

KAGGLEBENCH: A BENCHMARK FOR TABULAR PREPROCESSING REPAIR

Jonghu Jeong, Hsien-Hao Li, Sean Kraemer, Shreya Rao

University of Illinois Urbana-Champaign

{jonghuje2, hl128, seanmk2, shreya29}@illinois.edu

1 INTRODUCTION

Data preprocessing is central to practical machine learning: before model training, practitioners must handle missing values, encode categoricals, preserve relational grain, align temporal signals, and avoid leakage-prone transformations. These decisions are high-impact but context-dependent, so the same preprocessing step may be useful in one workflow and harmful in another.

Recent benchmarks evaluate language models and agents on code generation, machine learning engineering, and Kaggle-style problem solving. Most measure end-to-end outcomes, generated code, or model performance improvements. Those signals are useful, but they do not isolate whether a system made the right preprocessing decisions; a pipeline may score well while still containing brittle, leakage-prone, or conceptually incorrect transformations.

We introduce *KaggleBench*, a benchmark for preprocessing diagnosis and repair in tabular machine learning workflows. Our benchmark is grounded in high-quality Kaggle solutions and focuses on action-level reasoning rather than unconstrained pipeline generation. Each task provides Kaggle competition-level context, a fixed bank of candidate preprocessing actions, and a testcase-specific pipeline state. The agent must then repair the current state by adding beneficial actions and removing harmful ones. This formulation allows us to evaluate preprocessing *recovery*, *correction*, and *rollback* under realistic but controlled workflow states, while keeping scoring deterministic and reproducible.

The final benchmark contains 20 Kaggle-grounded tasks spanning wide sparse regression, relational classification, lookup joins, and temporal forecasting. Across these tasks we provide 80 scenario testcases and 20 human baseline outputs for `tbl_from_scratch`. The repository also contains automated outputs used by the companion agent paper; here we focus on human and rule-based results as benchmark-validation evidence.

2 RELATED WORK

Several existing benchmarks are related to our setting, but none directly evaluate expert-grounded preprocessing reasoning for tabular machine learning.

Chan et al. (2025) introduce MLE-bench, which evaluates autonomous agents on Kaggle-style machine learning engineering tasks using end-to-end competition outcomes. This is the closest prior benchmark in terms of task source and overall workflow, but its supervision signal is primarily leaderboard-oriented and outcome-driven, and therefore does not isolate whether an agent made the correct preprocessing or feature-engineering decisions.

Pietruszka et al. (2024) evaluates LLMs by asking them to generate feature-engineering code for tabular datasets and scoring the resulting improvement of a fixed XGBoost model. This benchmark is highly relevant to our application domain, but it still measures success mainly through downstream predictive gain rather than through action-level diagnosis, repair, or rollback.

Huang et al. (2024) propose DA-Code, an executable benchmark for agent-based data science code generation that includes challenging data wrangling and analytics tasks in realistic environments. While DA-Code captures multi-step planning and tool use, it focuses on code completion and task execution rather than on deciding which preprocessing actions should be proposed, discouraged, or reversed.

Gu et al. (2024) address the difficulty of evaluating open-ended data-science decisions, where multiple valid approaches and many surface forms can express the same underlying choice. Their design choice of using a curated, schema-validated answer space to obtain reproducible, action-level supervision directly motivates our use of a constrained candidate action bank. BLADE, however, targets exploratory scientific analysis rather than tabular preprocessing.

Finally, Zhang et al. (2025) present DataSciBench, a broader benchmark for LLMs in data science that introduces a semi-automated ground-truth pipeline and a Task–Function–Code evaluation framework. Although it covers a wider range of data-science workflows, it is still organized around task outputs rather than a fixed, schema-validated bank of expert preprocessing actions.

In contrast, our benchmark is designed around expert-grounded, constrained preprocessing repair. It derives candidate preprocessing actions from strong Kaggle notebooks and evaluates whether an agent can correctly edit a testcase-specific pipeline state by additions and removals. Our evaluation is based on reproducible, action-level assessment of diagnosis, correction, and rollback in context-dependent tabular preprocessing workflows.

3 BENCHMARK DESIGN

Task Domain and Scope Our benchmark evaluates whether an agent can diagnose and repair tabular machine learning preprocessing pipelines under a constrained action-selection setting. Each task provides Kaggle competition-level context, a set of candidate preprocessing actions, and a testcase-specific pipeline state. The agent must decide which harmful active actions should be removed and which beneficial actions should be added. This makes KaggleBench a structured pipeline-repair problem rather than an open-ended generation task. It covers missing-value handling, encoding, scaling, aggregation, temporal preparation, and leakage prevention.

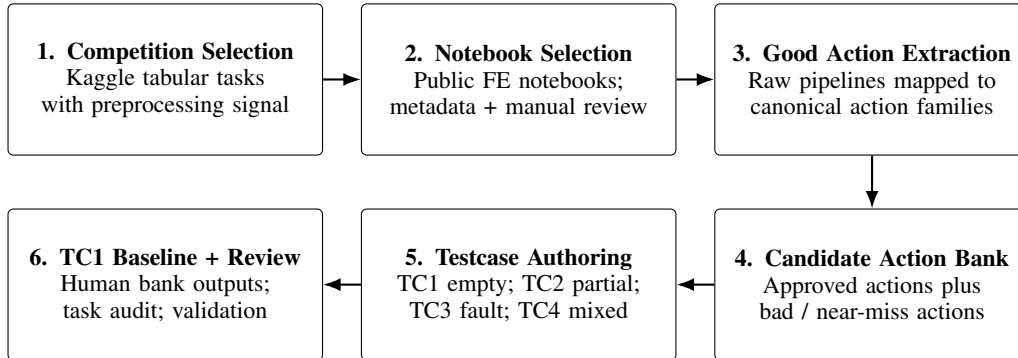


Figure 1: KaggleBench construction workflow, summarizing the sequence from competition selection through testcase authoring, human baseline creation, and review. Detailed authoring and validation checks are described in Appendix A.1.

Task Design Principles The benchmark was guided by four design principles, implemented through the workflow summarized in Figure 1. First, tasks are grounded in high-quality Kaggle solutions: we selected feature/preprocessing-focused public notebooks based on competition relevance, public metadata such as votes where available, and manual review for task-specific preprocessing evidence; actions without clear support were excluded or marked as bad or near-miss candidates. Second, contexts are intentionally imperfect, so agents must handle continuation and rollback in addition to recovery from scratch. Third, harmful but plausible alternatives test whether agents distinguish good preprocessing from leakage-prone, destructive, or incoherent choices. Fourth, curation remains human-reviewed: actions were iteratively refined through human–AI review to remove items that were too broad, inconsistent, or unfair to score.

Benchmark Structure Each task contains three components: a problem description, a candidate action set, and a collection of testcase-specific pipeline states. The problem description defines the

dataset and machine learning objective. The candidate action set contains both approved preprocessing actions and deliberately constructed bad actions. Each testcase specifies an initial state, such as an empty pipeline, a partially correct pipeline, a fault-injected pipeline, or a mixed history containing both correct and harmful decisions. For each testcase, the agent outputs a structured set of additions and removals. All 20 tasks are authored in the current schema-validated format, and Appendix Tables 3 and 4 summarize the final task set.

Example Tasks Representative tasks include Zillow Prize, a sparse real-estate regression task requiring parcel-level property joins and missingness handling (Kaggle, 2017b); Home Credit Default Risk, a relational credit-risk task requiring applicant-level aggregation from one-to-many history tables (Kaggle, 2018); and Recruit Restaurant Visitor Forecasting, a temporal restaurant-demand task requiring calendar and reservation alignment (Kaggle, 2017a). All use the same input schema, constrained output schema, and scoring rule; concrete schema details are in Appendix A.2.

Diversity, Coverage, and Task Variation We designed the benchmark to cover varied tabular preprocessing problems rather than many tasks with similar failure modes. It spans regression, binary classification, forecasting, and one multi-label/other task across flat tables, lookup-heavy settings, and multi-table relational workflows. Testcase variation comes from four fixed initial pipeline states: from-scratch, partial-good, fault-injected, and mixed-history. We avoid easy/medium/hard task labels because difficulty depends strongly on testcase state; preprocessing-complexity buckets are used only as a transparent task-level proxy, while TC1–TC4 define scenario-level difficulty.

4 EVALUATION PROTOCOL

Evaluation Setting We evaluate each testcase as constrained preprocessing repair over a fixed candidate action bank. Given a dataset, objective, and testcase-specific pipeline state, the agent predicts which actions to add and which active actions to remove. The evaluator derives correct edit targets from the candidate bank and current state. So correctness does not depend on free-form text generation. Because the benchmark uses a fixed action bank rather than open-ended answers, evaluation remains deterministic and comparable across agents.

Metrics and Aggregation For each run, we evaluate add-side and remove-side performance using precision, recall, and F1. Add-side metrics measure whether the agent recovers missing valid preprocessing actions, while remove-side metrics measure whether it correctly identifies harmful active actions. Our main summary metric is the task completion score,

$$\text{Task Completion Score} = 0.5 \times \text{Add F1} + 0.5 \times \text{Remove Recall}.$$

A run is counted as successful if its task completion score exceeds a threshold τ . We use $\tau = 0.5$ for the validation results reported in this paper.

We use Remove Recall in the main task-completion score because the primary rollback failure mode is failing to remove harmful active actions. However, over-removal is still monitored through remove-side precision/F1 in auxiliary reports, and future benchmark versions may use Remove F1 or final-state F1 as the primary rollback metric.

Because LLM-based agents are stochastic, each agent–testcase pair can be executed K times. We report single-run results and repeated-run robustness, including Pass@ K , success rate, and variance. Aggregate results use grouped (task, testcase, agent) units rather than raw runs; repeated runs are first averaged within each group. We also report runtime, token usage, cost, and tool usage when available.

Worked Zillow Scoring Example Zillow Prize is a real-estate regression task over transactions joined to property records, with `logerror` as the target. In `tc3_fault_injected`, the active context is {CA-000075, CA-000025, CA-000020, CA-000101}: a 2017 property join, aggressive target filter, zero-fill of valuation/size fields, and mixed-type sentinel fill. The primary gold add units are CA-000007, CA-000017, CA-000045, CA-000050, CA-000073, CA-000107, CA-000108, date group {CA-000010|CA-000079}, and outlier group {CA-000069|CA-000093}; equivalence groups count as one add unit, so the denominator is nine. Gold remove is the four active harmful actions.

Table 1: Worked Zillow tc3_fault_injected example using the real tc3_claude_code_try4 artifact.

Item	Add-side actions	Remove-side actions
Bank excerpt	CA-000107: 2016 parcel join; CA-000045: parse transaction date; CA-000050: median numeric imputation	CA-000075: 2017 join; CA-000025: aggressive filter; CA-000020: zero-fill; CA-000101: mixed-type sentinel fill
Prediction	predicted add {CA-000061, CA-000050, CA-000007}; true positives {CA-000050, CA-000007}; false positive CA-000061	predicted remove {CA-000025, CA-000020, CA-000101}; missed CA-000075

Thus Add Precision = $2/3$, Add Recall = $2/9$, and Add F1 = $1/3$. Remove Recall = $3/4$. The Task Completion Score is $0.5 \times 1/3 + 0.5 \times 3/4 = 13/24 = 0.541667$.

5 BENCHMARK VALIDATION

We validate the benchmark along three practical dimensions: task quality, artifact consistency, and baseline feasibility. First, task verification was iterative rather than one-shot: each task was reviewed using a coding-agent task audit (Appendix A.1), and the resulting feedback was reviewed by a human author. We revised task artifacts until candidate banks, testcase contexts, and evaluation semantics converged to stable specifications.

Second, we run repository-wide validation over the final artifacts. The artifact validator checks task, candidate-bank, testcase, output, and human-baseline schemas; validates candidate-bank composition rules; verifies testcase context coherence; and checks output references. In the final repository state, `uv run python eval/scripts/validate_artifacts.py` passes with 1,821 validated files, and `uv run python -m unittest discover -s eval/tests -p 'test_*.py'` passes 18 evaluator tests.

Third, baseline results show that the benchmark is executable while leaving substantial headroom. These outputs were scored by the same evaluator as agent outputs and were not used to define the derived gold targets.

Table 2: Main validation results. Detailed aggregate tables remain in the appendix.

Method	n	Add F1	Remove Recall	Task Completion	Success
Human baseline (tc1)	20	0.772	1.000 (vacuous)	0.886	1.000
Rule-based baseline	80	0.223	0.633	0.428	0.562

Because tc1 has an empty initial context, remove recall is vacuous; therefore the human tc1 result primarily validates add-side feasibility rather than rollback decisions. Human validation currently covers only tc1_from_scratch; future validation should include human or inter-rater checks for TC3/TC4 rollback decisions, where ambiguity is highest.

6 CONCLUSION

We introduced *KaggleBench*, a benchmark for preprocessing diagnosis and repair in tabular machine learning workflows. Grounded in strong Kaggle solutions, the benchmark represents tasks through fixed candidate action banks and testcase-specific pipeline states, and evaluates whether an agent can add beneficial actions and remove harmful ones under deterministic scoring. This focuses evaluation on action-level preprocessing reasoning rather than only end-to-end task outcomes.

KaggleBench provides a reproducible, expert-grounded way to compare data-science agents on context-dependent tabular workflows. The final artifact set includes complete task specifications, validation scripts, evaluation code, human baseline outputs, automated baseline outputs, README/setup instructions, and generated result reports. Future work should extend the benchmark beyond exact action-bank matching with richer semantic equivalence, broader human rollback validation, private or less web-exposed tasks, and downstream validation of selected preprocessing repairs.

REFERENCES

- 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. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=6s5uXNWGIh>.
- Ken Gu, Ruoxi Shang, Ruien Jiang, Keying Kuang, Richard-John Lin, Donghe Lyu, Yue Mao, Youran Pan, Teng Wu, Jiaqian Yu, Yikun Zhang, Tianmai M. Zhang, Lanyi Zhu, Mike A. Merrill, Jeffrey Heer, and Tim Althoff. BLADE: Benchmarking language model agents for data-driven science. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.
- 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*, pp. 13487–13521, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.748. URL <https://aclanthology.org/2024.emnlp-main.748/>.
- Kaggle. Recruit restaurant visitor forecasting. <https://www.kaggle.com/competitions/recruit-restaurant-visitor-forecasting>, 2017a. Accessed: 2026-04-06.
- Kaggle. Zillow prize: Zillow’s home value prediction (zestimate). <https://www.kaggle.com/competitions/zillow-prize-1>, 2017b. Accessed: 2026-04-06.
- Kaggle. Home credit default risk. <https://www.kaggle.com/competitions/home-credit-default-risk>, 2018. Accessed: 2026-04-06.
- Michał Pietruszka, Łukasz Borchmann, Aleksander Jędrasz, and Paweł Morawiecki. Can models help us create better models? evaluating llms as data scientists. *arXiv preprint arXiv:2410.23331*, 2024. URL <https://arxiv.org/abs/2410.23331>.
- Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. Datascibench: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897*, 2025. URL <https://arxiv.org/abs/2502.13897>.

A APPENDIX

Table 3: Twenty-task KaggleBench inventory.

Task	Domain	Type	Struct.	Metric	Rows	Feat.	Prep	G/B/P
Allstate Claims Severity	insurance	regr.	single	MAE	<i>med.</i> 188k	<i>high</i> 131	<i>low</i> 3	5/23/3
AMEX Default Prediction	finance	bin. cls.	single	AMEX score	<i>XL</i> 11.4M	<i>high</i> 192	<i>med.</i> 14	20/50/14
ASHRAE Energy Prediction	energy	forecast	lookup	RMSLE	<i>XL</i> 41.7M	<i>low</i> 17	<i>med.</i> 11	11/24/6
Predict Future Sales	retail	forecast	rel.	RMSE	<i>XL</i> 2.9M	<i>low</i> 12	<i>med.</i> 13	13/29/8
Elo Merchant Category Recommendation	finance	regr.	rel.	RMSE	<i>XL</i> 29.1M	<i>med.</i> 34	<i>med.</i> 14	16/50/14
Home Credit Default Risk	finance	bin. cls.	rel.	AUC	<i>XL</i> 27.3M	<i>high</i> 195	<i>high</i> 21	21/116/15
ICR Identify Age Related Conditions	biomedical	bin. cls.	lookup	Bal. log loss	<i>small</i> 617	<i>med.</i> 63	<i>med.</i> 9	9/24/3
IEEE Fraud Detection	fraud	bin. cls.	rel.	AUC	<i>large</i> 591k	<i>high</i> 434	<i>med.</i> 16	17/73/10
LISH-MOA	bioinfo.	other	single	MCLL	<i>small</i> 23.8k	<i>high</i> 876	<i>med.</i> 11	11/41/8
Mercedes Benz Greener Manufacturing	manuf.	regr.	single	R2	<i>small</i> 4.2k	<i>high</i> 378	<i>med.</i> 11	11/19/3
Optiver Trading At The Close	finance	regr.	single	MAE	<i>XL</i> 5.2M	<i>low</i> 16	<i>med.</i> 20	20/53/14
Porto Seguro Safe Driver Prediction	insurance	bin. cls.	single	Norm. Gini	<i>large</i> 595k	<i>med.</i> 57	<i>low</i> 7	14/61/7
Recruit Restaurant Visitor Forecasting	restaurant	forecast	rel.	RMSLE	<i>XL</i> 2.0M	<i>low</i> 17	<i>med.</i> 12	19/71/12
Restaurant Revenue Prediction	restaurant	regr.	single	RMSE	<i>small</i> 100k	<i>med.</i> 44	<i>med.</i> 11	11/15/5
Rossmann Store Sales	retail	forecast	lookup	RMSPE	<i>large</i> 1.0M	<i>low</i> 19	<i>med.</i> 12	16/45/12
Santander Customer Transaction Prediction	finance	bin. cls.	single	AUC	<i>med.</i> 200k	<i>high</i> 200	<i>low</i> 3	3/11/3
Sberbank Russian Housing Market	real estate	regr.	lookup	RMSLE	<i>small</i> 30k	<i>high</i> 391	<i>high</i> 22	22/102/8
Spaceship Titanic	transport	bin. cls.	single	Accuracy	<i>small</i> 8.7k	<i>low</i> 14	<i>med.</i> 17	15/36/11
Store Sales Time Series Forecasting	retail	forecast	lookup	RMSLE	<i>XL</i> 3.0M	<i>low</i> 17	<i>med.</i> 16	16/40/13
Zillow Prize 1	real estate	regr.	lookup	Zillow MAE	<i>XL</i> 3.0M	<i>med.</i> 66	<i>med.</i> 16	15/100/9

Notes. Type abbreviations: regr. = regression, bin. cls. = binary classification, forecast = forecasting. Struct. abbreviations: single = single table, lookup = multi-table lookup, rel. = multi-table relational. Rows report bucket and row count; Feat. reports dimensionality bucket and feature count; Prep reports preprocessing-complexity bucket and score. G/B/P gives good, bad, and effective primary good-unit counts after equivalence-group collapse.

Table 4: Task challenge summary for the 20 KaggleBench tasks.

Task	Core Challenge
Allstate Claims Severity	Sparse insurance regression with mixed categorical and numeric fields where the benchmark emphasizes conservative cleanup over broad feature synthesis.
AMEX Default Prediction	Repeated monthly customer statements require customer-level aggregation, sequence-aware summaries, and careful handling of a very large finance table.
ASHRAE Energy Prediction	Building-energy forecasting requires joining meter readings with weather and building metadata while preserving timestamp alignment.
Predict Future Sales	Retail forecasting over daily transactions requires aggregation to item-shop-month grain and time-aware feature construction.
Elo Merchant Category Recommendation	Card-level regression requires aggregating historical and new merchant transactions without leaking future purchase behavior.
Home Credit Default Risk	Multi-table credit-risk classification requires applicant-level aggregation from several one-to-many history tables.
ICR Identify Age Related Conditions	Small biomedical classification task where missingness, categorical encoding, and log-loss sensitivity make overprocessing risky.
IEEE Fraud Detection	Large fraud benchmark with sparse transaction and identity tables, high dimensionality, and many plausible but noisy encodings.
LISH-MOA	Wide multi-label bioinformatics data requires high-dimensional feature handling without collapsing treatment/control semantics.
Mercedes Benz Greener Manufacturing	Small, wide manufacturing regression task dominated by categorical encodings and dimensionality reduction decisions.
Optiver Trading At The Close	Finance time-series regression requires order-book feature preparation and leakage-safe temporal aggregation.
Porto Seguro Safe Driver Prediction	Anonymized insurance classification task where feature groups, missing-value sentinels, and categorical encodings must be handled carefully.
Recruit Restaurant Visitor Forecasting	Multi-table restaurant demand forecasting requires aligning reservations, visits, store metadata, and calendar features at the right grain.
Restaurant Revenue Prediction	Extremely small labeled training set with a much larger test set makes leakage control and cautious preprocessing central.
Rossmann Store Sales	Store-day forecasting requires joining store metadata, promotions, holidays, and temporal features without distorting the store-day grain.
Santander Customer Transaction Prediction	Dense anonymized classification task where the main benchmark target is minimal, defensible preprocessing rather than overengineered transforms.
Sberbank Russian Housing Market	Real-estate regression with macroeconomic lookup data requires date-aware joins and careful handling of many high-missing fields.
Spaceship Titanic	Passenger classification requires cabin decomposition, grouped passenger/family signals, and missing-value handling across mixed columns.
Store Sales Time Series Forecasting	Retail forecasting requires calendar, holiday, oil-price, and store metadata joins while preserving temporal evaluation semantics.
Zillow Prize 1	Real-estate residual regression requires parcel-level property joins, date parsing, and missingness handling across sparse property fields.

Table 5: Human baseline performance on tc1_from_scratch across 20 tasks. Remove recall is vacuous because tc1 has an empty initial context.

Method	n	Add P	Add R	Add F1	Remove R	Task Completion	Success
Human baseline (tc1)	20	0.844	0.753	0.772	1.000	0.886	1.000

Table 6: Deterministic rule-based baseline validation over all 20 tasks and four scenarios.

Method	n	Add P	Add R	Add F1	Remove R	Task Completion	Success	Avg Cost
Rule-based	80	0.289	0.229	0.223	0.633	0.428	0.562	\$0.000

Table 7: Agentic baselines validation over all 20 benchmark tasks and four scenarios.

Method	n	Add P	Add R	Add F1	Remove R	Task Completion	Success	Avg Cost
Single LLM	80	0.805	0.630	0.676	0.878	0.777	0.963	\$0.057
Generic agent	80	0.792	0.597	0.642	0.889	0.766	0.950	\$0.143
Claude Code	80	0.799	0.560	0.614	0.899	0.756	0.925	\$0.310
Proposed agent	80	0.816	0.615	0.676	0.822	0.749	0.913	\$0.196

A.1 TASK REVIEW PROMPT

You are reviewing one action-bank benchmark task for merge readiness.

Task root: <TASK_ROOT>

Competition slug: <COMPETITION_SLUG>

Use these documents as the source of truth:

- data/TASK_AUTHORING.md
- task_review_rules.md
- eval/README.md
- data/schema/*.json

Review these artifacts:

- <TASK_ROOT>/task.json
- <TASK_ROOT>/candidate_actions.json
- <TASK_ROOT>/notes.md
- <TASK_ROOT>/testcases/*.json
- <TASK_ROOT>/human_baseline/*
- <TASK_ROOT>/outputs/* if present

Your goal is to judge whether this task is internally coherent, aligned with the evaluation contract, deterministic to score, and reasonable as a benchmark of preprocessing diagnosis and repair over a fixed action bank.

Review checklist:

1. Contract and schema correctness

- Check that task metadata, candidate bank, testcases, outputs, and human baseline follow the documented schema and benchmark contract.
- Check that task.json contains only shared task context, and that any fixed benchmark slice or special assumption is stated in notes.md.

2. Candidate bank quality

- Check that the bank uses canonical action types and valid metadata.
- Check that primary good actions are conservative and truly core, not obviously model-specific unless justified.
- Check that good and bad actions do not collapse to the same behavior.
- Check that alternative good actions are grouped or narrowed when needed.
- Check that the bank has broad, plausible coverage with enough meaningful bad and near-miss candidates.
- Check that provenance and evidence are strong enough to support the curated good actions.

3. Scoring coherence and fairness

- Review equivalence_group, must_follow_action_ids, invalidates_action_ids, and conflicts_with_action_ids.
- Check that the benchmark still admits at least one valid execution order where expected.
- Flag any ambiguity, contradiction, or case where the same behavior could be rewarded and penalized.

4. Testcase and output consistency

- Check that testcase semantics are clear and still match the intended scenarios: from_scratch, partial_good, fault_injected, and mixed.
- Check that testcase difficulty comes from context_action_ids, not custom candidate subsets.
- Check that current contexts produce clean, unambiguous derived add/remove targets.
- If outputs are present, check add/remove membership, duplicate handling, add/remove overlap, final-state coherence, and artifact_refs existence.
- Check that the human baseline is honestly labeled as true bank-id authoring or a lossy migration.

```

5. Validation evidence
- If possible, run:
  - uv run python eval/scripts/validate_artifacts.py
  - uv run python -m unittest discover -s eval/tests -p 'test_*.py'
- Summarize any relevant failures.

Classify issues as blocking if they make scoring ambiguous,
contradictory, invalid under hidden ordering rules, or inconsistent
with the benchmark contract. Treat minor provenance, style, or
clearly labeled draft issues as non-blocking.

Output format:
1. Blocking issues
2. Non-blocking issues
3. Validation results
4. Final verdict: READY, READY WITH NON-BLOCKING ISSUES, or NOT READY

Keep the review concise, concrete, and evidence-based. Prefer curation
fixes over evaluator heuristics.

```

A.2 BENCHMARK INPUT/OUTPUT SCHEMA

Benchmark Input Schema. The benchmark input consists of shared task metadata, an agent-facing candidate action bank, and a testcase context. The full underlying artifact definitions are implemented in `task.schema.json`, `candidate_actions.schema.json`, and `testcase.schema.json`.

```

{
  "task": {
    "competition_slug": "string",
    "dataset": {
      "provider": "string",
      "competition": "string",
      "train_files": ["string", "..."],
      "test_files": ["string", "..."],
      "lookup_files": ["string", "..."],
      "sample_submission_files": ["string", "..."],
      "data_dictionary_files": ["string", "..."],
      "target_column": "string",
      "primary_key": "string"
    },
    "goal": "string",
    "task_characteristics": {
      "ml_domain": "string",
      "problem_type": "binary_classification | multiclass_classification
| regression | ranking | forecasting | other",
      "table_structure": "single_table | multi_table_lookup |
multi_table_relational",
      "dataset_size_bucket": "small | medium | large | xlarge",
      "dataset_size_row": 0,
      "feature_dimensionality_bucket": "low | medium | high",
      "feature_dimensionality_row": 0,
      "preprocessing_complexity_bucket": "low | medium | high",
      "preprocessing_complexity_row": 0,
      "analysis_notes": "string"
    }
  },
  "candidate_action_bank": {
    "competition_slug": "string",
    "generated_at": "string",
    "actions": [
      {

```

```

        "action_id": "string",
        "action_type": "string",
        "canonical_params": { "...": "..."},
        "eval_stage": "core_preprocessing | validation_protocol |
model_specific_preprocessing",
        "difficulty": "easy | medium | hard",
        "reasoning": "string"
    }
  ],
},
"testcase": {
  "testcase_id": "string",
  "competition_slug": "string",
  "task_ref": "string",
  "input": {
    "scenario": "from_scratch | partial_good | fault_injected | mixed",
    "context_action_ids": ["CA-000001", "..."]
  }
}
}

```

The `difficulty` field shown in candidate actions is action-level metadata; it is distinct from task-level difficulty, which is not encoded separately in the final artifacts.

Benchmark Output Schema. Each benchmark run produces one output artifact. The full output contract is implemented in `output.schema.json`.

```

{
  "competition_slug": "string",
  "testcase_id": "string",
  "agent_name": "string",
  "run_by": "string",
  "run_id": "string",
  "artifact_refs": [
    {
      "kind": "trace | log | metadata | notebook | notes | other",
      "path": "string",
      "description": "string (optional)"
    }
  ],
  "predicted_add_action_ids": ["CA-000001", "..."],
  "predicted_remove_action_ids": ["CA-000019", "..."],
  "time_spent_seconds": 0.0,
  "token_usage": {
    "input_tokens": 0,
    "output_tokens": 0,
    "total_tokens": 0
  },
  "notes": "string"
}

```

A.3 CANONICAL ACTION SCHEMA AND SYNONYM NORMALIZATION

The benchmark constrains preprocessing decisions to a fixed canonical action vocabulary stored in `data/schema/canonical_actions.json`. This schema is the source of truth for action naming, required parameters, optional parameters, aliases, and validation rules. In the current benchmark version, the vocabulary contains 40 canonical action families.

Canonical representation. Each benchmark candidate action is represented by two core fields: `action_type`, which names one canonical preprocessing family, and `canonical_params`, which stores the structured parameters for that action. This normalization lets us compare preprocessing intent at the action level rather than at the raw code-string level.

```
{
  "action_type": "GROUP_AGG",
  "canonical_params": {
    "group_by_columns": ["customer_id"],
    "target_columns": ["amount"],
    "agg_functions": ["mean"],
    "join_back": true
  }
}
```

Shared runtime fields. Many action families share a common set of runtime fields used for fitting, application scope, and error handling.

Table 8: Shared runtime fields in the canonical action schema.

Field	Role	Allowed values / meaning
mode	execution mode	fit, transform, fit_transform
fit_scope	fitting scope	train_only, full_data, none
artifact_id	fitted artifact handle	reusable identifier
random_state	reproducibility seed	integer when relevant
output_columns	explicit outputs	output feature names
handle_missing_	missing-column policy	strict, skip, create_default
input_columns		
on_error	failure policy	raise, warn, skip

Action families. For readability, we group the 40 canonical actions into six functional categories.

Table 9: Functional grouping of canonical action families.

Category	Representative actions
Selection and cleanup	CAST_DTYPE, DROP_COLUMNS, DROP_CONSTANT_COLUMNS, DROP_HIGH_NULL_COLUMNS, DROP_DUPLICATES, DROP_ROWS_WITH_MISSING, FILTER_ROWS, SAMPLE_ROWS, REPLACE_INF, CLIP_OUTLIERS
Missingness and categorical handling	IMPUTE_MISSING, CREATE_MISSING_INDICATOR, ENCODE_CATEGORICAL, ORDINAL_ENCODE, TARGET_ENCODE_OOF, MAP_CATEGORIES, RARE_CATEGORY_BUCKETIZE, BIN_NUMERIC
Numeric transforms and synthesis	SCALE_NUMERIC, LOG_TRANSFORM, POWER_TRANSFORM, APPLY_EXPRESSION, INTERACTION_FEATURE, HASH_CROSS, RATIO_FEATURE, POLYNOMIAL_FEATURE, PCA_REDUCTION, FEATURE_SELECTION
Time and sequence features	PARSE_DATETIME, DATE_PART_FEATURE, CYCLICAL_ENCODE, TIME_SINCE_REFERENCE, LAG_FEATURE, DIFF_FEATURE, ROLLING_WINDOW_FEATURE, EXPANDING_WINDOW_FEATURE
Relational and grouped features	GROUP_AGG, JOIN_LOOKUP
Text features	NORMALIZE_STRING, TEXT_VECTORIZATION

Action semantics. The schema distinguishes between *fittable transformers*, which learn state from data before reuse, and *pure transforms*, which can be applied directly. Examples of fittable transformers include `IMPUTE_MISSING`, `ENCODE_CATEGORICAL`, `TARGET_ENCODE_OOF`, and `SCALE_NUMERIC`. Examples of pure transforms include `DROP_COLUMNS`, `PARSE_DATETIME`, `JOIN_LOOKUP`, and `RATIO_FEATURE`.

Synonym normalization. The schema supports synonym normalization at two levels. First, each canonical action family may define local aliases used during notebook-to-schema normalization. For example, `JOIN_LOOKUP` includes aliases such as `merge_lookup` and `table_join`, while `CAST_DTYPE` includes `astype` and `convert_dtype`. Second, several older global action names are normalized into the current canonical vocabulary for backward compatibility.

Table 10: Deprecated aliases and their canonical normalization.

Deprecated alias	Canonical action	Fixed interpretation
COUNT_ENCODING	ENCODE_CATEGORICAL	method = "count"
FREQ_ENCODING	ENCODE_CATEGORICAL	method = "freq"
GROUP_AGG_FEATURE	GROUP_AGG	join_back = true
GROUP_AGG_JOIN	GROUP_AGG	join_back = true

Representative validation rules. The schema also encodes action-specific constraints. For example, `ENCODE_CATEGORICAL` restricts `method` to `onehot`, `label`, `freq`, `count`, or `hash`; `TARGET_ENCODE_OOF` restricts `cv_strategy` to `kfold`, `stratifiedkfold`, or `timeseriessplit`; and temporal operators such as `LAG_FEATURE`, `DIFF_FEATURE`, and `ROLLING_WINDOW_FEATURE` require explicit ordering columns to avoid ambiguous sequence semantics.

A.4 COMPLEXITY BUCKETING CRITERIA

The final benchmark does not assign a separate easy/medium/hard task-level difficulty label. Instead, it reports standardized dataset-size, feature-dimensionality, and preprocessing-complexity buckets, with preprocessing complexity serving as the closest task-level difficulty proxy. Dataset size is bucketed as *small* ($\leq 100,000$), *medium* ($\leq 500,000$), *large* ($\leq 2,000,000$), and *xlarge* ($> 2,000,000$). Feature dimensionality is bucketed as *low* (≤ 30), *medium* (≤ 100), and *high* (> 100). Preprocessing complexity is bucketed as *low* (≤ 8), *medium* (≤ 20), and *high* (> 20). Table 3 reports these standardized buckets.

CHANGELOG FROM DRAFT

Since the draft, we completed the full 20-task KaggleBench artifact set, replacing the earlier partial version. We enlarged and simplified the benchmark construction figure, moved aggregate human baseline validation results into the main paper, and added a fully worked Zillow example showing dataset context, candidate good/bad actions, testcase state, gold edit targets, and Task Completion Score computation. We also clarified the interpretation of the human `tc1` baseline, documented repository-wide validation results, and added metric limitations around remove-side scoring.