LOOKING BEYOND ACCURACY: SURFACING SILENT FAILURES IN AGENTIC DATA SCIENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models have significantly lowered the barrier to writing code. Naturally, this has also led to the development of agentic data science frameworks that promise automating the design and execution of end-to-end machine learning pipelines, from task formulation to model development. Existing evaluations of agentic data science (agentic DS) assess the downstream predictive performance and code correctness, providing preliminary insight in agentic DS capabilities. We argue that these evaluations overlook the fundamental requirements of data science: methodological data science knowledge and domain understanding. As such, only assessing downstream performance could mask methodological failures such as data leakage and inappropriate validation, limiting the development of robust agentic DS systems. In this paper, we evaluate agentic DS *beyond accuracy* from three different angles: 1) a large-scale analysis of failure modes in common agentic DS frameworks and models across 220 data science tasks, 2) a reusable and extensible test suite with curated data science tasks that test for six specific failures in a controlled manner, and 3) a qualitative assessment of agentic DS pipelines against expert solutions on Kaggle competitions. We find that the majority of agentic solutions contain at least one failure mode despite the complexity of frameworks. Our qualitative analysis shows that DS agent pipelines are generally lacking problem solving creativity and domain knowledge, compared to competition-winning data scientists. Moreover, we observe a positive correlation between the number of data leakage failures and predictive performance, illustrating the necessity to evaluate agents *beyond accuracy* We believe that principled failure mode analysis, as presented here, should be an integral part of agentic DS evaluations, and that the insights we surface are key in developing trustworthy agentic DS pipelines.

1 INTRODUCTION

The strong code generation capabilities in large language models (LLMs) have led to their wide adoption in software engineering (Jimenez et al., 2024). When connected to execution environments, LLMs can be deployed as agents for the autonomous completion of tasks, which has become an emerging practice across applications and industries (Pan et al., 2025). While coding agents generally improve with more capable backbone LLMs (Jimenez et al., 2024), understanding performance beyond end-to-end capabilities is key (Gu et al., 2025). Prior analyses of agentic code-based systems have surfaced common failure modes beyond code correctness, such as disobeying task specifications, misaligned agent interactions, and incorrect task verification (Cemri et al., 2025).

In parallel with coding agents, we observe an influx in the automation of data science pipelines through specialized *data science agents* as well (Guo et al., 2024; Grosnit et al., 2025; Du et al., 2026). Data science use-cases, particularly of a predictive nature, are traditionally characterized as multi-disciplinary processes that involve long-trained data scientists, domain experts, software and data engineers (De Bie et al., 2022). What constitutes a "good" data science pipeline is also inherently task- and data-dependent, and cannot be coupled with a hard verifiable signal. For example, model training and evaluation may require a temporal splitting procedure if the use-case demands temporal anticipation, while predictive performance metrics need to suit the target data distribution. These characteristics also translate into different evaluation protocols: whereas software is tested in an integrated verifiable manner, the assessment of (statistical) validity and utility in data science

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

is typically spread over the entire process and requires input from domain experts, engineers, and data scientists (De Bie et al., 2022; Cong et al., 2025). These characteristics, therefore, necessitate different evaluations beyond code generation capabilities.

An early large-scale evaluation of generic agents across diverse tasks common in industry, among which data science tasks, highlighted a significant difference where most LLMs had a success rate of zero on data science tasks compared to approximately 15% higher success rates on other tasks (Xu et al., 2025). Specialized DS agent benchmarks provide a more holistic analysis of metrics such as cost, efficiency, and success rates, surfacing the differences across models and DS agent frameworks (Huang et al., 2024; Jing et al., 2025; Nayak et al., 2026). However, *we argue that the potential presence of silent failures can give a distorted view of generalizability, hence making it risky to assess the actual capability of DS agents with downstream metrics only.* The development of reliable DS agents, therefore, demands more fine-grained evaluations beyond downstream performance metrics to understand their true failure modes and capabilities.

In this paper, we present ① the first large-scale analysis of common failure modes of various DS agents and models on 220 tasks, ② a reusable controlled test suite with curated tasks, and ③ a qualitative analysis of capabilities in comparison with human expert solutions, all grounded in data science best practices (Lones, 2024; Kumar et al., 2025). Across analyses, we find that the majority of agent-generated pipelines contain at least 1 data science failure mode regardless of the complexity data science agent framework (Figure 1). We also find that data leakage failure rates correlate with inflated predictive performance, underlining the necessity to evaluate agents beyond accuracy. Finally, our qualitative analysis shows that agents' solutions are weaker on task formulation and feature engineering compared with expert human solutions, illustrating a lack of problem-solving creativity and domain knowledge. We believe that principled failure mode analysis, as presented in this paper, should be an integral part of agentic DS benchmarks and evaluations. We also envision that the DS practices framework and insights we surface will be key in developing robust and trustworthy agentic DS pipelines.

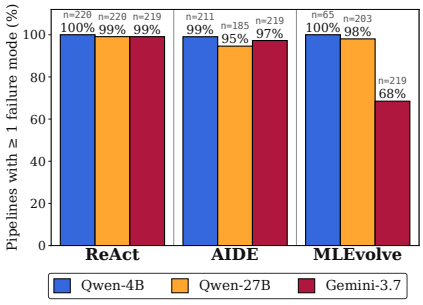


Figure 1: Percentage of pipelines with at least one failure mode per agent-model configuration.

2 Related Work

Automated data science. Automated Machine Learning (AutoML) systems address the need for automating the selection of model architectures and optimization of hyperparameters (Hutter et al., 2019), through frameworks like TPOT (Olson et al., 2016), AutoGluon (Erickson et al., 2020), and auto-sklearn (Feurer et al., 2015). By optimizing over a large search space, AutoML frameworks often match the performance of models engineered by machine learning experts (Hutter et al., 2019; Erickson et al., 2026), yet remain constrained to a predefined search space and narrow scope of the end-to-end data science workflow. The recent line of work on tabular foundation models (TFMs) intends to remedy the hyperparameter search and instead train and predict in one forward pass using a pre-trained in-context learning model (Hollmann et al., 2025; Qu et al., 2025). Given the time-consuming nature of data science and the limited human capacity, it has naturally become a new objective to automate many steps in the end-to-end data science workflow (De Bie et al., 2022). The increasing code generation capacity of LLMs has led to their deployment in autonomous data science agents, performing tasks across the entire workflow. Approaches for building and refining machine learning pipelines vary significantly. Some agents construct them in a single loop (Grosnit et al., 2025), while others iteratively refine them (Guo et al., 2024; Jiang et al., 2025; Zhang et al., 2026) or systematically explore the design space (Du et al., 2026). Our analysis includes DS-tailored agents, simpler ReAct agents, and AutoGluon as a reference AutoML implementation.

Agentic data science benchmarks and evaluations. Benchmarks have been proposed alongside the proliferation of research into data science agents. Their aim has mostly been to assess the predictive performance of the models these agents produce. These benchmarks predominantly build

on machine learning competitions sourced from Kaggle (Chan et al., 2025; Jing et al., 2025; Shu et al., 2025; Nie et al., 2026; Sun et al., 2026; Nayak et al., 2026; Huang et al., 2024), which provide reference solutions that can be used to define relative performance metrics. They focus on predictive performance and mostly do not analyze failures. Those that do report only high-level failure categories, mostly focused on execution errors such as incorrect tool use and coding mistakes (Nie et al., 2026; Shu et al., 2025). We instead analyze *how* data science agents implement predictive models by characterizing which practices they follow and where they fail.

Failure mode analysis in LLM agents. Analyzing code and execution traces of agentic systems has become essential for understanding these systems. Such analyses have yielded taxonomies of failure modes specific to multi-agent systems (Cemri et al., 2025), safety aspects (Bryan et al.), and alignment under adverse incentives (Li et al., 2026). A consistent finding across analyses is that the ability to write working code and correctly use tools is getting less relevant to the performance of agentic systems, whereas errors concentrate in planning and reasoning (Zhu et al., 2026) and understanding data (Zhu et al., 2026) and the broader context (Patel et al., 2026). Existing taxonomies and analyses are mostly proposed for code generation and thus focus on failures in agent coordination, grounding, and execution, providing an incomplete view of violations in data science methodology.

3 Evaluation Methodology

In this section, we introduce the methodology used to understand *why* data science agents fail. We first introduce the approach used to carry out large-scale analysis of data science failures using LLM-as-judge (Section 3.1), then we present the systematic test suites with curated tests (Section 3.2), and finally we outline the qualitative comparisons with high-performing solutions (Section 3.3).

3.1 Large-scale Failure Mode Analysis ⚖

Evaluating agentic data science across a large number of tasks through manual inspection is costly. Following previous evaluations of agentic systems (Cemri et al., 2025), we devise an LLM-as-Judge evaluation (Zheng et al., 2023) to assess whether generated pipelines exhibit specific failure modes. However, not all failure modes are applicable in every data science use case, and often depend on properties of the task, data, or pipeline itself. For example, ignoring temporal order when splitting the data constitutes a failure *only* when the data involves a temporal structure for which chronological separation is required, and is irrelevant in other cases.

To account for the task-conditional relevance of failure modes, we apply a two-stage LLM-based evaluation. In the first stage, the judge analyzes each generated solution to identify its pipeline structure and determine which failure modes are applicable. Specifically, it extracts whether the solution contains key pipeline components, including `data cleaning`, `feature engineering`, `validation`, `hyperparameter tuning`, `model comparison`, and `ensembling`, as well as whether it uses deep learning or stochastic models (`uses deep learning` and `uses stochastic model`). This structural information serves as a filter for determining the applicability of individual failure modes.

In the second stage, the judge evaluates the generated pipeline for the applicable failure modes. We evaluate all failure modes within a single LLM call, providing the pipeline structure extracted in the first stage as additional context. The judge produces a binary decision for each failure mode, indicating whether the corresponding failure is present in the solution, and, if so, extracting the lines of code where the failure mode occurs. Besides filtering out inapplicable failure modes per pipeline, this two-stage evaluation process separates pipeline understanding from failure detection and improves consistent performance across agents.

We inform the failure modes with commonly identified bad practices (Kumar et al., 2025; Lones, 2024), as shown in Table 1. These failure modes span the stages of data science pipelines . We broadly categorize them into "*data leakage*" modes, indicated using FM, and "*non-leakage*" modes, using FM. Detailed definitions for all failure modes are available in Appendix A.2. On top of the predefined failure modes, we added an "*other*" failure category for the judge to infer freely, accounting for any additional errors it might find.

Table 1: Failure modes used for the large-scale analysis. Identifier (ID) colors distinguish two categories: FM Data Leakage modes and FM Non-leakage modes. Failure modes marked with a dagger (†) are conditionally evaluated based on pipeline stage characteristics (e.g., whether deep learning or ensembling is used).

ID	Failure	Pipeline Stage	ID	Failure	Pipeline Stage
FM01	Wrong task formulation	Problem Formulation	FM09	Feature selection before splitting	Feature Selection
FM02	Peeking at test data during EDA	Exploratory Data Analysis	FM10	Wrong model type for task	Model Selection
FM03	Not cleaning data	Preprocessing	FM11	No regularisation in DL†	Model Training
FM04	Imputation before splitting	Preprocessing	FM12	Naive ensembling†	Model Optimization
FM05	Ignoring temporal order†	Data Splitting	FM13	Reusing test set across variants†	Model Evaluation
FM06	No train/val split	Data Splitting	FM14	Single eval of stochastic model†	Model Evaluation
FM07	No feature engineering	Feature Engineering	FM15	Using only one eval metric	Model Evaluation
FM08	Scaling before splitting	Preprocessing			

3.2 Curated Test Suite with Data Science Fallacies

Although the large-scale analysis provides quantitative insights into agent behaviour over naturalistic tasks, it cannot control what the correct practice should look like because the tasks were not designed with specific defects. To assess agents under more controlled conditions, we design a test suite in which each test targets a specific failure mode through a predefined perturbation. For example, to evaluate whether agents appropriately handle class imbalance, we construct a task with a highly imbalanced target distribution and examine whether the generated pipeline adopts suitable practices. Each test follows a "unit-test-style" design: the relevant pitfall is deliberately embedded in the task, and failure indicates that the agent did not identify or appropriately address the targeted issue. For each task, we additionally define multiple levels of hints that vary in instructional explicitness. The base condition provides no hint about the failure mode. We then increase the amount of guidance.

To determine whether an agent exhibits the appropriate behaviour for a given failure mode, we use an LLM-based judge to inspect the generated pipeline. For each test, we predefine the set of acceptable treatments corresponding to the targeted failure mode, and the judge checks whether any of these treatments is present. To obtain a stable estimation, we run each agent-task-hint combination five times and aggregate the results across runs. The six designed tests are shown in Table 2, with the formalization and evaluation defined in each row. These failure modes cover different stages of data science pipelines. We select these failure modes that satisfy three criteria: (1) can be instantiated through a controlled dataset or task design, (2) the expected methodological response can be specified clearly enough to support a controlled evaluation, and (3) the targeted decision can be isolated from other parts of the pipeline with limited confounding.

3.3 Skill Gap Analysis with Expert Data Science Solutions

The previous evaluations primarily focus on identifying practices that violate established data science principles, hence compromising the validity of a solution. However, the absence of such failures does not necessarily imply that a pipeline is highly effective. For example, an agent may produce a methodologically valid pipeline while ignoring a better task formulation, or a non-obvious feature transformation that can better capture the underlying structure of the task. To understand how data science *agents* compare to *human experts* in data science, we curate a set of Kaggle challenges that accept late submissions. For each task, we submit the predictions or solutions produced by agents to the leaderboard and compare their performance with high-performing solutions. The leaderboard provides an external reference for assessing generated pipelines. Moreover, we compare both the predictive performance and the modelling practice used by high-ranking solutions. This helps identify practices that contribute to performance gains even when their absence does not constitute a methodological failure. Additionally, this helps in understanding the true performance of agents-generated solutions on realistic challenges. As an additional reference, we compare the DS agent solutions with the solution of AutoGluon on selected tasks.

4 Why do State-of-the-art Data Science Agents Fail?

In this section, we present the insights discovered using the evaluation methods described in Section 3. We first describe the specific experimental setup used for the evaluation in Section 4.1,

Table 2: Six controlled failure-mode test suites. Each test introduces a targeted perturbation designed to expose a specific data science failure and is evaluated with a strict evaluation measure.

Failure Mode	Test Case	Assessment
TF01 **Ignoring Class Imbalance**	Dataset with high class-imbalance	Assess if the agent is alerting to the skew; classifier performing poorly on minority class
TF02 **Not Normalizing Formatting**	Inconsistent formatting of equivalent values (for example, string-encoded missing values)	Assess if an agent is aware of and reacts appropriately to the inconsistency during data preprocessing
TF03 **Not Separating Correlation from Causation**	Dataset includes a feature that strongly correlates with the target, without meaningful relationship	Assess if an agent is able to distinguish spurious relations by excluding spurious features
TF04 **No Domain Feature Engineering**	Prediction task requires a domain-specific feature transformation that is not apparent in the data	Assess if an agent is able to apply feature transformation utilizing its pre-trained domain knowledge.
TF05 **Data Leakage in Feature Selection**	Dataset includes features that are not expected to be used at training time.	Assess if an agent is deliberately excluding the features.
TF06 **Wrong Metric Selection**	Task with imbalanced classes yielding high accuracy if not evaluated with a proper class-balancing metric	Assess if an agent chooses the appropriate metric accounting for imbalance

then describe the results and main takeaways of the large-scale analysis in Section 4.2, subsequently show the results and insights in Section 4.3, and finally present the comparisons with expert solutions in Section 4.4.

4.1 Experimental setup

Datasets. The evaluation set contains 220 tasks paired with corresponding datasets and is used for the large-scale failure mode analysis ⚖. The tasks are drawn from five sources: 108 from STRABLE (Blayer et al., 2026), 44 from Kaggle, 37 from OpenML, 23 from UCI[1], and 8 from MLE-bench (Chan et al., 2025). The benchmark comprises 123 regression tasks, of which 122 have a single output and one has multi-outputs, and 97 classification tasks, of which 54 are binary and 43 are multi-class. To prepare the datasets, we perform an 80:20 split for the train and test sets. At runtime, agents are provided with both sets and are expected not to utilize the test set at training time, following the correct practice. Besides, a separate set of 17 tasks is used for the curated test suite 🧰, with 3 underlying datasets. Additional dataset details are available in Appendix A.3.

Data science agents. We evaluate three data science agents, namely a ReAct baseline (Yao et al., 2022), AIDE (Jiang et al., 2025), and MLEvolve (Du et al., 2026). MLEvolve is the highest-ranking open-source DS agent in MLEBench (Chan et al., 2025), while AIDE consistently achieved the strongest classification performance in the recent benchmark by Nayak et al. (2026). MLEvolve is a self-evolving multi-agent system that iteratively optimizes ML pipelines using methods such as progressive Monte Carlo Graph Search, retrospective memory, and hierarchical adaptive code generation. AIDE generates pipelines using an iterative code refinement process, leveraging evaluation feedback and execution logs to progressively improve generated solutions over multiple iterations. It consistently achieved the strongest classification performance in the recent benchmark by Nayak et al. (2026). Additionally, we include AutoGluon (Erickson et al., 2020), a widely used AutoML system, which serves as a strong non-agentic baseline based on heuristic-driven optimization and ensemble learning. Comparing these solutions across different failure modes allows us to obtain insights into what agent frameworks are really improving.

[1] Kaggle: kaggle.com OpenML: openml.org UCI: archive.ics.uci.edu

Agent	Model	*Wrong task formulation*	*Peeking at test data during EDA*	*Not cleaning data*	*Imputation before splitting*	*Ignoring temporal order*	*No train/val split*	*No feature engineering*	*Scaling before splitting*	*Feature selection before splitting*	*Wrong model type for task*	*No regularisation in DL*	*Naive ensembling*	*Reusing test set across variants*	*Single eval of stochastic model*	*Using only one eval metric*
ReAct	Qwen-4B	0.02	0.01	0.28	0.00	0.00	1.00	0.98	0.00	0.00	0.01	N/A	N/A	N/A	0.48	0.15
	Qwen-27B	0.01	0.12	0.04	0.13	0.36	0.05	0.50	0.00	0.03	0.00	N/A	0.49	0.00	0.95	0.89
	Gemini-3.7	0.02	0.07	0.30	0.06	0.48	0.40	0.38	0.00	0.08	0.00	N/A	0.90	N/A	0.04	0.14
AIDE	Qwen-4B	0.01	0.02	0.72	0.08	0.13	0.07	0.93	0.02	0.00	0.03	N/A	N/A	N/A	0.19	0.87
	Qwen-27B	0.03	0.08	0.21	0.03	0.15	0.01	0.71	0.00	0.02	0.00	N/A	0.81	N/A	0.08	0.85
	Gemini-3.7	0.01	0.15	0.45	0.13	0.47	0.00	0.12	0.03	0.14	0.00	N/A	0.46	0.00	0.03	0.88
MLEvolve	Qwen-4B	0.02	0.02	0.86	0.09	0.02	0.69	0.94	0.02	0.00	0.00	N/A	N/A	N/A	1.00	0.34
	Qwen-27B	0.01	0.08	0.17	0.03	0.17	0.01	0.16	0.00	0.01	0.00	0.00	0.51	0.06	0.76	0.87
	Gemini-3.7	0.00	0.10	0.10	0.10	0.36	0.00	0.00	0.01	0.06	0.00	0.00	0.04	0.01	0.13	0.43

Table 3: Occurrence rate of failure modes by agent and model over the pipelines to which the failure mode is applicable. N/A indicates that the number of applicable pipelines (as defined in Section 4.2) for a given failure mode is less than 20. Red colour in failure names indicates leakage modes and blue indicates non-leakage modes.

LLMs. The capacity of LLMs is important to agentic systems, since smaller models usually perform worse than larger LLMs in agentic tasks. To understand the impact of different models within agentic data science frameworks, we use three models of different capacities. We select `qwen-3.5-4b` and `qwen-3.8-27b`, which can be hosted locally on commodity hardware, and `gemini-3.7-flash`, which we select because it has a good performance-efficiency trade-off in agentic tasks . All LLM judges use the more capable `gemini-3.1-pro`. Additionally, for the curated test suite, we also evaluate `gpt-5.6-sol`.

Validation of dataset memorization robustness. Many tabular prediction tasks are publicly available online, and LLMs are likely to have seen them in their training data. To understand if a good predictive performance is partly because the underlying model has seen the tabular data, we run memorization tests using column header completion (Bordt et al., 2024; Lai et al., 2026). We then select a subset of tasks with high and low memorization rates, respectively. Following a similar procedure as in KramaBench (Lai et al., 2026), we then randomize the column names and rerun the agents on tasks with masked metadata and description. We also rerun the agents on the unmasked version as a placebo to obtain unbiased effects. A large difference in Δ_{plc} and Δ_{alt} for a high-memorization set indicates that the performance is derived from memorization and vice versa. Finally, we apply the non-parametric Wilcoxon test to compare whether there are significant predictive performance differences when tables are memorized. We test this on three DS agent frameworks paired with `gemini-3.7-flash` since it has the lowest data science failure rate. We present the results in Appendix A.1. All p values are non-significant in both sets with high and low memorization rates, indicating that memorizing tables does not significantly affect predictive performance.

4.2 Large-scale Analysis Surfaces Recurrent Silent Failures ⚖

Table 3 shows the prevalence of failure modes in agent frameworks and models. It shows that the specific failures the pipelines exhibit depend on how completely they cover the data science workflow. Failures happen either because agents perform operations incorrectly, or because they outright skip parts of the workflow (e.g., "*No val split*", "*No feature engineering*"). More capable models and elaborate frameworks shift failures from skipping steps to failures within these steps.

Failures occur across models and frameworks. All framework–model configurations have at least one failure mode that occurs in 43% or more of the pipelines. Agent frameworks reduce some of the failures of the ReAct baseline. With `gemini-3.7-flash`, both AIDE and MLEvolve reduce "*No val split*" from 0.40 to 0.00, and MLEvolve reduces "*No feature engineering*" from 0.38 to 0.00 and "*naive ensembling*" from 0.90 to 0.04. At the same time, frameworks also increase non-leakage failure modes. AIDE reports only a single evaluation metric in 85%–88% of its

pipelines regardless of the model, compared with 15% and 14% for ReAct with `qwen-3.5-4b` and `gemini-3.7-flash`, which is consistent with its design of scoring and selecting candidate solutions by a single validation metric (Jiang et al., 2025). Whether a framework improves adherence to DS practices depends on the model used. For example, AIDE lowers "*No feature engineering*" with `gemini-3.7-flash` (0.38 to 0.12), but raises it with `qwen-3.8-27b` (0.50 to 0.71).

Higher capacity models cover the data-science stages better but do not follow good practices. The low rates of leakage failures for small models do not indicate sound practices. ReAct with `qwen-3.5-4b` never designates a validation set and rarely performs feature engineering; consequently, the leakage modes that depend on the agent performing either do not occur ("*Impute before split*", "*Ignores temporal order*", "*Scale before split*", and "*Feature-selection before split*" are all 0.00). Under AIDE, `qwen-3.5-4b` does use a validation split, resulting in increased leakage across the related failure mode; for instance, *imputation before splitting* increases from 0.00 to 0.08. More capable models more frequently perform feature engineering and designate a validation set themselves, but they similarly show elevated leakage failure modes. For example, `gemini-3.7-flash` ignores the temporal order in more than a third of the pipelines over applicable tasks for all three agents. Overall, the results indicate that model capacity improves the *coverage* of the data-science workflow but that higher capacity models still commit errors within the expanded workflow. Thus, the raw number of failures over a pipeline is not indicative of the pipelines quality, since skipping part of the workflow removes the opportunity to fail within it.

Leakage extends beyond the taxonomy. We additionally analyse the failures flagged by the judge that fall outside of the fixed taxonomy, with full details reported in Appendix A.6. We find additional failures in 37% of the pipelines. Most commonly in the form of further leakage modes, namely leakage of validation into train data, e.g., where preprocessing or target encoding is fitted before cross-validation, and additional leakage of test into train data. The leakage modes show increasing prevalence with increasing model capacity, following the same trend as the leakage modes inside the taxonomy. Notably, 50% of the pipelines of MLEvolve with `gemini-3.7-flash`, the configuration with the fewest failures within the taxonomy, exhibit leakage between validation and train data, even though MLEvolve includes a dedicated agent for checking leakages (Du et al., 2026).

Failure types correlate with reported performance. Lastly, we analyse how the failures relate to predictive performance. For each task, we rank the pipelines of each framework–model configuration by the predictive performance the pipelines achieve on the test set *provided to the agent*. Since the configuration (i.e., a combination of model and the agent framework) influences the failures that occur in a pipeline as well as its performance, a direct comparison of raw ranks would primarily reflect differences between these configurations. We therefore compare each pipeline's rank to the rank expected for its configuration, and measure how this difference changes when a failure mode is present (Appendix A.5). Figure 2 shows that leakage and non-leakage modes affect performance in opposite directions. *All data leakage modes lead to improved rankings, which can be expected as leakage introduces information from the evaluation data into the training process.* This illustrates why downstream performance alone is not a good measure of agentic DS capability. The non-leakage failure modes have the opposite relation: more failure modes correlate with lower ranks. Not using a validation split (FM06) to assess model generalizability has the largest effect, deteriorating the performance by more than a full rank position on average. An exception to the pattern is that using only one evaluation metric (FM15) is associated with a better rank. We hypothesize that pipelines that evaluate a single metric optimize narrowly for the metric specified by the task, which is also the metric we rank by, possibly at the cost of robustness that a

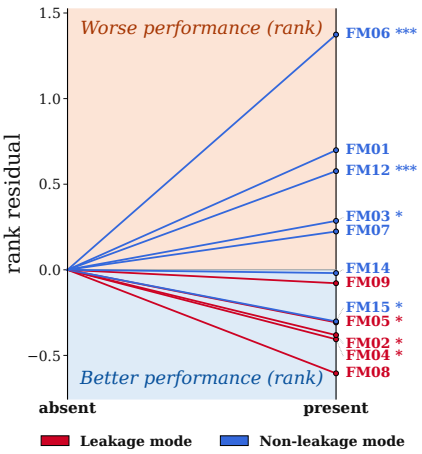


Figure 2: Adjusted effect of failure modes on residual rank in the *test* set that is accessible to the agent. An increase in rank indicates worse performance, while a decrease indicates improvement. * indicates significance.

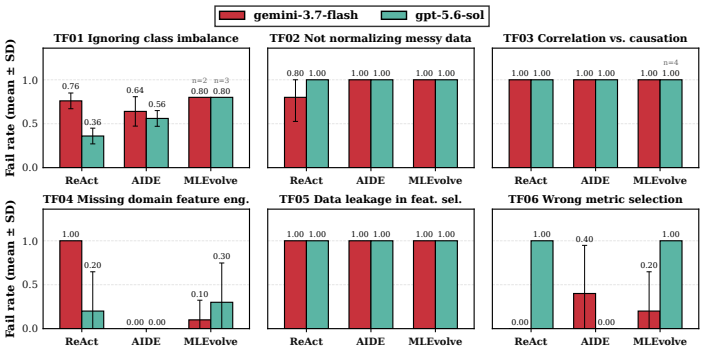


Figure 3: Results for the curated test suite of failure modes. Failure rate represents failure occurrences per failure mode, averaged over $n = 5$ runs. A failure rate of 1.0 means it occurs in every run.

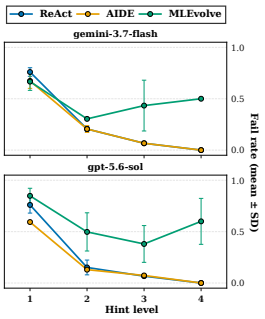


Figure 4: Average FMs per hint level. Hint level 1 is no hint; higher is more explicit)

single metric cannot capture. Overall, the results show that a better score on test data does not imply a better pipeline, and evaluating agents solely by their scores can reward leakage modes.

4.3 Controlled Evaluation Confirms Robust Patterns

Figure 3 reports failure rates on the six controlled tests for ReAct, AIDE, and MLEvolve using `gemini-3.7-flash` and `gpt-5.6-sol`. Figure 4 further examines how these failures change as increasingly explicit hints are provided. We also evaluated `qwen-3.5-4b` and `qwen-3.8-27b`; however, most runs failed to produce usable executable pipelines and therefore are excluded from the analysis. Next, we discuss our main findings in turn.

Some methodological failures persist across configurations. The clearest examples are correlation and causation reasoning (TF03) and leakage in feature selection (TF05), where every configuration reaches a failure rate of 1.0. Messy data normalization (TF02) shows similar behaviour: five configurations fail every run, while ReAct with `gemini-3.7-flash` fails at a rate of 0.8. In contrast, class imbalance (TF01), domain feature engineering (TF04), and metric selection (TF06) vary substantially across agents and models, indicating that different methodological decisions depend on both the underlying LLM and the agent framework.

Stronger models do not consistently yield better data science practice. Compared to `gemini-3.7-flash`, `gpt-5.6-sol` reduces failures for ReAct on class imbalance (TF01) and domain feature engineering (TF04), but the effect does not generalize across tests. This suggests that improvements in model capability do not transfer uniformly to methodological decision-making.

Explicit hints reduce failures, but not all. Figure 3 shows that ReAct and AIDE improve sharply as hints become more explicit, with failure rates approaching zero under the strongest guidance. This suggests that many failures arise from failing to recognize the relevant methodological constraint rather than from an inability to carry out the correct practice once prompted. However, MLEvolve behaves differently: although moderate hints reduce its failure rate, stronger hints do not lead to further improvement, and its failure rate remains substantially higher than ReAct and AIDE under more explicit guidance. *This points to a limitation of more advanced frameworks, where elaborate search and optimization may fail to maintain explicit constraints during the pipeline generation.*

4.4 Data Science Agents Rarely Discover Exert-Level Strategies

Table 4 compares agent-generated pipelines with AutoGluon solutions and the 1st-place Kaggle solutions across five challenges. Expert solutions remain ahead, and the performance gap varies by challenge. On the two Playground competitions, Heart Disease Prediction and Obesity risk, the gap is relatively small, while larger gaps remain on Single-Cell Perturbation, Problematic Internet Use, and Survival Predictions. This suggests that automated pipelines can be competitive on relatively accessible tabular tasks, while more specialized challenges leave large room for improvements.

Table 4: Leaderboard performance across Kaggle challenges. Scores reported as **private / public**. Best overall score is bolded, and the second-best is underscored. AutoGluon solutions are applicable only for challenges with CSV uploads (i.e., heart disease prediction and obesity risk) which are playground tasks and hence more simplistic in nature.

		Data Science Agents			AutoML	Baseline
Challenge	**Metric**	**ReAct**	**AIDE**	**MLEvolve**	**AutoGluon**	**1st Place (Human)**
Single-Cell Perturbations	MRRMSE ↓	1.1020 / 0.7690	0.8970 / 0.6600	0.8310 / 0.6170	– / –	**0.7290 / 0.5230**
Heart Disease Prediction	AUC-ROC ↑	0.9547 / 0.9531	0.9549 / 0.9533	0.9546 / 0.9531	0.9551 / 0.9536	**0.9554 / 0.9542**
Obesity Risk	Accuracy ↑	0.9065 / 0.9090	0.9059 / 0.9025	0.9071 / 0.9035	0.9074 / 0.9075	**0.9116 / 0.9234**
Problematic Internet Use	QWK ↑	0.4440 / 0.4680	0.4280 / 0.4640	0.4170 / 0.4480	– / –	**0.4820 / 0.5010**
Survival Predictions	C-index ↑	0.6502 / 0.6465	0.6536 / 0.6488	0.3155 / 0.3233	– / –	**0.7013 / 0.6981**

No agent dominates across tasks. MLEvolve performs best among the agents on Single-Cell Perturbations, ReAct on Problematic Internet Use, and AIDE on Survival Predictions. These results suggest that improvements introduced by the frameworks are not universal. Interestingly, MLEvolve performs poorly on Survival Predictions because it optimized on the inverted ranking, illustrating how a single error in decision-making can dominate downstream performance.

Case Study. To examine where this gap comes from, we compare the pipelines generated by ReAct, AIDE, and MLEvolve using `gemini-3.7-flash` with the 1st-place solution for the Kaggle *Problematic Internet Use* challenge (Figure 8 in Appendix A.7). Our test split contains `PCIAT-*` variables, including `PCIAT-PCIAT_Total`, from which the target `sii` can be derived, although these variables are not included in the actual Kaggle test set. ReAct and AIDE selected these leaked features and consequently achieved local QWK scores near 1.0, despite producing pipelines that would fail under the competition setting. *This again confirms that predictive scores during agents' optimization are not trustworthy.* Structural evaluation further reveals recurring weaknesses across frameworks: ReAct exhibited preprocessing leakage, limited feature engineering, arbitrary ensemble weights, and invalid test IDs; AIDE relied largely on raw features and mishandled categorical variables; and MLEvolve produced richer domain-informed features but still introduced leakage during encoding and relied on equal-weight ensembling. But this case study also indicates that *the gap to expert solutions goes beyond failures.* The 1st-place solution has fundamentally different, creative modelling choices. For example, it predicts the continuous `PCIAT_Total` rather than `sii` directly, applied optimized thresholds only at the final stage, used a Tweedie objective, adopted multi-seed 10-fold cross-validation, and combined diverse model families. These findings can inform future DS agent design to also focus on higher-level choices in problem formulation, task-specific objectives, validation design, feature engineering, and model diversity.

5 Conclusion

With the strong coding capabilities of large language models, they are increasingly integrated in data science agents. With higher-capacity models and advanced data science scaffolding, recent benchmarks show improving downstream performance. In this work, *we systematically evaluate failure modes in the pipelines that DS agents implement to get a more nuanced view of their true data science practices beyond accuracy*. We perform a large-scale analysis across 220 tasks and nine configurations of model–agent configurations, an extensible controlled test suite, and comparisons with expert real-world solutions. We surface many different failures that purely downstream performance evaluation masks. In fact, our results indicate that data leakage failure modes – which we identify to exist in many forms – can actually lead to stronger predictive performance on a test set. We also find that more sophisticated agent frameworks do not necessarily eliminate these failures. Despite the presence of dedicated data leakage checks in some frameworks, these can still exhibit leakage failures. This suggests that preventing such failures requires more than adding specialized components. Agent designs must reason more explicitly about methodological validity throughout the pipeline. Altogether, our findings reinforce the need to evaluate agentic data science *beyond accuracy*. We envision that our findings will improve agentic DS evaluations broadly, and inform the design of future agentic data science systems and models.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, to clean and reformat datasets, to support qualitative and thematic data analysis, and to interpret results. We have not used generative AI tools to design or provide feedback on research methodology or experiments, to generate synthetic data sets, to help develop theoretical models or conceptual frameworks, to propose or refine hypotheses, and to assist in the writing of proofs, to formulate mathematical claims, to provide critical ingredients for proving mathematical claims, and to assist with translation are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures, suggest experimental parameters, create and edit software code, create artifacts such as traces and LLM-judge logs evaluated in this project, and to edit the research paper to improve readability. We have reviewed all AI-assisted work. We validated each of the generated artifacts, and thoroughly reviewed and tested the generated code by one author. We validated every generated figure and assessed its validity. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Gioia Blayer, Myung Jun Kim, Félix Lefebvre, Lennart Purucker, Alan Arazi, Eilam Shapira, Roi Reichart, Frank Hutter, Marine Le Morvan, David Holzmüller, and Gaël Varoquaux. Strable: Benchmarking tabular machine learning with strings, 2026. URL https://arxiv.org/abs/2605.12292.

Sebastian Bordt, Harsha Nori, Vanessa Cristiny Rodrigues Vasconcelos, Besmira Nushi, and Rich Caruana. Elephants never forget: Memorization and learning of tabular data in large language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=HLoWN6m4fS.

Pete Bryan, Giorgio Severi, Joris de Gruyter, Daniel Jones, Blake Bullwinkel, Amanda Minnich, Shiven Chawla, Gary Lopez, Martin Pouliot, Adam Fourney, Whitney Maxwell, Katherine Pratt, Saphir Qi, Nina Chikanov, Roman Lutz, Sekhar Rao Dheekonda, Bolor-Erdene Jagdagdorj, Eugenia Kim, Justin Song, Keegan Hines, Daniel Jones, Richard Lundeen, Sam Vaughan, Victoria Westerhoff, Yonatan Zunger, Chang Kawaguchi, Mark Russinovich, and Ram Shankar Siva Kumar. Taxonomy of Failure Mode in Agentic AI Systems. URL https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Taxonomy-of-Failure-Mode-in-Agentic-AI-Systems-Whitepaper.pdf.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei A. Zaharia, Joseph Gonzalez, and Ion Stoica. Why Do Multi-Agent LLM Systems Fail? *Advances in Neural Information Processing Systems*, 38, 2025. doi: 10.52202/085713-4082. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b1041e52d3be19f0a9bc491657488e4a-Abstract-Datasets_and_Benchmarks_Track.html.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering. *International Conference on Learning Representations*, 2025:50466–50494, May 2025.

Tianji Cong, Julian Martin Eisenschlos, Daniel Gomm, Leo Grinsztajn, Andreas C Müller, Anupam Sanghi, Jan-Micha Bodensohn, Vadim Borisov, Michael Cochez, Katharina Eggensperger, et al. Position: unlocking the full potential of data science requires tabular foundation models, agents, and humans. 2025.

Tijl De Bie, Luc De Raedt, José Hernández-Orallo, Holger H Hoos, Padhraic Smyth, and Christopher KI Williams. Automating data science. *Communications of the ACM*, 65(3):76–87, 2022.

Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, Yifan Zhou, Xin Li, Jie Zhou, Liang He, Bo Zhang, and Lei Bai. MLEvolve:

A Self-Evolving Framework for Automated Machine Learning Algorithm Discovery, June 2026. URL https://arxiv.org/abs/2606.06473v1.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-tabular: Robust and accurate AutoML for structured data, 2020. URL https://arxiv.org/abs/2003.06505. arXiv: 2003.06505 [stat.ML].

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data. *Advances in Neural Information Processing Systems*, 38, 2026.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in neural information processing systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf.

Antoine Grosnit, Alexandre Maraval, Refinath S. N, Zichao Zhao, James Doran, Giuseppe Paolo, Albert Thomas, Jonas Gonzalez, Abhineet Kumar, Khyati Khandelwal, Abdelhakim Benechehab, Hamza Cherkaoui, Youssef Attia El-Hili, Kun Shao, Jianye Hao, Jun Yao, Balázs Kégl, Haitham Bou-Ammar, and Jun Wang. Kolb-Based Experiential Learning for Generalist Agents with Human-Level Kaggle Data Science Performance, September 2025. URL http://arxiv.org/abs/2411.03562. arXiv:2411.03562 [cs.LG].

Alex Gu, Naman Jain, Wen-Ding Li, Manish Shetty, Kevin Ellis, Koushik Sen, and Armando Solar-Lezama. Tasks, challenges, and paths towards AI for software engineering. In *ICLR 2025 Third Workshop on Deep Learning for Code*, 2025. URL https://openreview.net/forum?id=t0QaoJqjdw.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 16813–16848. PMLR, July 2024. URL https://proceedings.mlr.press/v235/guo24b.html.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, January 2025. ISSN 1476-4687. doi: 10.1038/s41586-024-08328-6. URL https://www.nature.com/articles/s41586-024-08328-6.

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*, volume 235 of *ICML'24*, pp. 20271–20309, Vienna, Austria, July 2024. JMLR.org.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.). *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer International Publishing, Cham, 2019. ISBN 978-3-030-05317-8 978-3-030-05318-5. doi: 10.1007/978-3-030-05318-5. URL http://link.springer.com/10.1007/978-3-030-05318-5.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138. arXiv: 2502.13138 [cs.AI].

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

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.

Smitha Kumar, Michael A Lones, Manuel Maarek, and Hind Zantout. Navigating pitfalls: Evaluating llms in machine learning programming education. *arXiv preprint arXiv:2505.18220*, 2025.

Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Om Chabra, Sivaprasad Sudhir, Anna Zeng, Anton Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, et al. Kramabench: A benchmark for ai systems on data-to-insight pipelines over data lakes. In *International Conference on Learning Representations*, volume 2026, pp. 142883–142912, 2026.

Miles Q. Li, Benjamin C. M. Fung, Martin Weiss, Pulei Xiong, Khalil Al-Hussaeni, and Claude Fachkha. A benchmark for evaluating outcome-driven constraint violations in autonomous AI agents, 2026. URL https://arxiv.org/abs/2512.20798. arXiv: 2512.20798 [cs.AI].

Michael A Lones. Avoiding common machine learning pitfalls. *Patterns*, 5(10), 2024.

Kartik Nagaraj Nayak, Nitishkumar Solpure, Niladri Mitra, Tom Zehle, Alexander Pfefferle, Omar Swelam, and Frank Hutter. Towards Benchmarking Agentic Data Scientists. In *AutoAI Meets Foundation Models*, August 2026. URL https://openreview.net/forum?id=UoYzHOKZaL.

Fan Nie, Junlin Wang, Harper Hua, Federico Bianchi, Yongchan Kwon, Zhenting Qi, Owen Queen, Shang Zhu, and James Zou. DSGym: A Standardized and Holistic Framework for Evaluating and Training Data Science Agents. June 2026. URL https://openreview.net/forum?id=2WrNPRW9Km.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pp. 123–137. Springer International Publishing, 2016. ISBN 978-3-319-31204-0. doi: 10.1007/978-3-319-31204-0_9. URL http://dx.doi.org/10.1007/978-3-319-31204-0_9.

Melissa Z Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, et al. Measuring agents in production. *arXiv preprint arXiv:2512.04123*, 2025.

Liana Patel, Siddharth Jha, Negar Arabzadeh, Carlos Guestrin, Ion Stoica, and Matei Zaharia. What happens when the model eats the stack? Rethinking the research agenda for data agents to withstand the bitter lesson, 2026. URL https://arxiv.org/abs/2609.03141. arXiv: 2609.03141 [cs.DB].

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267. PMLR, 2025. doi: 10.48550/arXiv.2502.05564. URL https://proceedings.mlr.press/v267/qu25d.html.

Fan Shu, Yite Wang, Ruofan Wu, Boyi Liu, Zhewei Yao, Yuxiong He, and Feng Yan. DARE-bench: Evaluating Modeling and Instruction Fidelity of LLMs in Data Science. In *The Fourteenth International Conference on Learning Representations*, October 2025. URL https://openreview.net/forum?id=eJV3JhJvZF.

Zhaoyan Sun, Shan Zhong, Daizhou Wen, Jiaxing Han, Guoliang Li, Ying Yan, Peng Zhang, Yu Su, Xiang Qi, Baolin Sun, Chengyuan Yang, Tao Fang, and Huaiyu Ruan. AgenticDataBench: a comprehensive benchmark for data agents, 2026. URL https://arxiv.org/abs/2607.01647. arXiv: 2607.01647 [cs.DB].

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Ruiyi Zhang, Peijia Qin, Qi Cao, Li Zhang, and Pengtao Xie. AIBuildAI: An AI agent for automatically building AI models. In *ICML 2026 Workshop: AI as a Tool for Mathematics, Computer Science, and Machine Learning*, 2026. URL https://openreview.net/forum?id=nckjXxN9WZ.

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. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Yuqi Zhu, Yi Zhong, Jintian Zhang, Ziheng Zhang, Shuofei Qiao, Yujie Luo, Lun Du, Da Zheng, Ningyu Zhang, and Huajun Chen. Why do open-source LLMs struggle with data analysis? a systematic empirical study. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, volume 40 of *AAAI'26/IAAI'26/EAAI'26*, pp. 35239–35247. AAAI Press, June 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i41.40831. URL https://doi.org/10.1609/aaai.v40i41.40831.

A Appendix

A.1 Details on memorization check setup

Table 5 presents the p values for the paired Wilcoxon test on both the high-memorised set and low-memorised set. The testing shows that the performance has no significant changes regardless of memorizing the tables.

Table 5: Wilcoxon signed-rank test of Δ_{alt} vs. the rerun randomized Δ_{plc}, computed separately for high- and low-memorization rate tasks across agentic frameworks. None of the six tests reaches significance at $\alpha = 0.05$, so dataset memorization does not measurably influence performance.

Agent	n (high)	Wilcoxon p (high)	n (low)	Wilcoxon p (low)
ReAct	17	0.890	18	0.212
AIDE	16	0.231	16	0.782
MLEvolve	16	0.597	17	0.890

A.2 Failure Modes Definitions

In Table 6 we include the definition of each of the failure modes evaluated in the large-scale analysis.

ID	**Failure Mode**	**Definition**
FM01	Wrong task formulation	The agent frames the modelling problem in a way that discards the signal the task actually provides.
FM02	Peeking at test data during EDA	Exploring test data during initial analysis creates untestable assumptions baked into model design.
FM03	Not cleaning data	Ignoring data duplication, missing values, and outliers/inconsistencies before model building.
FM04	Imputation before splitting	Performing mean/median imputation before splitting – leaks test statistics into training.
FM05	Ignoring temporal order	Randomly splitting time-series data without respecting temporal order – introduces look-ahead bias.
FM06	No train/val split	Model trained and evaluated on the same data – performance is a misleading overestimate.
FM07	No feature engineering	The agent uses raw columns as-is (at most basic imputation/encoding) with no derived features or domain-informed transforms, even where the task or column names suggest obvious ones.
FM08	Scaling before splitting	Fitting scaler on the full dataset before splitting – test statistics leak into training.
FM09	Feature selection before splitting	Selecting features on the full dataset before splitting – causes test leakage.
FM10	Wrong model type for task	Using a regression model for a classification task (or vice versa) – conceptual mismatch.
FM11	No regularisation in DL	Training deep learning models without dropout or weight decay – prone to overfitting.
FM12	Naive ensembling	When the agent combines multiple models, it uses an unweighted/equal-weight average or simple majority vote with no tuning of the combination itself, where a more sophisticated combination is straightforward given what's already computed.
FM13	Reusing test set across variants	Same test set used to select among multiple models – test set gradually leaks into model selection (sequential overfitting).
FM14	Single eval of stochastic model	Single evaluation of a stochastic/unstable model; results may not reflect true performance.
FM15	Using only one eval metric	Single metric gives an incomplete picture; different metrics reveal different failure modes.

Table 6: Failure modes definition.

A.3 Dataset Details

Table 7 shows the characteristics of the tasks we curated, including source information, number of tasks by type, median row, and median of features. Table 8 shows the domain distribution.

Source	N	Regr.	Bin.-cls.	Multi-cls.	Med. rows	Med. feat.
Kaggle	44	18	17	8	100,820	24
MLE-bench	8	6	2	0	6,976	13
OpenML	36	10	15	11	3,104	18
STRABLE	108	76	13	19	7,710	17
UCI	24	12	7	5	11,805	13
Total	220	122	54	43	8,693	18

Table 7: Dataset characteristics across the 220 benchmark tasks.

Domain	Tasks
Healthcare	50
Finance	47
Science	25
Retail	23
Social	21
Energy	17
Transportation	14
Education	11
Environment	8
Mixed	3
Bioinformatics	1
Total	220

Table 8: Domain distribution across the 220 benchmark tasks.

A.4 Failure Analysis

In Table 9 we show the number of applicable pipelines for each of the failure modes that require a specific pipeline stage or type of data to be present.

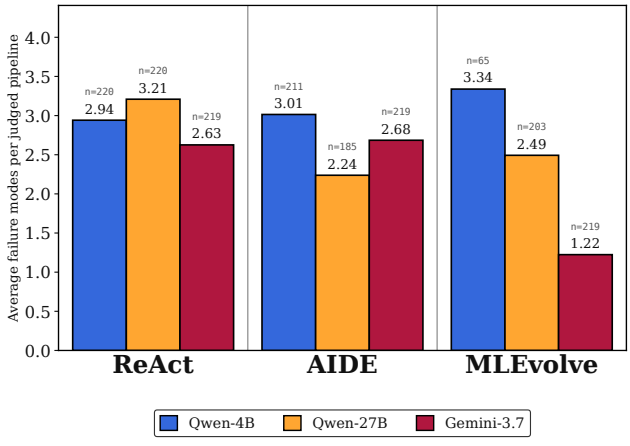


Figure 5: **Average failure modes per pipeline by agent-model configuration.**

Agent	Model	n judged	*Ignoring temporal order*	*No regularisation in DL*	*Reusing test set across variants*	*Single eval of stochastic model*	*Naive ensembling*
ReAct	Gemini-3.7	219	137	0	0	211	198
	Qwen-4B	220	138	0	0	219	1
	Qwen-27B	220	138	0	124	217	118
AIDE	Gemini-3.7	219	137	3	41	215	215
	Qwen-4B	211	133	0	0	190	3
	Qwen-27B	185	112	0	4	176	27
MLEvolve	Gemini-3.7	219	124	134	118	194	189
	Qwen-4B	65	45	2	1	21	2
	Qwen-27B	203	125	45	46	194	98
Total		1761	1089	184	334	1637	851
as % of total		–	61.8%	10.4%	19.0%	93.0%	48.3%

Table 9: Number of applicable pipelines per failure mode, by agent and model.

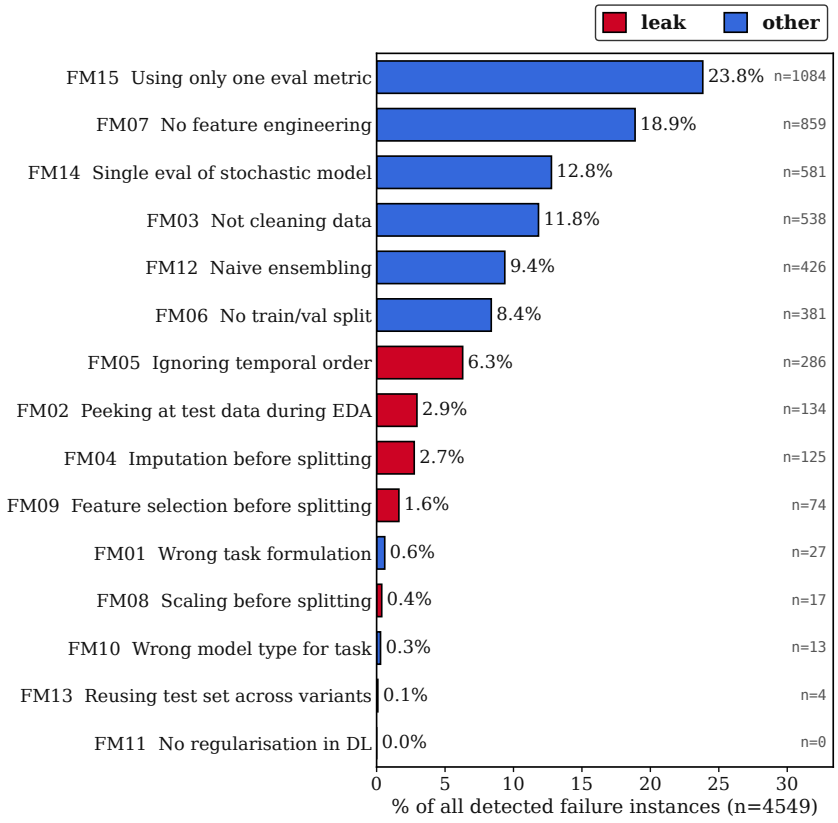


Figure 6: **Failure mode distribution. 4,557 failures detected over 1,761 pipelines, averaging 2.58 failures per pipeline.**

A.5 Effects of failures on predictive performance

To calculate the failure modes' effect on the predictive performance, we provide the correlation between the two variables. However, scores alone are not comparable because tasks use different metrics. Therefore, we use the rank among all solutions as the measurement of predictive performance. We run generated solutions on the tasks and calculate the score on the pre-processed test set. Finally, we rank the solutions across all 9 configurations. A naive correlation between failure modes and predictive rank is confounded by model and agent capability. For example, `qwen-3.5-4b` may exhibit fewer detected failures because it produces simpler pipelines, while also ranking poorly. We therefore estimate the effect of each failure mode after controlling for model and agent identity.

$$\text{rank} \sim \text{present} + C(\text{model}) + C(\text{agent})$$

For each failure mode, we fit a model that predicts a pipeline's rank from three things: whether the failure mode is present, which model produced it, and which agent produced it. Giving the model and agent their own separate adjustments, a model that's simply bad overall doesn't get mistaken for the failure mode itself being harmful. The one number we care about is how much presence of the failure mode shifts the rank, once that adjustment is made.

For visualization in Figure 7, we use the corresponding partial residual:

$$r_i = \beta_1 \text{present}_i + \hat{\varepsilon}_i.$$

Its group means are 0 when the failure is absent and β_1 when present, so the plotted difference directly represents the adjusted effect.

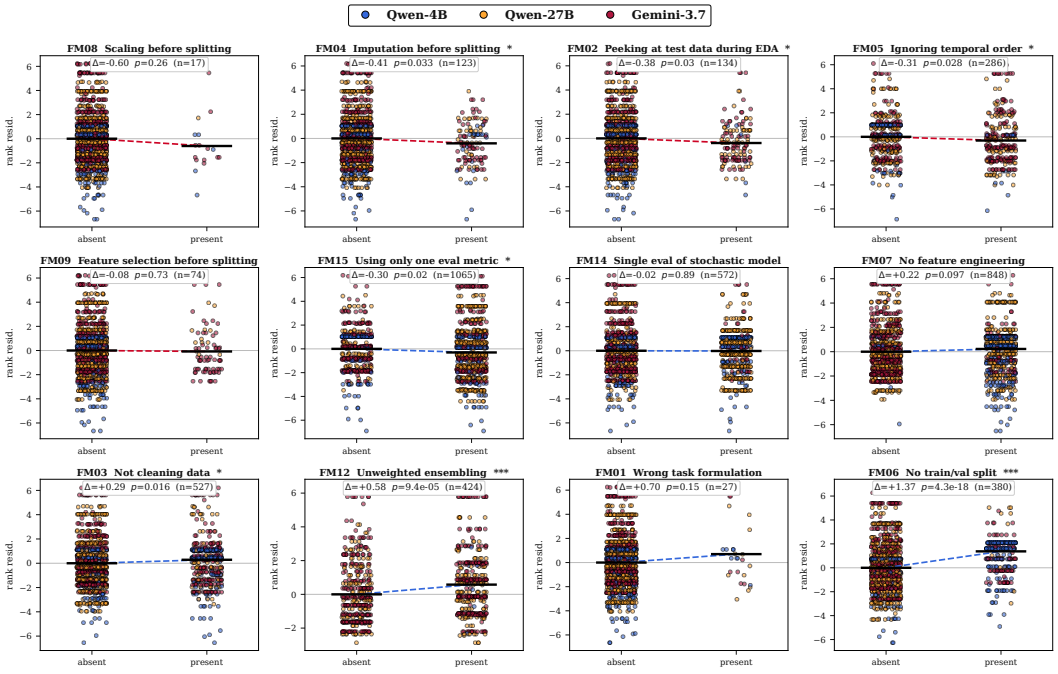


Figure 7: Effects of each failure mode on predictive performance measured in residual rank.

A.6 Failure Modes beyond the Taxonomy

In addition to the failure modes captured in the taxonomy, we evaluate the additional failure modes the LLM-judge reports as a list of free-text failures (Section 3.1). Across pipelines, the judge flags 2,294 additional failures. We filter these failures to only include actual violations of data science principles, filtering out other issues flagged by the judge, such as failures to produce code or implementation bugs, yielding 1,343 additional data science failures. We assign these failure modes into

Agent	Model	*Validation→train leakage*	*Test→train leakage*	*Flawed training protocol*
ReAct	Qwen-4B	0.00	0.00	0.03
	Qwen-27B	0.08	0.02	0.07
	Gemini-3.7	0.15	0.15	0.12
AIDE	Qwen-4B	0.16	0.01	0.29
	Qwen-27B	0.07	0.02	0.03
	Gemini-3.7	0.13	0.35	0.01
MLEvolve	Qwen-4B	0.05	0.00	0.05
	Qwen-27B	0.33	0.00	0.02
	Gemini-3.7	0.50	0.14	0.01

Table 10: Share of pipelines for which the judge reports the three most prominent failure modes beyond the taxonomy.

categories that we extract from manually screening them and then assign a category to each failure by independent keyword matching and through an LLM-judge (`qwen-3.5-9b`), where disagreements are manually resolved. On a sample of 59 reports that were labelled blindly, the final labels agree in 95% of the cases.

Additional failure modes. Table 10 shows the three most prominent categories. *Validation→train leakage* concerns instances where information from the validation data leaks into the trained model, without involving the test set. For instance, preprocessing or target encoding fitted on the full training data before cross-validation, early stopping on the fold the model is scored on, and ensemble weights fitted and scored on the same predictions. *Test→train leakage* encompasses leakage of test data into the training that falls outside of the taxonomy, such as transforms (e.g., label encoding, TF-IDF, or aggregated features) that are fitted on the training and test data. *Flawed training protocol* includes training without early stopping, predicting with the model of a single fold, and not retraining the final model on the full training data. Less frequent additional categories are inconsistent preprocessing of training and test data, nominal features treated as continuous, identifier columns used as features, target leakage through a feature, and incorrectly implemented metrics, each occurring in at most 6% of the pipelines.

Observations. The additional failures follow the same pattern described in Section 4.2. Validation→train leakage increases with model capacity, from 8% of the pipelines for `qwen-3.5-4b` to 16% for `qwen-3.8-27b` and 26% for `gemini-3.7-flash`. It is most frequent for MLEvolve (33% with `qwen-3.8-27b` and 50% with `gemini-3.7-flash`), whose pipelines frequently combine cross-validation with target encoding, early stopping, and blending, and it almost never occurs when there is no validation step, as for ReAct with `qwen-3.5-4b`. MLEvolve with `gemini-3.7-flash`, the configuration with the fewest failures within the taxonomy, has the highest share of pipelines with a failure falling outside the taxonomy (57%), indicating that it insufficiently accounts for leakage. Since the judge reports these failures without being prompted for them specifically, the shares should be interpreted as lower bounds on their actual prevalence.

A.7 Procedural comparisons between agentic solutions and expert solutions

In Figure 8 we extract the key steps and procedures across solutions generated by three agents and expert solutions from Kaggle.

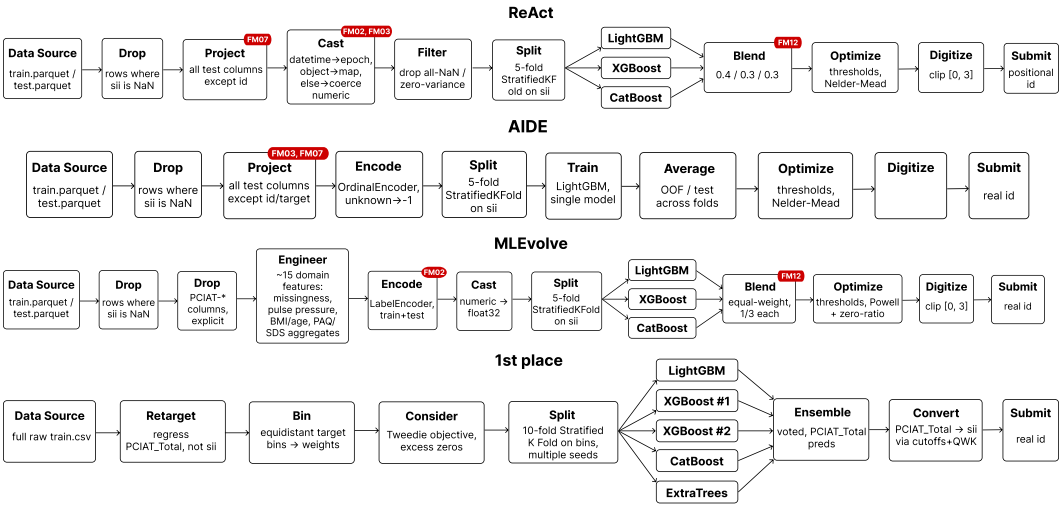


Figure 8: Exemplar pipelines of DS agents generated solutions using `gemini-3.7-flash` v.s. human expert solution.