluations in https://huggingface.co/spaces/adyen/DABstep/tree/main . Additionally, all benchmark data is publicly available here https://huggingface.co/datasets/adyen/DABstep (we only hide the ground truth answers to preserve the benchmark integrity). On top of this, in Section A.2, we also describe with pseudocode the evaluation algorithm, and, in Section 4, we detail the main high level steps involved to generate the baselines scaffolding.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The only experiment we conduct is running a baseline agent, powered by different state-of-the-art LLMs, against the presented benchmark. We detail the task formulation and statistics of the tasks forming the benchmark that we run the agent against. The different sets of data in this case are Easy or Hard splits. Additionally, we provided a high level overview of the data sources the benchmark provides in Section A.3 but full details are disclosed in https://huggingface.co/datasets/adyen/DABstep. Additionally, reproduction of all experiments can be easily accomplished by running this baseline script, https://huggingface.co/spaces/adyen/DABstep/blob/main/baseline/run.py, with the desired set of arguments (model name, maximum steps per task, ...). Specifically, one must run such script with $max\_steps = 10$ to reproduce our results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Creating error bars for each of the baselines would be too computationally expensive given the high compute demands of LLMs.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: In Section 4.1 we share the GPU cluster used to deploy and benchmark open source LLMs but its unknown to us the cluster characteristics of closed source LLM providers so we just provide the inference costs in Table 1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

Answer: [Yes]

Justification: Given this is a submission to the Datasets and Benchmarks track, we are submitting in a single-blinded fashion without anonymizing any of the links to the benchmark artifacts.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: Our contribution is purely a benchmark and associated dataset constructed from non-sensitive, anonymized data. It does not involve personal or demographic attributes, decision-making systems, or any functionality that can be directly deployed. Hence, there is no clear pathway through which it could produce either beneficial or harmful societal effects.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The data we are releasing has no risk of misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Provided benchmark data, baseline code and evaluation code are all original from this paper. The only external assets we use are the LLMs which are appropriately cited. We also properly license our released assets under CC-BY-4.0 in section A.1.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes, the benchmark data and baseline experiments code hosted in Huggingface is accompanied with instructions to work with it. On top of this we provide a quickstart notebook linked from the benchmark leaderboard.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification:

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification:

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification:

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.