Preserving LLM Output Distributions for Probabilistic Query Evaluation

Angelo Saadeh
angelo.saadeh@ens.fr
DI ENS, ENS, CNRS, PSL University, Inria
Paris, France

Pierre Senellart
pierre@senellart.com
DI ENS, ENS, CNRS, PSL University, Inria
Paris, France

Abstract

Language models can be used to augment databases; typically, when this is done, their top prediction is used while the uncertainty in this prediction is ignored. We propose a framework that preserves output distributions from language model predictions and uses them to evaluate SQL queries probabilistically. First, a language model is used to infer new features from existing ones. Second, ProvSQL, a PostgreSQL extension implementing a probabilistic database through data provenance, encodes the predictions as mutually exclusive events in a provenance circuit and evaluates probabilistic queries. We extend ProvSQL to support text-value comparisons and join operations on language model predictions. We demonstrate the framework on 10 612 Amazon musical products and 230 156 user reviews, locally augmented with the Llama 3.1 model. We show that probabilistic querying reveals contradictions between model predictions and existing labels, recovers data records otherwise missed, and propagates joint uncertainty across multi-table queries.

CCS Concepts

• **Information systems** → **Data management systems**; • **Computing methodologies** → **Natural language processing**; • **Theory of computation** → **Data provenance**; **Probabilistic computation**.

Keywords

Probabilistic database, Data provenance, Data uncertainty, Data augmentation, LLM predictions

ACM Reference Format:
Angelo Saadeh and Pierre Senellart. 2026. Preserving LLM Output Distributions for Probabilistic Query Evaluation. In *Proceedings of the 35th ACM International Conference on Information and Knowledge Management (CIKM '26), November 07–11, 2026, Rome, Italy*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3799682.3840264

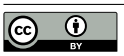
1 Introduction

Incomplete, unstructured or uncertain data hinders good data analysis and hence data-based decision-making processes. Depending on the considered application and problem, different solutions may be used [1, 4]. One solution in particular uses a language model to handle missing features by augmenting the database [6]. When this is done, the top prediction from the model is used. Oftentimes models are not confident about their predictions, and by ignoring the other predictions of the model, uncertainty is ignored as well. In our proposed framework, we preserve the output distributions of models and use them for probabilistic SQL query evaluation.

First, a language model is used to deal with missing values, missing features, and unstructured data. A new feature is inferred given other features or general knowledge, and a prompt is engineered accordingly. We explore the most likely outputs of the language model in a beam-search manner and get their log-probabilities (log-probs), which are used to generate a mapping from values to probabilities for each data record. Second, since language models tend to be uncertain, we pass on their uncertainty to the database allowing a probabilistic evaluation of queries. To do this, we use ProvSQL [9, 11], a PostgreSQL extension for implementing a probabilistic database through data provenance. ProvSQL works by adding a new column called `provsql` to tables, containing universally unique identifiers (UUIDs) identifying gates in a provenance circuit. At initialization, tables contain one fresh UUID per data record that corresponds to an input gate of the circuit; then, for each SQL operation, ProvSQL generates new provenance tokens for the corresponding operation; these are the inner gates of the circuit. Queries are rewritten to produce a valid provenance annotation in the `provsql` column. At probability evaluation time, gates contributing to the final data record are used to compute the probability of that data record. We extend the ProvSQL library with two new operations: the first one compares the probability mapping from the model with text values, and the second one joins tables on the probability mapping.

We demonstrate the proposed framework on two datasets from the "Amazon Reviews 2023" [8] about musical products and user reviews. Both datasets are augmented locally using the language model Llama 3.1 [13] with eight billion parameters. In the demonstration, we add features to the table by categorizing musical products by type (accessory vs. instrument) and by instrument, and categorizing user reviews by sentiment (positive, neutral, or negative). These new features are cross-referenced with reference data about musical instruments for further insight. We run multiple queries and compare the result of our probabilistic query evaluations to deterministic evaluations (that only take into account the output of the language model with the highest probability).

We provide a short video[1] illustrating the demonstration scenario, while the code and datasets are available on GitHub[2].



CIKM '26, Rome, Italy

ACM ISBN 979-8-4007-2539-5/2026/11
https://doi.org/10.1145/3799682.3840264

[1] Video: https://youtu.be/VWvf6BEABKM
[2] GitHub: https://github.com/angelosaadeh/musical-demo

2 Architecture

Data Augmentation. We consider a language model $\mathcal{M}$, and a dataset with $d-1$ features $A_1, \ldots, A_{d-1}$; the goal is to add a new feature A_d to the dataset using $\mathcal{M}$. To do this, given the task (new feature to add), we engineer a prompt P that uses the data record and other features for this data record with few-shot examples to help the language model. The prompt P, a number of tokens, and a number k for choices are sent to a server that hosts the language model $\mathcal{M}$; the server first runs $\mathcal{M}(P)$ for one token and gets the top k answer tokens $a_1, \ldots, a_k$. Answers with very low log-probs compared to the highest log-prob are eliminated and only answers a_i for $i \in I \subseteq \{1, \ldots, k\}$ are kept; each of the remaining answers is appended to P and the server runs $\mathcal{M}(P + a_i)$ for every $i \in I$. This step is repeated until the desired number of tokens is reached or a pre-defined stopping point is attained. This method allows the predictions to evolve in different directions while limiting choices, in a beam-search style. At the end, the server returns, in JSONB format, a mapping of values and probabilities, the latter computed by multiplying the probabilities at each step. The model calls could be done live in SQL via a user-defined function (UDF) or they could be pre-computed and stored in the table. In our demonstration, we pre-computed the language model outputs and stored them in the table to avoid latency issues, but the framework supports both options.

Aggregation of the Model Outputs. We consider a table with a column of type JSONB that contains for a given record a probability distribution encoded as a mapping from a finite set of values to probabilities: $\{v_1 : p_1, \ldots, v_n : p_n\}$, where each record could have a different number of values and these values are sorted by probability from highest to lowest: $p_1 \geq p_2 \geq \cdots \geq p_n$. We use ProvSQL to iteratively create a gate in the provenance circuit for each value $v_1, \ldots, v_n$ and assign probabilities to these gates so that values are encoded as mutually exclusive events, enabling correct probabilistic query evaluations. Indeed, two values v_i and v_j cannot be correct at the same time; therefore, we transform the raw probabilities $p_1, \ldots, p_n$ into conditional probabilities, and we create a circuit for the values using monus and plus gates, representing **AND NOT** and **OR** logical operations, respectively.

Given a record with values $\{v_1 : p_1, \ldots, v_n : p_n\}$, first we create n input gates (fresh UUIDs) $t_1, \ldots, t_n$ with conditional probabilities $f(p_1), \ldots, f(p_n)$. Second, we create a gate g_1 equal to t_1 and assign v_1 to it, a gate g_2 equal to $t_2 \ominus t_1$ and assign v_2 to it, a gate g_3 equal to $t_3 \ominus (t_1 \oplus t_2)$ and assign v_3 to it, and so on. The symbols $\ominus$ and $\oplus$ denote monus and plus gates, respectively; they are implemented in ProvSQL. These are inner gates obtained by performing operations on other gates: a circuit is being created. Generally, for $i > 1$, we create a gate g_i equal to $t_i \ominus \left(\bigoplus_{j=1}^{i-1} t_j\right)$. This ensures that all the gates (and hence values) are mutually exclusive; furthermore, if we define the function f as:

$$f(p_i) = \begin{cases} p_i & \text{if } i = 1 \\ \dfrac{p_i}{\prod_{j=1}^{i-1}(1 - f(p_j))} & \text{if } i > 1, \end{cases}$$

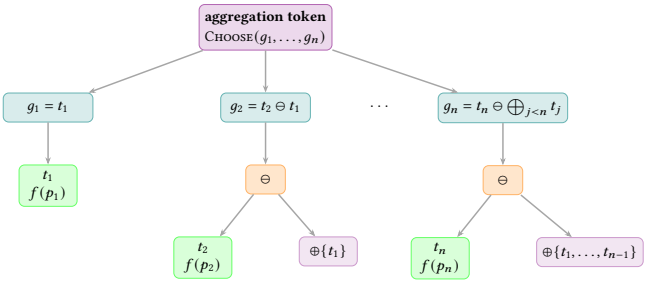


Figure 1: Provenance circuit built by BuildProvenance for a data record with mapping $\{v_1{:}p_1, \ldots, v_n{:}p_n\}$. Input gates t_i carry conditional probabilities $f(p_i)$.

then we obtain probabilities $p_1, \ldots, p_n$ for the gates $g_1, \ldots, g_n$, respectively. This process is achieved via a function called BuildProvenance that takes a column of type JSONB as input, and returns an aggregation token as an output. When evaluated, BuildProvenance creates all the required gates $g_1, \ldots, g_n$ then aggregates them using a function called Choose. The Choose function picks the first valid value it encounters, and since the values were sorted by probability, the gates have inherited this order. The aggregation token itself is also a gate obtained by performing an operation on $g_1, \ldots, g_n$. We illustrate the circuit for one data record in Figure 1.

Operations on Aggregation Tokens. The function BuildProvenance applied to a column of type JSONB results in a column of type aggregation token. While the ProvSQL extension supports some operations on aggregation tokens, we extended the library to support further operations, as explained in the following paragraphs. All changes made to ProvSQL for this demonstration are released as part of the latest version (1.9.0) of the extension.

Compare aggregation tokens with text values. The ProvSQL extension supports comparing values of type aggregation token to numerical values (operators $=, \neq, \geq, \leq, <, >$). We extended the library to support comparisons with text values (operators $=, \neq$) in the specific case where the aggregation token is the result of the aggregation function Choose applied on mutually exclusive gates, in which case we only need to make a linear scan of the aggregated values; this may sound trivial, but it does not hold for other aggregation functions. Indeed, aggregation functions like sum, mean, min, max, etc., are usually applied to numerical values, and comparing them to text values is not meaningful. Moreover, when applying comparison operators on such aggregation results, a linear scan of the aggregated elements is not enough; we must explore all the possible combinations in which values could be present. Furthermore, ProvSQL also handles logical operations like **`OR`**, **`AND`**, and **`NOT`** in **`WHERE`** and **`HAVING`** clauses, which are crucial for our framework.

Join tables on aggregation tokens. ProvSQL supports join operations between two tables when joining two text or numerical columns. We extended the library to support joining two tables where the first table contains an aggregation token and the second table contains a numerical or text column. We do this by performing an operation called *exploding the aggregation token.* Consider a

data record $(x_1, \ldots, x_d, \tau)$ where x_d is an aggregation token, and τ is the UUID in an extra column called `provsql` that contains the provenance of that particular data record. First, we use ProvSQL functionalities called GetChildren and GetExtra to walk the circuit and retrieve the information from the mapping that gave us x_d: the values of the gates $g_1, \ldots, g_n$ and their probabilities $p_1, \ldots, p_n$. Second, we replicate each row n times to obtain n new data records $(x_1, x_2, \ldots, x_{d-1}, v_i, \tau \otimes g_i)$, for $1 \leq i \leq n$, where $\otimes$ is a times gate. This means the first $d-1$ columns remain the same for all the new n records, the d-th column that used to contain the aggregation token now contains the value v_i and the `provsql` column now contains a new UUID equal to $\tau \otimes g_i$. Since the probability of g_i is p_i and the probability of τ is 1 by default, the probability of record i is p_i. Third, the column that used to contain the aggregation token is now a text or numerical value, depending on the type of $v_1, \ldots, v_n$, so it can be used to join on another table: a classic join handled by existing functionalities in ProvSQL.

3 Demonstration Scenario

To demonstrate our framework, we use a database containing three datasets: two datasets from Amazon [8], the first one being a musical products dataset and the second one a user reviews dataset of the products; the third one is a reference table containing musical instruments with their families and types. We first summarize the cleaning and augmentation process of the data, then show examples of how language model augmentation and probabilistic evaluation of queries produce new insights about the data, and finally illustrate the user interface from which queries can be executed.

Datasets. The musical instrument datasets from Amazon originally contain 213 thousand products and 3 million user reviews. First, we filter the products dataset to keep only the products whose `asin` (identifier), `title`, `description`, `features`, `price` and `average_rating` are non-empty. Second, we only keep the products that have between 10 and 50 user reviews, to keep the number of reviews per product comparable. We are left with two datasets `meta` and `reviews` containing 10 612 products and 230 156 reviews respectively. The `reviews` table contains the ASIN of the product being reviewed, the text review, and the user rating between 1 and 5. We augment both tables. From the existing features of the `meta` table, we derive two new columns: (1) `is_instrument` with values "yes" or "no" to know if a product is an instrument or an accessory; it is obtained from the server (Section 2) by allowing only 1 token from the language model, and (2) `instrument` to know which musical instrument a product pertains to; it is obtained with 3 tokens. Both columns use few-shot examples in the prompts and use the `title`, `description`, and `features` of the products in the prompt. The `reviews` table is augmented with a column called `sentiment`, which uses 1 token to obtain three answers: "positive", "neutral" or "negative". The prompt only uses the text review; it does not use the user's rating. All inferences were run locally on an Apple M4 laptop (10-core CPU, 16 GB unified memory) using the model Llama-3.1-8B-Instruct [13] with llama.cpp [7] and Metal GPU acceleration. The reference table `instruments` contains four columns `name`, `type`, `subtype`, `family`; it was generated by a language model and is used to derive further information about the products.

New Insights from Probabilistic Querying. We demonstrate that combining language model predictions with probabilistic queries produces new insights about the data by running example queries.

Improve data quality. We notice that some reviews are labeled positive by the language model but give a low star rating to the product, and conversely. These are not ironic reviews misclassified by the model. They are data quality problems in the `rating` column, so using the `sentiment` attribute in queries produces more correct data analysis results. An example query and its top three results are illustrated in Figure 2.

```
SELECT asin, rating, review, sentiment, probability
FROM reviews
WHERE sentiment = 'positive' AND rating = 1
ORDER BY probability DESC;
```

```
asin       | rating | review                          | sentiment | probability
-----------+--------+---------------------------------+-----------+-------------
B082SKRWCT |      1 | Great product, just what we ... | positive  | 0.99965
B08K2RKS6X |      1 | I love this little device! S... | positive  | 0.99961
B078Y4DWGV |      1 | Wow - this is a great, great... | positive  | 0.99937
```

Figure 2: Top three reviews confidently labeled as positive by the language model but with a 1-star rating.

The data also reveals information on user behavior: we counted such contradictory ratings and found that negative reviews with good ratings (4 or 5 stars) are about 6.8 times more frequent than positive reviews with bad ratings (1 or 2 stars). This could be interpreted in multiple ways and could help orient data analysis tasks to better understand user behavior.

Recover borderline predictions of the language model. While the language model improves data quality and offers information about behavioral patterns otherwise invisible without data augmentation, its outputs may be uncertain. Therefore, it is crucial to complement the data augmentation with a probabilistic evaluation of the queries. We demonstrate that running queries in a deterministic manner (by picking the top choice of a language model) may cause incorrect results since the model is not always confident.

To demonstrate this, we create a table from the `meta` dataset where the column `is_instrument` is deterministic and contains values "yes" or "no" depending on which one has a higher log-prob by the language model; we call this new table `meta_det`. Joining `meta` and `meta_det` on the identification number `asin`, and filtering records classified as non-instruments in `meta_det`, but as instruments with substantial probability in `meta`, shows that many products with uncertain outputs are wrongly classified as non-instruments in the deterministic table, as shown in Figure 3.

For instance, the second product returned is classified as a non-instrument despite a 0.4648 probability of being an instrument. Many such errors appear in the top results: `meta_det`, built from the model's top predictions, gives incorrect results, whereas `meta` preserves the model's uncertainty through probabilistic evaluation. Mistakes from deterministic tables occur when the language model gives predictions with close log-probs; choosing one output over another in this case does not correctly reflect the data.

```
SELECT m.asin, m.title, probability
FROM meta AS m
JOIN meta_det AS md ON m.asin = md.asin
WHERE m.is_instrument = 'yes' AND md.is_instrument = 'no'
ORDER BY probability DESC;
```

```
asin       | title                                              | probability
-----------+----------------------------------------------------+-------------
B0772SPTK1 | Seymour Duncan Studio Bass Compressor Pedal         | 0.4708
B000VTKLX6 | Latin Percussion LP304LP Pro Merengue Guiro         | 0.4648
B07DMKYJ6H | Guitar Hook Auto Lock Guitar Hanger                 | 0.4619
```

Figure 3: Top three products classified as non-instruments despite a probability close to 0.5 of being an instrument.

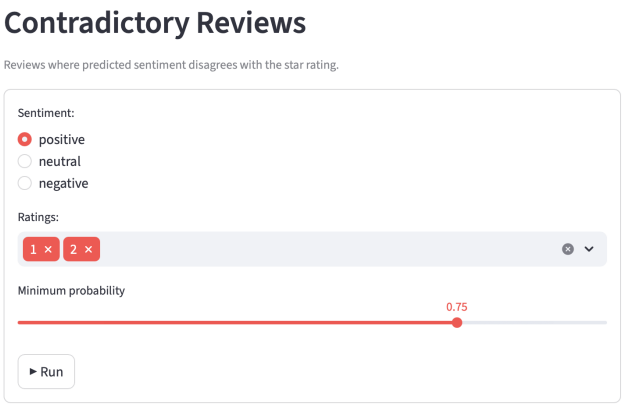


Figure 4: One of the tabs of the user interface, used to run queries on the augmented `reviews` table.

Control compounded uncertainty. The power of ProvSQL lies in being able to run multiple successive queries that may depend on each other and still be able to evaluate the probability of a result at any time. We write such queries where all three tables `meta`, `reviews` and `instruments` are joined and filtered with **`WHERE`** and **`HAVING`** conditions. For example, we can filter the products that are wind instruments with "positive" reviews.

When a query is evaluated probabilistically we obtain a joint probability of all filtering conditions to which deterministic evaluations are blind. Our framework offers the option to adjust the probability threshold of a query allowing a new dimension of control for better data analysis tasks.

User Interface. Our user interface contains multiple tabs, each of which allows a user to run a specific kind of query, like filtering reviews by sentiment and ratings, or browsing the products table by instrument type. We also made a query builder tab where users can compose and run custom SQL queries without writing SQL code; the generated SQL can still be edited before execution. The results of the queries can be filtered by a probability threshold. We show in Figure 4 the tab in the user interface that helps uncover contradicting reviews as shown in Figure 2. A user can choose a sentiment, a star rating and a probability threshold to filter the `reviews` table. In Figure 5, the results of the query are shown alongside the execution time and the probability method used to

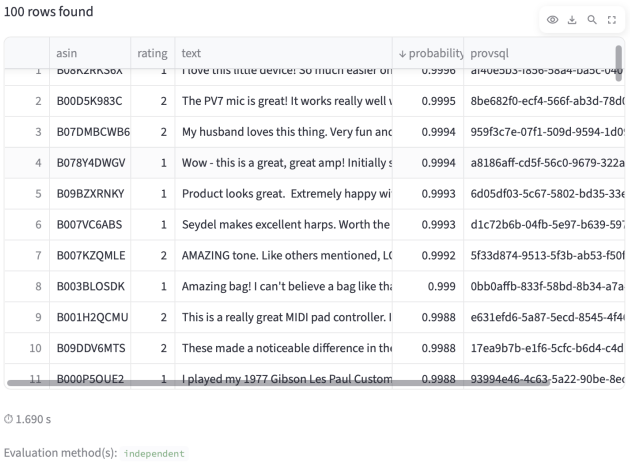


Figure 5: Results of the query illustrated in Figure 4.

evaluate the query. Users also have access to the query rewriting: they can see how the query is originally written, and how ProvSQL rewrites it to make all operations semiring-compatible. In addition, the equivalent query can be run in parallel on the deterministic table to check for differences. Finally, we will complement the demo interface with ProvSQL Studio [10], the user interface we develop for ProvSQL, which allows running queries on test cases and visualizing provenance circuits.

4 Conclusion

In this paper, we presented a framework combining data augmentation from a language model and probabilistic query evaluations in ProvSQL. Previous works use outside sources or models to enhance tables with additional knowledge [12]. In [3], language models are used to query datasets by transforming natural language questions like "Which country is each individual from?" to code that augments the dataset with LM calls. A related, more efficient query engine is proposed in [5]; however, neither paper addresses uncertainty. In [14], outputs and model confidence from an object detection model are used to search for images containing certain items in a probabilistic manner. In contrast, our framework allows encoding mappings of mutually exclusive probabilities on multiple tables and performing SQL operations on the whole database, allowing new insight about the data we already have for better data analysis and decision-making. The probability mapping that we use to add values in the augmented tables comes directly from the log-probs of the language models. Log-probs by themselves are not a calibrated representation of correctness, but nothing in the proposed construction depends on them being so; the function BuildProvenance consumes a mapping from values to probabilities, whatever its source. Some works [2] propose calibrations to map log-probs to more accurate probabilities, and others estimate uncertainty by sampling at non-zero temperature. Either could be substituted directly in our framework for better probabilistic querying.

GenAI Usage Disclosure

Claude Code by Anthropic was used to assist in the implementation of the experiments by generating, correcting and improving code (in Python, SQL, and C).

References

[1] Felix Bießmann, David Salinas, Sebastian Schelter, Philipp Schmidt, and Dustin Lange. 2018. "Deep" Learning for Missing Value Imputation in Tables with Non-Numerical Data. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, and Haixun Wang (Eds.). ACM, Torino, Italy, 2017–2025. doi:10.1145/3269206.3272005

[2] Lihu Chen, Alexandre Perez-Lebel, Fabian M. Suchanek, and Gaël Varoquaux. 2024. Reconfidencing LLMs from the Grouping Loss Perspective. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024 (Findings of ACL)*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 1567–1581. doi:10.18653/V1/2024.FINDINGS-EMNLP.85

[3] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding Language Models in Symbolic Languages. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=lH1PV42cbF

[4] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, San Francisco, CA, USA, 2201–2206. doi:10.1145/2882903.2912574

[5] Hanjun Dai, Bethany Wang, Xingchen Wan, Bo Dai, Sherry Yang, Azade Nova, Pengcheng Yin, Phitchaya Mangpo Phothilimthana, Charles Sutton, and Dale Schuurmans. 2024. UQE: A Query Engine for Unstructured Databases. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/34b3a40ec9752c1ae48fe85fef8fe8dc-Abstract-Conference.html

[6] Mohamed Y. Eltabakh, Zan Ahmad Naeem, Mohammad Shahmeer Ahmad, Mourad Ouzzani, and Nan Tang. 2024. RetClean: Retrieval-Based Tabular Data Cleaning Using LLMs and Data Lakes. *Proc. VLDB Endow.* 17, 12 (2024), 4421–4424. doi:10.14778/3685800.3685890

[7] Georgi Gerganov. 2023. *llama.cpp*. https://github.com/ggml-org/llama.cpp

[8] Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian J. McAuley. 2024. Bridging Language and Items for Retrieval and Recommendation. *CoRR* abs/2403.03952 (2024). arXiv:2403.03952 doi:10.48550/ARXIV.2403.03952

[9] Aryak Sen, Silviu Maniu, and Pierre Senellart. 2025. ProvSQL: A General System for Keeping Track of the Provenance and Probability of Data. *CoRR* abs/2504.12058 (2025). arXiv:2504.12058 doi:10.48550/ARXIV.2504.12058

[10] Pierre Senellart. 2026. ProvSQL Studio. https://provsql.org/docs/user/studio.html. Accessed: 2026-06-01.

[11] Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. 2018. ProvSQL: Provenance and Probability Management in PostgreSQL. *Proc. VLDB Endow.* 11, 12 (2018), 2034–2037. doi:10.14778/3229863.3236253

[12] Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2024. TAP4LLM: Table Provider on Sampling, Augmenting, and Packing Semi-structured Data for Large Language Model Reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024 (Findings of ACL)*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 10306–10323. doi:10.18653/V1/2024.FINDINGS-EMNLP.603

[13] Llama Team. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). arXiv:2407.21783 doi:10.48550/ARXIV.2407.21783

[14] Fajrian Yunus, Pratik Karmakar, Pierre Senellart, Talel Abdessalem, and Stéphane Bressan. 2025. Using A Probabilistic Database in an Image Retrieval Application. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, Barcelona, Spain, 1106–1109. doi:10.48786/EDBT.2025.100