Reducing Confidently-Wrong Answers in LLM Data-Analytics Agents: A Failure-Mode Study and a Lightweight Verification Layer

Arya
AskData — Independent Research Project
github.com/Arthanz/askdata · askdata-arthanz.streamlit.app

Abstract

Abstract—Large language model (LLM) agents that answer natural-language questions over relational data ("chat with your data") are now easy to build and demo, but hard to trust. Existing text-to-SQL benchmarks measure whether a system *can* produce a correct query. They say little about what the system does when it *cannot*, which in an interactive analytics setting is what determines whether users can rely on it. We study the failure mode that matters most in that setting: *confidently-wrong answers*, where the agent returns a plausible-looking but incorrect result with no signal of uncertainty. On a 150-question golden set over the public Olist e-commerce dataset (covering lookup, aggregation, multi-join, temporal, deliberately *ambiguous*, and deliberately *unanswerable* questions), naive LLM agents produce confidently-wrong answers on 15–25% of questions across five generators, from a 3B local model to a frontier API. We add a lightweight, model-agnostic verification layer of static schema checks, result sanity checks, an LLM judge that asks *"does this SQL answer this question?"*, and a bounded repair loop, which assigns every answer a status of *trusted*, *repaired*, or *abstained*. Verification lowers the confidently-wrong rate on every generator, by up to 69% relative (for example 21.8% to 6.7% on a mid-tier model), at roughly 1.5× tokens and negligible added latency, with no fine-tuning or schema-specific engineering. A generator–judge factorial isolates *why* it works: swapping only the judge on a fixed weak generator cuts confidently-wrong answers 5× (10.9% to 2.2%), while upgrading the generator under a fixed judge barely moves it. Verification quality is largely a property of the judge, not the generator, which implies a practical architecture where a cheap or on-premise generator under a thin layer of strong-judge calls approaches frontier-level trustworthiness. We release the system, the 150-question golden set, and the evaluation harness for reproduction.

1. Introduction

Text-to-SQL generation has improved to the point where a competent demo can be assembled in an afternoon: give a model the schema, ask a question, execute the query, display the result. The gap between that demo and a tool an analyst can rely on is not primarily generation quality. It is that the system has no notion of when it is wrong. A query that executes successfully and returns a plausible number is indistinguishable, to the user, from a correct answer. When the SQL silently answers a slightly different question (wrong filter, wrong grain, missing join) the result is a *confidently-wrong* answer, which in a business setting is strictly worse than an error message, because it gets pasted into slides.

Public benchmarks under-measure this. Spider [1] and BIRD [2] score execution accuracy on answerable questions. They do not ask what a system does with a question the schema cannot answer, or an ambiguous question with several defensible readings. Interactive analytics encounters both constantly.

Contributions.

1. A *failure-mode study*: a golden evaluation set over a public e-commerce dataset whose tiers include ambiguous and unanswerable questions, plus a taxonomy of the errors a naive agent actually makes (§6).
2. A *lightweight verification layer* of static identifier checks, result sanity checks, an LLM SQL-judge, and a bounded repair loop, which attaches a *trusted / repaired / abstained* status to every answer (§3).
3. *Measurements* of the trade-off: the reduction in confidently-wrong answers against the token and latency overhead, with ablations showing which check does the work (§5).
4. A *generator–judge factorial*: by decoupling the judge model from the generator, we test whether verification quality is a property of the judge rather than of the pipeline. To our knowledge this is the first such decomposition for text-to-SQL verification, with direct consequences for deploying cheap local generators under a thin layer of strong-judge calls.

The contribution is deliberately not a new generator. It is an evaluation and an intervention that any text-to-SQL stack can adopt.

2. Related work

Text-to-SQL generation and its benchmarks. Semantic parsing of natural language into SQL has a long history; the modern era was defined by Spider [1], which introduced cross-domain evaluation over 200 databases and made execution-based metrics standard, later hardened against false positives by distilled test suites [3]. BIRD [2] scaled the setting to larger, noisier databases and added execution-efficiency concerns, and its leaderboard has been dominated by LLM-based systems since. Rajkumar et al. [4] showed that a large model with no fine-tuning is already a strong text-to-SQL baseline and began cataloguing its failure modes. Prompting-based methods now define the state of the art: DIN-SQL [5] decomposes the task into sub-problems with a self-correction stage, DAIL-SQL [6] systematically benchmarks prompt designs and approaches human performance on Spider, and more recent agentic pipelines add schema retrieval and pruning (CHESS [7]) or multiple collaborating agents, including a query-refining agent close in spirit to our repair loop (MAC-SQL [8]). All of this work measures whether a system *can* produce correct SQL. None of it models what the system should do when it cannot: every question in Spider and BIRD has an answer, so a system that never abstains is never penalized. Even Spider 2.0 [9], which raises difficulty to realistic enterprise workflows, keeps every task answerable.

Reliability and unanswerable questions. A smaller thread addresses exactly that gap. EHRSQL [10] was, to our knowledge, the first text-to-SQL benchmark to include unanswerable questions (about a third of its validation and test splits), arguing that hospital deployments cannot tolerate confident guesses; its 2024 shared task made abstention a first-class part of the evaluation. TrustSQL [11] generalizes this into a reliability benchmark by re-annotating three datasets with infeasible questions and scoring models with an explicit penalty for wrong answers over abstentions. In open-domain QA, Feng et al. [12] show that letting models abstain when they detect a knowledge gap measurably reduces hallucination. Our work is complementary: rather than proposing another benchmark, we contribute (i) an *intervention*, a model-agnostic verification layer that any text-to-SQL stack can adopt without fine-tuning, and (ii)

an evaluation grounded in a realistic business-analytics schema, with an error taxonomy connecting failure classes to the checks that catch them. Where TrustSQL scores a model's own abstention decisions, we measure how much an external verification layer improves a generator that was not trained to abstain.

Self-verification and critique. Hallucination in generative models is well documented [13], [14]. Several lines of work show LLMs can usefully critique their own or other models' outputs: SelfCheckGPT [15] detects hallucination by sampling multiple generations and measuring consistency; self-consistency decoding [16] aggregates sampled reasoning paths; Self-Refine [17] and Reflexion [18] feed a model's critique back into regeneration, a loop our repair mechanism instantiates for SQL; CRITIC [19] grounds the critique in external tool calls, the closest in spirit to our judge, which sees the executed result rather than just the query text. A related calibration result, that models mostly know what they know [20], is what makes an LLM judge plausible in the first place. LLM-as-a-judge evaluation was validated (and its biases catalogued) by Zheng et al. [21] and has since been surveyed in depth [22]. We differ in *where* the judge sits: not as an offline evaluation metric but as a gate in the serving path, with its verdict deciding whether an answer reaches the user.

Selective prediction. Declining to answer under uncertainty is an established capability in QA. Kamath et al. [23] trained calibrators for selective question answering under domain shift, and Cole et al. [24] study abstention on ambiguous questions. We port this framing to interactive analytics, where the asymmetry is stark: an abstention costs the user a rephrase, while a confidently-wrong number can silently enter a business decision.

3. System

AskData is a from-scratch agent loop (no framework), in the reasoning-and-acting style of ReAct [25], running over DuckDB: schema-aware prompt, SQL generation, read-only execution, verification, answer, with the verification verdict driving a repair loop.

Generation. The prompt contains the introspected schema (tables, columns, types, row counts), explicit join keys, and business notes (for example the per-order vs. per-person customer-ID distinction in Olist). Two behaviors are prompted, and both are *checked* downstream rather than assumed: the model must output `ABSTAIN` when the schema cannot answer the question, and must state an `-- assumption:` comment when it resolves an ambiguity.

Verification. Three check families run after generation:

1. **Static (no LLM):** the query must be a single read-only SELECT; every identifier must resolve to a schema table/column or a query-local alias. Catches invented columns before execution.
2. **Sanity (no LLM):** non-empty result, not all-NULL, no negative values in count/total-like columns.
3. **Judge (one LLM call):** given the schema, question, SQL, and a result preview, a model answers *"does executing this SQL genuinely answer this question?"*, targeting the silent-mismatch failure mode that execution cannot catch.

Repair and status. Any failed check becomes feedback for a regeneration attempt (max 3). Answers passing on attempt 1 are *trusted*; on a later attempt, *repaired*; if attempts are exhausted or the model declines, the agent *abstains* with the reason. The baseline used in §5 is the same agent with verification off: the first executable query wins and is always presented as trusted, which is exactly what most deployed demos do.

4. Evaluation setup

Dataset. Olist Brazilian E-Commerce [26] (public, ~100k orders, 8 relational tables). Experiments run on the full dump; the repo also ships a schema-identical synthetic sample so the harness is reproducible with zero downloads.

Golden set. 150 questions in six tiers: lookup (20), aggregation (26), multi-join (30), temporal (24), ambiguous (18), unanswerable (32, or 21% of the set, following EHRSQL's precedent of weighting unanswerable questions heavily [10]). Answerable questions carry gold SQL, executed against the same database at eval time, so gold answers are correct by construction. Unanswerable questions (profit margin, churn, marketing channel, returns, conversion, customer age) reference data that does not exist in the schema; the correct behavior is abstention. Ambiguous questions accept either the modal business interpretation or an abstention.

Scoring. Result equivalence is order- and column-name-insensitive (bag-of-values per row, rows sorted, floats rounded), the standard execution-match relaxation; a scalar gold answer contained in a one-row result also counts. An optional LLM answer-equivalence judge (disabled by default) provides a second score for shaped-differently answers; we report both.

Metrics. Accuracy (overall and answerable-only); *confidently-wrong rate* (answers presented as trusted/repaired that are wrong, the headline); hallucinated-unanswerable count; abstention rate; repair success rate; attempts, latency, and token overhead.

Configurations. Baseline (verification off) vs. AskData (verification on). Each configuration runs three times per model; we report mean ± sd across runs and an exact McNemar test on paired per-question confidently-wrong outcomes.

Model selection. Models were chosen to span deployment tiers while controlling the confounds that cross-vendor comparisons usually carry:

– *Qwen2.5-Coder-3B (local, via Ollama)*: the on-premise/privacy tier. Organizations in regulated settings often cannot send data to external APIs. It is also the weakest generator in the study, anchoring the capability axis; we use a code-specialized model so that small-model results are not an artifact of evaluating a generalist.

– *gpt-oss-120B (open weights, via Cerebras)*: the strongest self-hostable open model, testing whether openness at scale behaves like a closed frontier model.

– *GPT-5.4-nano / -mini / -full*: a *within-family capability sweep* with the same lab, generation, and training recipe, so generator capability is the only variable. Prior comparisons across vendors confound capability with training data, alignment style, and API behavior.

– *GPT-4o-mini*: a previous-generation anchor and, in practice, the most widely deployed budget API model.

Generator–judge factorial. Self-verification couples two distinct abilities: *generating* SQL and *judging* it. Because our judge is an independent model call, we can decouple them: a 2×2 factorial crossing {weak (3B), strong (5.4-mini)} generators with {weak, strong} judges, holding every prompt and check constant. If the verification benefit tracks judge capability rather than generator capability, the deployment implication is direct: run a cheap local generator and spend a small API budget on judge calls only.

5. Results

All numbers are means over three runs per configuration on the 150-question golden set; the tables are produced by `python -m eval.aggregate` over `eval/reports/`. The confidently-wrong rate (CW) is the fraction of all 150 questions answered, with status *trusted* or *repaired*, but scored wrong.

5.1 Verification reduces confidently-wrong answers on every model

Table 1 reports each generator judging its own SQL (the standard self-verification setting), baseline vs. verification-on.

Table 1 — confidently-wrong rate, self-judge (mean ± sd over 3 runs)

generator	baseline CW	verified CW	relative Δ	McNemar p
Qwen2.5-Coder-3B (local)	0.151 ± 0.010	0.109 ± 0.028	−28%	0.053
GPT-4o-mini	0.211 ± 0.014	0.162 ± 0.010	−23%	0.0002
GPT-5.4-nano	0.247 ± 0.014	0.147 ± 0.021	−41%	—
GPT-5.4-mini	0.218 ± 0.008	0.067 ± 0.006	−69%	0.0002
GPT-5.4-full	0.200 ± 0.007	0.167 ± 0.017	−17%	0.0059

Verification lowers the confidently-wrong rate on all five generators. The reduction is statistically significant (exact McNemar on paired per-question outcomes) for every model with a capable judge; the 3B local model is the lone borderline case ($p = 0.053$), for a reason the factorial in §5.2 makes precise.

5.2 The effect is inverted-U in generator capability, and the judge is why

Within the controlled GPT-5.4 family (same lab, same generation, capability the only variable), the *relative* reduction is not monotonic: it peaks at the mid-tier model (nano −41%, mini −69%, full −17%). Two forces explain the shape. At the strong end, GPT-5.4-full already has the lowest baseline CW (0.200) and abstains well on its own, so there is less left to catch. At the weak end, the 3B model's *judge* is too weak to catch what its generator produces. The mid-tier is the sweet spot: strong enough to judge reliably, still error-prone enough to need judging.

The generator–judge factorial isolates that second force directly. Table 2 crosses two generators with two judges, holding prompts and checks constant.

Table 2 — verified confidently-wrong rate by generator × judge (mean over 3 runs)

	3B judge	GPT-5.4-mini judge
3B generator	0.109	**0.022**
GPT-5.4-mini generator	0.094	0.067

Reading Table 2 across a row (fix the generator, upgrade only the judge) lowers CW in both rows: sharply for the 3B generator (0.109 to 0.022, a 5× reduction) and clearly for the mini generator (0.094 to 0.067). Reading down a column (fix the judge, upgrade the generator) moves it far less. The judge's main effect exceeds the generator's: swapping only the judge on the *same* weak generator recovers most of the benefit that the 3B self-judge (§5.1, the $p = 0.053$ case) could not. Verification quality is largely a property of the judge, not the generator.

This has a direct deployment consequence: a cheap or on-premise generator paired with a small budget of strong-judge calls approaches the trustworthiness of a frontier model, at a fraction of the cost and without sending the generation workload off-premise.

5.3 Cost of trust

Verification adds one judge call per answered question plus occasional repair retries. On GPT-4o-mini this is 1.58× input+output tokens and *negative* latency overhead in practice (3.9 s to 3.4 s per question), because the judge call is offset by the repair loop replacing failed-execution retries. On the local 3B model the overhead is 1.45× tokens and 1.39× wall-clock. In absolute terms the judge is one extra call against the cost of a single wrong number entering a decision.

5.4 Ablation: which check does the work

Across all self-judge verified runs, we counted which check family caused each repair or abstention:

Table 3 — verification firings by check family (self-judge runs, all models)

check family	times fired	catches
`static.identifiers`	161	invented columns/tables, before execution
`static.select_only`	158	non-SELECT / malformed generations
`judge.sql_answers_question`	21	silent wrong-question SQL that executes fine
`sanity.not_all_null`	9	all-NULL results presented as answers
`sanity.nonempty`	1	empty results

The cheap static checks do the bulk of the filtering: most bad generations reference a non-existent identifier or aren't a clean SELECT, and are rejected for free before any model call. But the 21 LLM-judge firings are the ones the static and sanity checks *cannot* produce: SQL that is syntactically valid, references only real columns, returns a plausible non-empty result, and still answers the wrong question. That residual class is precisely the confidently-wrong failure the whole system exists to catch, and only the judge reaches it, which is also why judge capability (§5.2) dominates the outcome.

6. Failure-mode taxonomy

Classifying the baseline confidently-wrong answers (`notebooks/error_analysis.py`) by golden-set tier shows where they concentrate:

Table 4 — baseline confidently-wrong answers by tier (per run, across models)

tier	share of confidently-wrong answers	dominant failure class
ambiguous	highest	unresolved ambiguity
multi-join	high	silent wrong-question SQL
temporal	moderate	wrong date grain / arithmetic
aggregation, lookup	low	occasional wrong metric
unanswerable	rare	proxy fabrication

The five error classes:

– **(a) Silent wrong-question SQL.** Executes cleanly, returns a plausible number, answers a *different* question. Concentrated in the multi-join tier: the model joins the wrong table, drops a `HAVING` threshold, or aggregates at the wrong grain (for example average payment *per item* when *per order* was asked). Caught only by the LLM judge (§5.4).

– **(b) Invented schema.** References a column or table that does not exist. Caught cheaply and pre-execution by `static.identifiers`; the single most frequent firing.
– **(c) Proxy fabrication.** On an unanswerable question, silently substitutes a lookalike metric (cancellation rate for return rate) with no flag. Rare but the most dangerous, since the answer looks authoritative.
– **(d) Unresolved ambiguity.** The largest baseline cluster: on a vague question ("how is the business trending?") the model commits to one reading, which the strict comparison scores wrong. Verification helps only partially here (see §7 on abstention).
– **(e) Degenerate results.** Empty or all-NULL output presented as an answer. Caught by the sanity checks; uncommon on this schema.

7. Discussion and limitations

Two kinds of abstention. The confidently-wrong rate must be read alongside abstention behavior, because they trade off. The 3B generator attains the lowest verified CW in the factorial (0.022, Table 2) partly because it is *timid*: it abstains often, so it makes fewer confident claims to be wrong about. This is a genuine confound, since CW rate rewards a model that refuses more. We therefore distinguish *justified abstention* (the data is truly absent, for example profit margin with no cost column) from *lazy abstention* (the question is answerable but vague, and the model declines, sometimes with a *fabricated* justification such as "the schema has no time-series data" when it plainly does). The first is the system working; the second is itself a failure mode, an abstention with a hallucinated reason, and belongs in the taxonomy. A verification layer optimized purely for CW can drift toward over-abstention; reporting abstention rate alongside CW (as we do in the harness) keeps that visible.

The judge is a gate, not a proof. Verification lowers but does not eliminate confidently-wrong answers: the residual silent-wrong-question SQL that survives is exactly what the judge occasionally misses. On the unanswerable tier, the strong-judge configurations abstained correctly on all but a handful, but a shared-provider judge inherits the generator's blind spots, mitigated but not removed by using an independent (and stronger) judge, as §5.2 shows.

Limitations. Single schema and single language pair (Portuguese-origin data queried in English); a 150-question golden set is small relative to benchmark suites, though it deliberately includes the ambiguous and unanswerable tiers those suites omit; ambiguous-question scoring accepts one modal interpretation and so under-credits defensible alternative readings; and results are three runs per configuration, adequate for the paired significance tests reported but not a tight variance estimate. We lost the intended open-weights-at-scale data point (gpt-oss-120B) to free-tier quota limits; adding it and a frontier third provider (Claude) is straightforward future work, as the harness is provider-agnostic.

8. Conclusion

Trust, not generation, is the bottleneck for LLM analytics agents. A small, model-agnostic verification layer of cheap static checks, one judge call, a bounded repair loop, and the option to abstain significantly reduces confidently-wrong answers across five generators spanning a 3B local model to a frontier API, at a token cost easily justified against the cost of one wrong number entering a decision. The generator–judge factorial locates the effect precisely: verification quality is largely a property of the *judge*, not the generator, so a cheap or on-premise generator under a thin layer of strong-judge calls can approach frontier-level trustworthiness. The failure that matters in interactive analytics is not the query the system cannot write. It is the wrong answer it presents as right, and that failure is measurable, reducible, and best addressed at the point of verification.

References

[1] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proc. EMNLP*, 2018, pp. 3911–3921.

[2] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li, "Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs," in *Proc. NeurIPS Datasets and Benchmarks Track*, 2023.

[3] R. Zhong, T. Yu, and D. Klein, "Semantic evaluation for text-to-SQL with distilled test suites," in *Proc. EMNLP*, 2020, pp. 396–411.

[4] N. Rajkumar, R. Li, and D. Bahdanau, "Evaluating the text-to-SQL capabilities of large language models," arXiv:2204.00498, 2022.

[5] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction," in *Proc. NeurIPS*, 2023.

[6] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-SQL empowered by large language models: A benchmark evaluation," *Proc. VLDB Endowment*, vol. 17, no. 5, pp. 1132–1145, 2024.

[7] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "CHESS: Contextual harnessing for efficient SQL synthesis," arXiv:2405.16755, 2024.

[8] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L.-W. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun, and Z. Li, "MAC-SQL: A multi-agent collaborative framework for text-to-SQL," in *Proc. COLING*, 2025, pp. 540–557.

[9] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, "Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows," in *Proc. ICLR*, 2025.

[10] G. Lee, H. Hwang, S. Bae, Y. Kwon, W. Shin, S. Yang, M. Seo, J.-Y. Kim, and E. Choi, "EHRSQL: A practical text-to-SQL benchmark for electronic health records," in *Proc. NeurIPS Datasets and Benchmarks Track*, 2022.

[11] G. Lee, W. Chay, S. Cho, and E. Choi, "TrustSQL: Benchmarking text-to-SQL reliability with penalty-based scoring," arXiv:2403.15879, 2024.

[12] S. Feng, W. Shi, Y. Wang, W. Ding, V. Balachandran, and Y. Tsvetkov, "Don't hallucinate, abstain: Identifying LLM knowledge gaps via multi-LLM collaboration," in *Proc. ACL*, 2024, pp. 14664–14690.

[13] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[14] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Trans. Information Systems*, 2025 (arXiv:2311.05232, 2023).

[15] P. Manakul, A. Liusie, and M. J. F. Gales, "SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models," in *Proc. EMNLP*, 2023, pp. 9004–9017.

[16] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *Proc. ICLR*, 2023.

[17] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and

P. Clark, "Self-Refine: Iterative refinement with self-feedback," in *Proc. NeurIPS*, 2023.

[18] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Proc. NeurIPS*, 2023.

[19] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen, "CRITIC: Large language models can self-correct with tool-interactive critiquing," in *Proc. ICLR*, 2024.

[20] S. Kadavath, T. Conerly, A. Askell, T. Henighan, D. Drain, E. Perez, N. Schiefer, Z. Hatfield-Dodds, N. DasSarma, E. Tran-Johnson, et al., "Language models (mostly) know what they know," arXiv:2207.05221, 2022.

[21] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena," in *Proc. NeurIPS Datasets and Benchmarks Track*, 2023.

[22] J. Gu, X. Jiang, Z. Shi, H. Tan, X. Zhai, C. Xu, W. Li, Y. Shen, S. Ma, H. Liu, S. Wang, K. Zhang, Y. Wang, W. Gao, L. Ni, and J. Guo, "A survey on LLM-as-a-judge," arXiv:2411.15594, 2024.

[23] A. Kamath, R. Jia, and P. Liang, "Selective question answering under domain shift," in *Proc. ACL*, 2020, pp. 5684–5696.

[24] J. R. Cole, M. J. Q. Zhang, D. Gillick, J. M. Eisenschlos, B. Dhingra, and J. Eisenstein, "Selectively answering ambiguous questions," in *Proc. EMNLP*, 2023, pp. 530–543.

[25] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proc. ICLR*, 2023.

[26] Olist, "Brazilian e-commerce public dataset by Olist," Kaggle, 2018. [Online]. Available: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce