A Generative Benchmark Creation Framework for Detecting Common Data Table Versions

Daniel C. Fox
Worcester Polytechnic Institute
Worcester, USA
dcfox1@wpi.edu

Aamod Khatiwada
Northeastern University
Boston, USA
khatiwada.a@northeastern.edu

Roee Shraga
Worcester Polytechnic Institute
Worcester, USA
rshraga@wpi.edu

ABSTRACT

Multiple versions of the same dataset can exist in a data repository (e.g., data warehouses, data lakes, etc.), mainly because of the interactive and collaborative nature of data science. Data creators generally update existing datasets and upload them as new datasets to data repositories without proper documentation. Identifying such versions helps in data management, data governance, and making better decisions using data. However, there is a dearth of benchmarks to develop and evaluate data versioning techniques, which requires a lot of human effort. Thus, this work introduces a novel framework to generate benchmarks for data versioning using Generative AI (specifically Large Language Models). The proposed framework offers properties that existing benchmarks do not have, including proper documentation, version lineage, and complex transformations generated by an LLM. We also share VerLLM-v1, the first version of the benchmark that features these properties, and compare it to existing benchmarks.

CCS CONCEPTS

• **Information systems** → **Deduplication**; *Data cleaning*; Mediators and data integration; *Version management*; *Data extraction and integration*; • **Computing methodologies** → *Natural language generation*.

ACM Reference Format:
Daniel C. Fox, Aamod Khatiwada, and Roee Shraga. 2024. A Generative Benchmark Creation Framework for Detecting Common Data Table Versions. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3627673.3679157

1 INTRODUCTION

Enterprise and government organizations store massive numbers of datasets (tables) in centralized storage such as data lakes [9]. Generally, they upload each table to the data lake as an independent file (e.g., CSV file) [17, 22], typically using superficial documentation, e.g., embedded in filenames. Furthermore, when data in a table needs an update, the necessary amendments are made to the original table that may already exist in the data lake [24]. The amendments could include adding a new row, deleting an existing column, changing a cell value, and more. The amended table is then uploaded to the data lake as a new dataset.

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10.
https://doi.org/10.1145/3627673.3679157

Example 1. *To illustrate the real-world necessity, consider Tables (a) and (b) in Fig. 1 that are snippets of real Open Data Tables [10]. The tables show the budgets allocated for different areas in Fiscal Years (a) 2013 and (b) 2015. The table for 2015 is clearly generated by amending the table for 2013 (i.e., a version of it). Yet, these tables are not labeled as such. For illustration, we highlight the changes between the tables in each row with the same color. Note, again, that these changes are not marked in a typical case. The first two rows have the updates in the amounts allocated for the same functions. The third row of Table (b) have different* `Program`, `Fund Name`, *and* `Amount` *than Table (a). The fourth row of Table (a) about Libraries, Culture, and Recreation is replaced with General Government* `Function` *in Table (b). The last rows of the tables have different programs (*`Programs Name`*).*

Function	Department	Program Name	Fund Name	Amount
Community Development and Housing	Housing and Community Affairs	Multi-Family Housing	Montgomery Housing In.	1114441
Environment	Environmental Protection	Watershed Management	Water Quality Protection	5030
Health and Human Services	Health and Human Services	Community Action Agency	Grants	5169
Libraries, Culture, and Recreation	Public Libraries	Public Library Services	General Fund	2430
Public Safety	Police	Field Services	General Fund	193146

(a) Fiscal Year 2013 Budget

Function	Department	Program Name	Fund Name	Amount
Community Development and Housing	Housing and Community Affairs	Multi-Family Housing	Montgomery Housing In.	1757420
Environment	Environmental Protection	Watershed Management	Water Quality Protection	130020
Health and Human Services	Health and Human Services	Assessment & Case Mgmt.	Elder Affairs Grants	1796
General Government	Technology Services	County Cable Montgomery	Cable TV	13956
Public Safety	Police	Patrol Services	General Fund	85880

(b) Fiscal Year 2015 Budget

Figure 1: Tables (a) and (b) are Open Data Table snippets that are versions of each other and show budgets allocated in 2013 and 2015, respectively. Cells with different values between the two tables are highlighted using the same colors.

Identifying the versions of data lake tables could help in efficient data management, effective data analysis, and proper decision-making, featuring research questions such as *Given a pair of tables, are they versions of one another or are both of them versions of an original (root) table present in the data lake?* Note that a newer version of a table may be a complex transformation over the original table, or even over another table that is a version of the original table. For instance, continuing from Ex. 1, one may update the rows and columns of the table for Fiscal Year 2015 (Table (b)) and create a new dataset for Fiscal Year 2016, which would be a two-hop transformed version of Table (b) for Fiscal Year 2013. However, the version detection techniques have been developed and evaluated over manually created benchmarks by considering one-hop versions and only for simpler transformations such as detecting normalized columns, arithmetic transformation over a column, and so on [16, 24]. This is mainly because it is not straightforward and very labor-intensive to manually create data versioning benchmarks with complex multi-hop transformations.

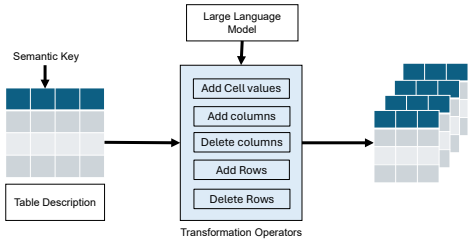

Figure 2: Data Versioning Benchmark Creation Framework

Recently, Large Language Models (LLMs) have been showing promising results in annotation and semantic tasks such as column type detection [14], entity matching [21], benchmarking table union search [19], and more [4, 7]. Therefore, in this work, we propose a novel LLM-based method for creating data versioning datasets that can be used to develop and evaluate the data versioning techniques. We consider five different transformations that cover cell-level, column-level, and row-level differences between the table versions.

We summarize our contributions on the following points.

- We present a novel LLM-based data versioning benchmark generation method that considers five transformations over the original table to create its newer versions.
- We create a novel data versioning benchmark that covers multi-hop transformation over the original table to better represent real data lake scenarios.
- We open-source our code and benchmarks: https://github.com/danielcfox/genbenchver.

Related Work. Prior work creates datasets for other tasks by changing contents of an existing table [13, 18]. For instance, Park et. al. [20] developed a method for synthesizing a table based on the content of an existing table to anonymize data in an existing table using CNN-based Generative Adversarial Networks (*table-GAN*). Xu and Veeramachaneni [25] also developed such a method using an RNN-based architecture. Zhang et al. [27] developed a method to predict cell values of tables. They pre-trained an auto-regressive Language Model (LM) on a corpus drawn from a public tabular dataset and fine-tuned it for their downstreaming tasks. Furthermore, Pal et al. [19] created a benchmark for unionable table search task [13, 18] using an LLM. Specifically, they prompted the model to generate a pair of unionable and non-unionable tables which they post-processed to obtain query and data lake tables for the search task. All these works, however, create benchmarks that are not for data version detection in data lakes, which is our focus. The closest work to ours is Explain-Da-V framework [24], which takes two different versions of the same table and explains the data transformations that are necessary to migrate from one table to the other. However, they employ manual transformations to create their dataset which limits their scalability. Also, they consider single hop transformations whereas our framework, powered by an LLM [12], enables multi-hop transformations to better represent the table versioning scenario in the data lakes. Earlier work on data versioning emphasizes its management [5, 6, 23] and thus does not generate or provide publicly available benchmarks.

2 METHODOLOGY

Now we explain our framework to create datasets for the data versioning task. Fig. 2 shows a high-level architecture of our system. The input is a table, which we call a root (seed) table, along with some description. Furthermore, we allow users to select the column(s) within the table to guide the version creation process, which we call a *semantic key*. We then apply a number of transformations, which the user can select over the input table and create its versions, also accounting for multi-hop transformations. We first explain the transformations that we use in our system. Then, we discuss how we use them to create multi-hop versions of the seed tables.

2.1 Transformations

We propose different transformations over the seed tables on cell level, column level, and row level to account for the possible transformations that could be applied in the data lake tables.

Update Cell Values: As discussed in Ex. 1, a new table can be created by updating the cell values of an existing table. Such transformation is accounted by the *Add Cell Values* component in our system. For the given seed table, this component randomly replaces its values with LLM-generated values. We prompt the seed table to the LLM, along with its description and semantic key, as follows:

> *For {description}, retrieve a missing value of real data (not fictional) from externally available resources, corresponding to the first row and attribute named {attribute}, with a dtype of {col_dtype}, within the following table: {table}*
> *Retrieve the attribute value according to the values of the semantic key: {key[i]} = {semantic_values[i]}*
> *Fill in the missing data, and output the resulting table in {delimiter}-delimited .csv format. Then output from where the data was retrieved.*

Add Columns: Another important transformation applied in data versioning is the addition of new columns to the existing table. In a new version based on column addition, the added columns are generally related to the existing columns and their semantics. So, to cover this, we present the *Add Columns* component to our benchmark generator. This component inputs column headers, a table description, and the semantic key. The column headers are ones either currently present or were present within the table's lineage. The latter are added to ensure that duplicate tables are not created within a benchmark. Then, based on the table semantics captured by its semantic key(s), it uses the following LLM prompt to generate additional column(s) that we add to the seed table:

> *Generate {ncols} new attributes for a table about {description}. The {delimiter}-separated header of attributes to not generate is: {header}*
> *Generate real attributes. Do not generate fictional ones. Here are the {delimiter}-separated rows of the table by semantic key only: {table_key_only}.*
> *Generate values of real data for all existing rows of the table. Generate and output a new table, include the table header, with only the attributes {[keys[i]]} and the new attributes in the format of a {delimiter}-separated .csv file. Then explain the source of the new data.*

Delete Columns: A new version of the table can also be generated by deleting certain columns from the seed table. To capture this, we

include the *Delete Columns* component, which randomly deletes some columns to create a new table. As this component does not need semantic modification, it does not involve an LLM.

Add Rows: Next, we include an *Add Rows* transformation to the system that adds rows to the new version that are not already present in the seed table. We use an LLM to generate such rows. For this, it is important to understand what the table and its columns are. So, we input the seed table and the set of its column headers. Moreover, we do not want to generate the rows that are now or were previously present in the table's lineage (the latter to prevent duplicated tables included in the benchmark). And since the token limit can be an issue to input all the rows, we specify values from only the semantic key column(s) of the seed table in the prompt, which we input into the LLM to generate additional rows that can extend the seed table vertically.

Generate {nrows} new rows for a table of {description}. The {delimiter}-separated headers of attributes for the table are: {header} Generate the rows from real known data. Here is a list of {delimiter}-separated rows not to be generated by semantic key only: {table_inelligible_rows}. Output the rows in the format of a {delimiter}-separated .csv file with a column header. Then explain the source of the new data.

Delete Rows: Next, we include the *Delete Rows* component in our benchmark generator, which randomly deletes a given number of existing rows from the seed table to generate a new version.

2.2 Version Generation

To generate versions of a table, we input the table, its description, semantic key, and the number of versions to be generated.

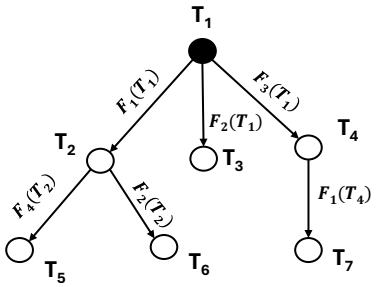

Figure 3: Illustration of applying different transformations over the seed table.

Fig. 3 shows the process of creating new versions from the seed table. Specifically, the benchmark generator randomly applies different transformations(see Sec. 2.1) over the input table to create a version. Furthermore, to create multi-hop transformations, the system also selects a newly generated table and applies a transformation over it. The output of this step will generate the set of tables, along with a ground truth for each of them, which shows their parent tables and the applied transformations.

Example 2. *In Fig. 3, we show an example of generating the versions of seed Table T_1. First, we apply transformation F_1 over T_1. Then to ensure multi-hop transformation, we apply F_4 and F_2 over Table T_2 to generate Tables T_5 and T_6 respectively. As Table T_6 has Table T_2 as a parent, Table T_6 is related with T_1 by two hops.*

As presented in Fig. 3, our framework generates and maintains the lineage of the versions created for a seed table. This feature is not currently available in existing data versioning benchmarks and is an important property in the management of versions [5, 6, 23]. Using this feature, we can explore the following problem: *Given a set of versions $V_1, V_2, \ldots, V_n$, construct a directed version graph, describing the lineage of their generation.* Existing frameworks like DataHub [5] can use this graph for tracking, in which nodes correspond to versions and a directed edge between V_i and V_j represents that V_j was derived from V_i. Being storage-oriented, the edge annotations in DataHub correspond to storage and recreation costs. In our generated graph, the edge annotations are associated with the transformations used to generate the versions.

Latin Name	Kingdom	Genus	Common Name	Size
Didelphis virginiana	Eukarya	Didelphis	Virginia Opossum	Medium
Latrodectus mactans	Eukarya	Latrodectus	Black Widow	Small
Alligator mississippiensis	Eukarya	Alligator	American Alligator	Large
Lumbricus terrestris	Eukarya	Lumbricus	Earthworm	Small
Limulus polyphemus	Eukarya	Limulus	Horseshoe Crab	Medium
Mytilus edulis	Eukarya	Mytilus	Blue Mussel	Small
Squalus spp.	Eukarya	Squalus	Dogfish Shark	Medium
Anguilla anguilla	Eukarya	Anguilla	European Eel	Medium
Accipiter cooperii	Eukarya	Accipiter	Cooper's Hawk	Small
Elasmobranchii	Eukarya		Sharks, Rays, Skates	Medium

Table 1: Subset of the Biology Seed Table

Example 3. *Table 1 illustrates a subset of the Biology seed table[a] created for the first version of our benchmark (see below). Table 2 presents a subset of version 5 of the Biology table[b]. Note that there are 5 (0–4) intermediate version between the two as can be seen in the documentation of version 5 (see Biology_5.json). The documentation also provides details on what changed, the exact lineage, creation time and others. Version 5 features two added columns (green left most columns) and three added rows (blue bottom rows), all LLM-generated. In addition, a column was removed (Kingdom in Table 1) and an LLM-generated cell update (9th row marked in orange) was applied. Note that the added values are valid (yet not verified). For example,* `Dogfish Shark` *habitat is the* `Ocean` *and the* `Black Widow` *is, in fact,* `Carnivorous`*. Using the provided documentation, we can see, for example, that the cell update was done in the generation of version 3 (see Biology_3.json), the provided source is allaboutbirds.org/guide/Coopers_Hawk/id/# and the attached confidence for the change is 95%.*

[a] full table: github.com/danielcfox/genbenchver/blob/main/tables/Biology.csv
[b] full table: github.com/danielcfox/genbenchver/blob/main/tables/Biology_5.csv

3 BENCHMARK DESCRIPTION

We now describe a benchmark that we create using our benchmark generator. A summary of the existing benchmarks and the new one is given in Table 3. More details are provided below.

3.1 Existing Data Versioning Benchmarks

Existing benchmarks include the Semantic Data Versioning Benchmark (SDVB) we manually curated [24] and the Auto-Pipeline

Diet	Habitat	Latin Name	Genus	Common Name	Size
Omnivore	Deciduous and Pine Forests, Urban Areas	Didelphis virginiana	Didelphis	Virginia Opossum	Medium
Carnivore	Varied, including human structures	Latrodectus mactans	Latrodectus	Black Widow	Small
Carnivore	Freshwater Swamps, Marshes, Rivers	Alligator mississippiensis	Alligator	American Alligator	Large
Detritivore	Soil	Lumbricus terrestris	Lumbricus	Earthworm	Small
Omnivore	Estuaries, Shallow Ocean Water	Limulus polyphemus	Limulus	Horseshoe Crab	Medium
Insectivore	Urban Areas, Forests	Mytilus edulis	Mytilus	Blue Mussel	Small
Carnivore	Ocean	Squalus spp.	Squalus	Dogfish Shark	Medium
Carnivore	Freshwater and Marine Water	Anguilla anguilla	Anguilla	European Eel	Medium
Carnivore	Woodlands, Forests	Accipiter cooperii	Accipiter	Cooper's Hawk	16.15 in
Carnivore	Ocean	Elasmobranchii		Sharks, Rays, Skates	Medium
Herbivore	Freshwater	Mayaheros urophthalmus	Cichlasoma	Convict Cichlid	6 inches
Carnivore	Marine	Carcharodon carcharias	Carcharodon	Great White Shark	15-20 feet
Omnivore	Terrestrial	Canis lupus familiaris	Canis	Domestic Dog	24-36 inches

Table 2: Subset of Version 5 in the Biology Benchmark

Benchmark	Topic (Name)	Original Tuples	Created Tuples	Original Attributes	Created Attributes	Versions	Version-pairs	Linage Length	Meta-data
Auto-Pipeline [26]	**GitHub data pipelines (GitHub)**	1–200k	0	1–27	0	2–4	158	1	-
SDVB [24]	**Movies and TV shows (IMDB)**	1,000	0	6	0	72	29	1	-
SDVB [24]	**NBA Players (NBA)**	11,700	0	9	0	68	27	1	-
SDVB [24]	**Wines Reviews (WINE)**	129,971	0	6	0	72	29	1	-
SDVB [24]	**Iris Flowers (IRIS)**	150	0	5	0	58	22	1	-
SDVB [24]	**Titanic Passengers (TITANIC)**	891	0	6	0	72	29	1	-
Ours (VerLLM-v1)	**Living Organisms (Biology)**	10	11	11	8	18	109	12	+
Ours (VerLLM-v1)	**Regional Climate (Climatology)**	7	6	11	2	18	108	10	+
Ours (VerLLM-v1)	**Plant-based Food (Horticulture)**	16	21	12	3	19	125	11	+
Ours (VerLLM-v1)	**Classical Literature (Literature)**	10	5	10	6	20	105	9	+
Ours (VerLLM-v1)	**Mythical Creatures (Mythology)**	13	13	4	9	20	131	7	+

Table 3: Data Versioning Benchmarks including Auto-Pipeline, SDVB, and VerLLM-v1 (Ours)

Benchmark [26] we adopted, both were reused by Lou et al. [16] and Glavic et al. [8]. Auto-Pipeline [26][1] was originally designed for data-pipelines and was transformed into a versioning benchmark by Shraga and Miller [24]. It contains tables of different topics, each containing 2–4 versions and range in size. SDVB was directly generated for data versioning. Its creation was mostly manual and included multiple steps to achieve valuable verification.

Both benchmarks do not contain documentation of transformations that were applied, lineage and meta-data including, for example, the description of the table. These features are available in the first version of our benchmark generator – VerLLM-v1.

3.2 New Versioning Benchmark (VerLLM-v1)

VerLLM-v1 uses the Mistral.AI Mixtral 8x7b Instruct model version 0.1 LLM (mistralai/Mixtral-8x7B-Instruct-v0.1) [27]. The framework builds the model and executes the prompt natively using nnsight [1]. Our framework can use any LLM, hosted or open source with small modifications. We used five root tables from a recent work by Pal et. al. [19], who also uses Mixtral 8x7b to generate tables. Compared to existing benchmarks, the lineage of the VerLLM-v1 results in many more version pairs. Each version pair involved documented, intermediate versions from the same seed table. In Example 3, for example, there are 5 intermediate versions between Table 1 and Table 2. It is also noteworthy that, while the seed tables we use are fairly small, we create multiple rows and columns using LLMs, making the variety of changes more challenging for future systems.

3.3 Challenges and Future Outlook

The new benchmark addresses new problems for a which a solution is yet to-be-developed. Thus, the core future work includes developing such solutions and experimenting with the new benchmark.

LLMs have a limited context length making it challenging to generate large data. In our context, being able to generate versions to a large seed table is the challenge. First, we would have to feed the large table as an input, which requires a significant amount of tokens. Second, we would also require the model to generate large-scale data by having it add a column. Another well-known issue in LLMs is hallucination [11].The problem is also mitigated as the new data only has to seem correct enough to be challenging. This is where a human could be employed to determine veracity if that is important enough. In addition, when crafting our prompts, we instruct the model to "explain the source of the new data"; however, we still do not have a systematic way of verifying the sources. Also, it is known that the LLM's response to an input prompt can change with even subtle changes to the prompt [15, 28]. We have experimented with several prompt designs but note that with new models becoming available in an accelerated rate (e.g., Llama-3 [2] and GPT-4o [3]), new prompting challenges will arise. Finally, applying domain-specific knowledge to generate complex transformations, while out of the scope of this effort, was purposely kept in mind when developing the framework. Such transformations may be injected into the automated process with little effort.

4 CONCLUSION

Our work offers a unique feature of utilizing LLMs in the creation and maintenance of versions. Our code and the first version of a new benchmark are publicly available on GitHub. We believe that a structured creation of data versioning benchmarks will foster future research and allow the development of new, exciting work.

ACKNOWLEDGMENTS

This work was supported in part by NSF award numbers IIS-2348121 and IIS-2325632. We would like to thank Koyena Pal for her help.

[1] https://gitlab.com/jwjwyoung/autopipeline-benchmarks

REFERENCES

[1] 2024. https://nnsight.net/
[2] 2024. https://llama.meta.com/llama3/
[3] 2024. https://openai.com/index/hello-gpt-4o/
[4] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. https://www.vldb.org/pvldb/vol17/p92-arora.pdf
[5] Anant P. Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya G. Parameswaran. 2015. DataHub: Collaborative Data Science & Dataset Version Management at Scale. In *CIDR*.
[6] Souvik Bhattacherjee, Amit Chavan, Silu Huang, Amol Deshpande, and Aditya Parameswaran. 2015. Principles of dataset versioning: Exploring the recreation/storage tradeoff. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 8. NIH Public Access, 1346.
[7] Alexander Brinkmann, Roee Shraga, Reng Chiz Der, and Christian Bizer. 2023. Product Information Extraction using ChatGPT. *arXiv preprint arXiv:2306.14921* (2023).
[8] Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, Donatello Santoro, and Enzo Veltri. 2024. Similarity Measures For Incomplete Database Instances. In *EDBT*. 461–473.
[9] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An Intelligent Data Lake System. 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, 2097–2100. https://doi.org/10.1145/2882903.2899389
[10] The home of the U.S. Government's open data. 2020. https://data.gov/
[11] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.
[12] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. *Mistral 7B.* Technical Report.
[13] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), Article 9. https://doi.org/10.1145/3588689
[14] Keti Korini and Christian Bizer. 2023. Column Type Annotation using ChatGPT. In *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023 (CEUR Workshop Proceedings, Vol. 3462)*. CEUR-WS.org. https://ceur-ws.org/Vol-3462/TADA1.pdf
[15] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.* 55, 9, Article 195 (jan 2023), 35 pages. https://doi.org/10.1145/3560815
[16] Yuze Lou, Chuan Lei, Xiao Qin, Zichen Wang, Christos Faloutsos, Rishita Anubhai, and Huzefa Rangwala. 2024. DataLore: Can a large language model find all lost scrolls in a data repository? (2024).
[17] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116
[18] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973
[19] Koyena Pal, Aamod Khatiwada, Roee Shraga, and Renee J. Miller. 2023. Generative Benchmark Creation for Table Union Search. *Proceedings of the VLDB Endowment* (Aug. 2023).
[20] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data Synthesis based on Generative Adversarial Networks. *Proceedings of the VLDB Endowment* 11, 10 (July 2018), 1071–1083. https://doi.org/10.14778/3231751.3231757
[21] Ralph Peeters and Christian Bizer. 2023. Using ChatGPT for Entity Matching. In *New Trends in Database and Information Systems - ADBIS 2023 Short Papers, Doctoral Consortium and Workshops: AIDMA, DOING, K-Gals, MADEISD, PeRS, Barcelona, Spain, September 4-7, 2023, Proceedings (Communications in Computer and Information Science, Vol. 1850)*. Springer, 221–230. https://doi.org/10.1007/978-3-031-42941-5_20
[22] Franck Ravat and Yan Zhao. 2019. Data Lakes: Trends and Perspectives. In *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11706)*, Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil (Eds.). Springer, 304–313. https://doi.org/10.1007/978-3-030-27615-7_23
[23] Maximilian E Schüle, Josef Schmeißer, Thomas Blum, Alfons Kemper, and Thomas Neumann. 2021. TardisDB: Extending SQL to Support Versioning. In *Proceedings of the 2021 International Conference on Management of Data*. 2775–2778.
[24] Roee Shraga and Renee J. Miller. 2023. *Explaining Dataset Changes for Semantic Data Versioning with Explain-Da-V.* Technical Report.
[25] Lei Xu and Kalyan Veeramachaneni. 2018. *Synthesizing Tabular Data using Generative Adversarial Networks.* Technical Report. Cambridge, MA, USA.
[26] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-pipeline: synthesizing complex data pipelines by-target using reinforcement learning and search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2563–2575.
[27] Tianping Zhang, Shaowen Wang, Shuicheng Yan, Jian Li, and Qian Liu. 2023. *Generative Table Pre-training Empowers Models for Tabular Prediction.* Technical Report.
[28] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate Before Use: Improving Few-shot Performance of Language Models. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 12697–12706. https://proceedings.mlr.press/v139/zhao21c.html