A Demonstration of RELIC: A System for REtrospective Lineage InferenCe of Data Workflows

Mohammed Suhail Rehman
University of Chicago
suhail@uchicago.com

Silu Huang
Microsoft Research
silu.huang@microsoft.com

Aaron J. Elmore
University of Chicago
aelmore@cs.uchicago.edu

ABSTRACT

The ad-hoc, heterogeneous process of modern data science typically involves loading, cleaning, and mutating dataset(s) into multiple versions recorded as artifacts by various tools within a single data science workflow. Lineage information, including the source datasets, data transformation programs or scripts, or manual annotations, is rarely captured, making it difficult to infer the relationships between artifacts in a given workflow retrospectively. We demonstrate Relic, a tool to retrospectively infer the lineage of data artifacts generated as a result of typical data science workflows, with an interactive demonstration that allows users to input artifact files and visualize the inferred lineage in a web-based setting.

PVLDB Reference Format:
Mohammed Suhail Rehman, Silu Huang, and Aaron J. Elmore. A Demonstration of RELIC: A System for REtrospective Lineage InferenCe of Data Workflows. PVLDB, 14(12): 2795 - 2798, 2021.
doi:10.14778/3476311.3476347

1 INTRODUCTION

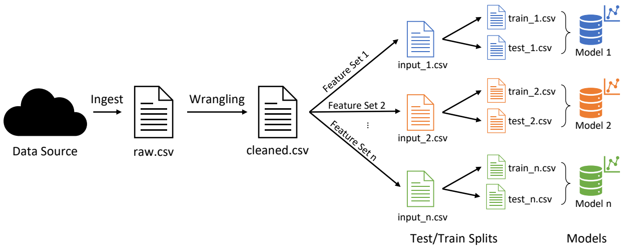


Figure 1: Typical data Analysis/ML prep workflow, with each stage transforming the input and producing artifact file(s).

The emergence of collaborative data science and machine learning (ML) platforms has made it possible for data scientists and analysts to manipulate, process, and analyze modest to large amounts of tabular data in an exploratory or ad-hoc manner. This data analysis *workflow* typically involves dealing with raw data from one or more sources, followed by multiple stages of data preparation, spanning ingestion, cleaning, transformation/analysis, and exported to reports, visualizations or user to train machine learning models.

Figure 1 illustrates a common data science workflow pattern, involving the ingest of a source dataset, cleaning, featurization, test/train splits, and model generation, performed multiple times until the analyst is satisfied with the generated model. As illustrated, it is common for such workflows to generate several artifact files without any lineage information [11].

Lineage information is crucial in understanding the intent and significance of any specific workflow [14]. The lack of accurate lineage information affects downstream reproducibility, debugging, maintenance, explain-ability, and data discovery. Consider the worst-case scenario; an analyst may be presented with a data dump of workflow artifacts with little-to-no documentation with either missing or undocumented code. The analyst will have to manually reconstruct the workflow steps by retracing the derivation of each file in context. Having an automated tool to assess artifact relationships would enable users to contextualize each data artifact; thereby helping to hypothesize their derivation, and retrospectively understand their evolution.

Prior work in data/workflow lineage does not address the problem of retrospective analysis of data artifacts. Most prior work deals with lineage capture or data discovery in annotated data lakes [10, 12, 14, 19], which requires active API calls or logging "hooks" into the workflow tools to record lineage. This not useful in a retrospective context. In the worst-case scenario mentioned above, the data analyst cannot use these tools as the workflow already completed its execution at some point in the past and thus offers no help for the problem at hand.

A closely related field is Query/Program Synthesis [3, 6, 9, 16], which attempts to derive the exact SQL query or program statement(s) that transforms a set of labeled inputs into some desired output. Synthesis may help in looking at pairs of artifacts to determine the exact operation used to transform one to another but does not help in understanding the lineage of n artifacts, where input and output labels to the artifacts may be unclear.

In this demonstration we showcase Relic [21], a retrospective lineage analysis tool that infers the lineage of multiple artifacts that belong to a workflow and generates a lineage graph that most closely resembles the original derivation of the artifacts. Relic uses a variety of similarity metrics and containment-based features to infer lineage relationships in a systematic manner; starting with the most similar pairs of artifacts first and then trying to infer more complex relationships such as joins, groupbys, and pivots. Users can provide custom input in the form of zipped CSV artifact files, or choose a canned demonstration, infer the lineage of the artifacts using the Relic system and interactively view the generated lineage graph.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476347

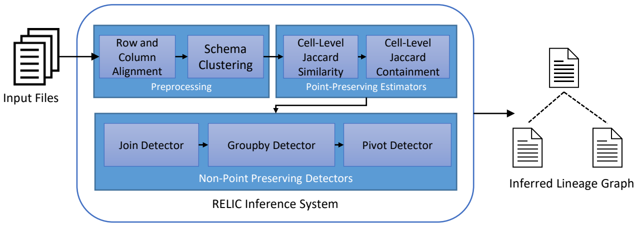


Figure 2: Overall architecture of Relic

2 RELIC ARCHITECTURE

Relic accepts a set of tabular artifact files, with no additional metadata, including versioning or temporal ordering information. The task is to infer the underlying lineage graph that describes the evolution of files in the workflow. Relic generates an undirected[1] tree up to $(n-1)$ edges that represent its best estimate of how the n artifact files evolved from one another in a single workflow[2].

Relic focuses on two-types of transformations, namely *point-preserving transformations* (PPTs), in which there is a 1:1, 1:0, or 0:1 row mapping between source and destination artifacts. A destination row either exists as a modified version of an existing row in the source (1:1), is filtered out (1:0), or is a new row(0:1), in case of a concatenation or row addition. Examples of PPTs include row and column selections, sampling, or spreadsheet-style cell-level edits.

Similarly, transformations with an N:1, 1:N, or N:N row mapping between source and destination artifacts are called *non-point preserving transformations* (NPPTs). Examples of NPPTs include joins, groupbys, and pivots. Relic uses different techniques to infer edges produced by PPTs (fine-grained distance metrics) and NPPTs (operation-specific detectors) respectively.

Our technique (outlined in Figure 2) first involves two preprocessing steps, namely, row and columnar alignment of the input artifacts, followed by clustering of the artifact files by those that have the same schema (set of column names or identifiers). We then compute pairwise distance metrics (Jaccard distances and containment scores) to infer point-preserving artifacts, and add edges within each schema cluster in the order of highest score. We then look at more complex relationships and attempt edge inference between artifacts connected by a join, groupby, or pivot in that order. The final, inferred lineage graph is output to the user for analysis. We further expand on our technique as follows:

Preprocessing: Given a set of input tabular artifact files F_i, Relic first attempts to infer consistent row/column mappings across all artifacts using unique key detection techniques for row matching [13] and column alignment using schema matching techniques [20]. Columns that share the same column label and same datatype (as inferred from the column values) across artifacts are considered to refer to the same column entity. If the row-indices of two artifact files do not share at least 50% of their values, we re-index the artifacts to align them [3]. We then *cluster* the artifacts based on column labels, such that artifacts that share the exact set of column labels are placed into the same cluster.

Inferring PPTs: Two pairwise distance scores are computed for each set of artifacts; the *Cell-Level Jaccard Similarity* (δ_{cell}) and *Cell-Level Jaccard Containment* ($\delta_{contain}$). They are computed for a pair of artifact files F_i and F_j as follows (Equation 2.1):

$$\delta_{cell}(F_i, F_j) = \frac{|V_{F_i} \cap V_{F_j}|}{|V_{F_i} \cup V_{F_j}|} \tag{2.1a}$$

$$\delta_{contain}(F_i, F_j) = \frac{|V_{F_i} \cap V_{F_j}|}{\min(|V_{F_i}|, |V_{F_j}|)} \tag{2.1b}$$

where V_{F_i} denotes the (row-id, column label) indexed cell values in the artifact F_i.

Inferring NPPTs: Relic infers NPPTs using operation-specific detectors. These detectors look for specific column label and value containment patterns that indicate the presence of a specific type of transformation. The detectors have been sketched below; the complete description is available in [21].

The *join* detector evaluates the likelihood that three artifacts F_i, F_j, F_k were involved in a join operation. The detector first looks for schema compatibility by determining which of the two files (labeling them F_r and F_s) could have been joined to create the third (labeled F_t), by looking at the set of column labels. It also looks for a common key column k such that the set of values in common between the source and destination columns are coherently contained, similar to a technique used in [16]. A join score ($\delta_{join}(F_i, F_j, F_k)$) can then be computed from the various containment scores for the artifact triple.

Similarly, the *groupby* detector determines if a pair of artifacts F_i and F_j were involved in a groupby operation. It first assigns a source label F_s to the artifact that has a higher number of rows, as we assume that a valid groupby operation results in a reduction of the number of rows after the transformation. The detector then finds a subset of columns C_g that are in common between F_s and F_t whose values in F_s are fully contained in F_t. It additionally checks that the set of values in C_g are distinct in F_t while not distinct in F_s. If these conditions are satisfied, a groupby score ($\delta_{groupby}(F_i, F_j)$) is computed using the group column containment, schema difference and missing group values for the artifact pair.

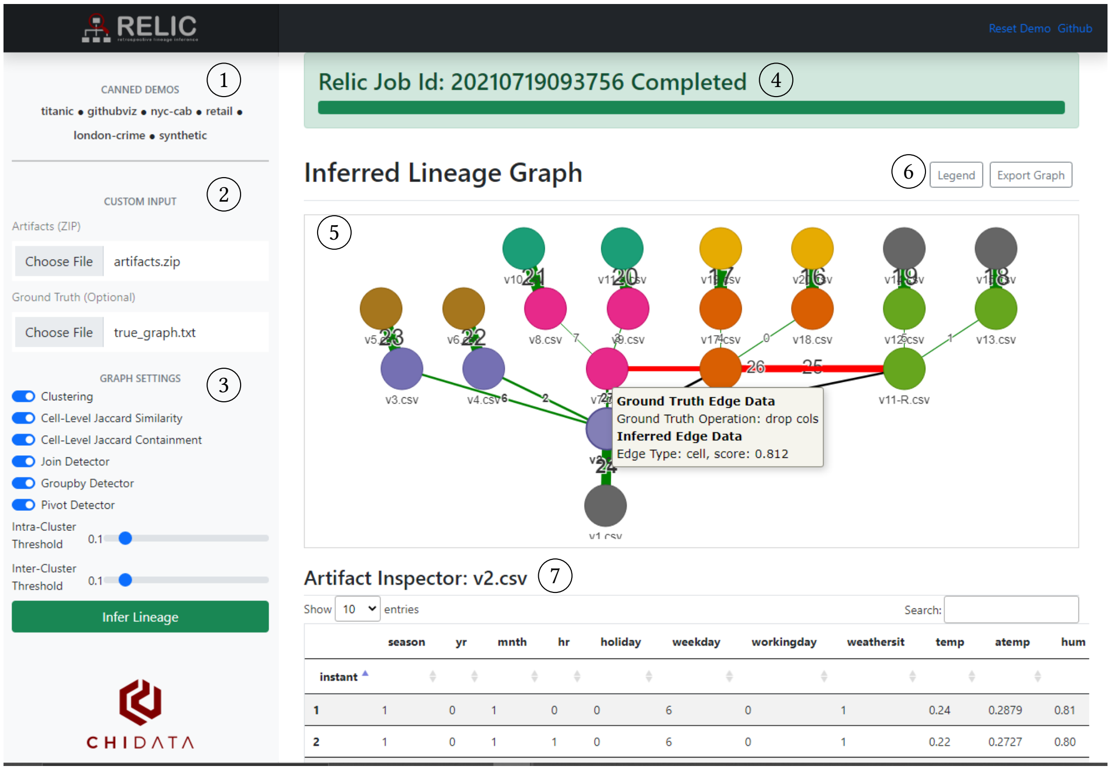
Finally, the *pivot* detector looks for pivots between pairs of artifacts F_i and F_j, looking for value containment in the row-id and column labels in of the artifacts (and assigning it a target artifact label F_t) from the other (hence labeled a source artifact F_s). It determines three separate column mappings, C_i, C_c and C_v based on containment of values in the destination artifact's index, column labels and values. Using these assignments, a final pivot score ($\delta_{pivot}(F_i, F_j)$) is then computed from these containment scores.

Building the Lineage Graph: Relic adds edges in decreasing order of similarity scores to the graph in the following sequence:

[1] Some operations (such as join) have directionality implied, while others, such as column add/drop, may be ambiguous. We plan to explore directional inference in future work.

[2] Relic works on artifacts from multiple mixed workflows; albeit with reduced accuracy.

[3] In case there are no pairs of columns that serve as an appropriate index, the artifact cells are then compared in their physical, materialized order.



CANNED DEMOS

titanic • githubviz • nyc-cab • retail • london-crime • synthetic

CUSTOM INPUT

Artifacts (ZIP)

Choose File artifacts.zip

Ground Truth (Optional)

Choose File true_graph.txt

GRAPH SETTINGS

● Clustering

● Cell-Level Jaccard Similarity

● Cell-Level Jaccard Containment

● Join Detector

● Groupby Detector

● Pivot Detector

Intra-Cluster Threshold 0.1

Inter-Cluster Threshold 0.1

Infer Lineage

Relic Job Id: 20210719093756 Completed

Inferred Lineage Graph

Legend Export Graph

Ground Truth Edge Data

Ground Truth Operation: drop cols

Inferred Edge Data

Edge Type: cell, score: 0.812

Artifact Inspector: v2.csv

Show 10 entries Search:

instant	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum
1	1	0	1	0	0	6	0	1	0.24	0.2879	0.81
2	1	0	1	1	0	6	0	1	0.22	0.2727	0.80

Figure 3: Relic Demo. The user first designates the input in the provided areas (① for canned demo workflows or ② for a custom input). The settings for the inference algorithm are in ③. ④ shows the progress of the running algorithm, while the inferred graph is interactively displayed in ⑤. ⑥ allows the user view the graph legend and export the graph as JSON. ⑦ is an area to inspect individual artifacts as they are clicked on by the user.

(1) δ_{cell} edges within a cluster of artifacts that have the same schema until a threshold (ϵ_{intra_cell})
(2) $\delta_{contain}$ edges within a cluster of artifacts that have the same schema, until a threshold (ϵ_{intra_cell})
(3) δ_{join} edges
(4) δ_{cell} edges between clusters of artifacts that have the same schema until a threshold (ϵ_{inter_cell})
(5) $\delta_{contain}$ edges between clusters of artifacts that have the same schema until a threshold (ϵ_{inter_cell})
(6) $\delta_{groupby}$ edges
(7) δ_{pivot} edges

There are additional implementation details in Relic, such as the exact formulation of detector conditions, scoring functions, thresholds, and tie-breaking techniques which are described in detail in [21]. The ϵ_{intra_cell} and ϵ_{inter_cell} thresholds are user-selectable, as described in the next section.

3 DEMONSTRATION

We will demonstrate Relic using a web-based interface. The demo is interactive; The user can provide any set of CSV files or choose a canned demo, set graph parameters, and run our inference algorithm. While the algorithm runs, Relic visualizes the current state of the lineage graph along with several user interaction options. The detailed demo flow (Figure 3) is as follows:

Step ①:RELIC contains a few canned workflows representing Jupyter notebooks sampled from GitHub[22], which the user can select and interact with.

Step ②:Alternatively, if the user wishes to try Relic with custom input, the user can upload a set of artifact files as a zipped archive of CSV files [4]. The user can optionally provide a ground-truth annotation (as a plain-text graph edge-list) to Relic in case it is available.

Step ③:User sets algorithm parameters; The user can select if pre-clustering should be performed or not, the types of edges that Relic should attempt to infer, as well as thresholds for edge inclusion for artifact pairs that are within and across schema clusters. Once all the options have been set, the user can click on the **Infer Lineage** button, which will submit the input and parameters to the server to start the inference job.

Step ④:While the algorithm is running, status updates are presented to the user describing what phase the algorithm is currently running, along with details of completed and pending algorithm phases. For small workflows of $<$ 1000 rows and $\approx$ 10 − 20 artifacts, Relic takes around 10 seconds to complete. The demonstration uses cached results when available to improve user experience.

[4] The demo is currently limited to 50 artifacts and 100MB total size due to size and run-time constraints

Step ⑤:Once the algorithm is finished, the inferred graph is displayed to the user. Edges have widths proportional to the similarity score computed by Relic. If a ground-truth graph is available, the edges are colored; green edges are true positives, black edges (with an 'X' label) are false negatives, and red edges are false positive edges. Graph edges are additionally labeled with a number that denotes the order in which the edges were inferred by Relic. If the user opted for clustering, the nodes are clustered by schema and are colored to match the cluster they belong to. The graph is interactive; the user can click on each of the nodes and edges to view additional properties. Hovering over an edge shows additional information on the edge type; this includes the score/detector type and score computed by Relic for that edge, and the ground-truth annotation (the exact transformation) of the edge if the ground-truth is provided.

Step ⑥:The user can export the inferred graph as a JSON file if desired.

Step ⑦:Clicking on any node in Step ⑤ provides a preview of the artifact. The user can inspect the table values, order the columns and search for specific values in the artifact file.

The example shown in Figure 3 is that of a machine learning preparation workload of bike sharing data. Starting from a base version, variations in feature selection lead to different branches, in turn leading to different test/train splits and then converted to a form ready to train a machine learning model, similar to the motivational example described in Figure 1.

The key takeaway of our demonstration is the ability to retrospectively infer lineage for common data analysis workflows, and the user interaction provided to tweak the resulting inferred graph as desired. Relic should help users understand how artifacts were derived from one another, allowing for workflow reconstruction and reproducibility without code and/or other lineage information available beforehand.

4 RELATED WORK

Work prior to Relic can be broadly classified into three themes: systems that attempt to capture lineage in a non-retrospective manner, relationship discovery systems, and query synthesis techniques:

Systems such as OrpheusDB [15] and ProvDB [17] provide (git-style) version control for relational datasets. Lineage capture and data discovery systems both from the industry [8, 18, 23] and academia [4, 7, 15] either require access to code or explicit API calls to register and link artifacts in a curated fashion.

ReConnect [1] and Rediscover [2] attempt to discover the relationship for a given dataset pair. Systems such as Aurum [7] and D^{3}L[5] use sampling and estimation techniques to build, maintain, and query datasets in an Enterprise Knowledge Graph. Query Reverse Engineering (QRE) and Query By Example (QBE) [3, 6, 9, 16] focuses on reverse-engineering code (SQL queries, spreadsheet formulae, or pandas statements) that can be used to transform labeled input data into a provided example output. In contrast, Relic tries to determine a derivation tree of data artifacts and does not determine the exact operation used to derive one artifact from another.

REFERENCES

[1] Abdussalam Alawini, David Maier, Kristin Tufte, and Bill Howe. 2014. Helping Scientists Reconnect Their Datasets. In *Proc. 26th International Conference on Scientific and Statistical Database Management* (Aalborg, Denmark) *(SSDBM '14)*. ACM, New York, NY, USA, Article 29, 12 pages. https://doi.org/10.1145/2618243.2618263

[2] Abdussalam Alawini, David Maier, Kristin Tufte, Bill Howe, and Rashmi Nandikur. 2015. Towards Automated Prediction of Relationships among Scientific Datasets. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management* (La Jolla, California) *(SSDBM '15)*. ACM, New York, NY, USA, Article 35, 5 pages. https://doi.org/10.1145/2791347.2791358

[3] Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. 2019. AutoPandas: Neural-Backed Generators for Program Synthesis. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 168 (Oct. 2019), 27 pages. https://doi.org/10.1145/3360594

[4] 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. http://cidrdb.org/cidr2015/Papers/CIDR15_Paper18.pdf

[5] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, New York, NY, 709–720. https://doi.org/10.1109/ICDE48307.2020.00067 ISSN: 2375-026X.

[6] Anna Fariha and Alexandra Meliou. 2019. Example-driven Query Intent Discovery: Abductive Reasoning Using Semantic Similarity. *Proc. VLDB Endow.* 12, 11 (July 2019), 1262–1275. https://doi.org/10.14778/3342263.3342266

[7] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. 2018. Aurum: A Data Discovery System. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1001–1012. https://doi.org/10.1109/ICDE.2018.00094

[8] LF AI & Data Foundation. 2020. Amundsen Project. https://github.com/amundsen-io/amundsen. [Online; accessed 12/21/2020].

[9] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-Output Examples. In *Proc. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)* (Austin, Texas, USA) *(POPL '11)*. ACM, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

[10] Philip J. Guo and Margo Seltzer. 2012. BURRITO: Wrapping Your Lab Notebook in Computational Infrastructure. In *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance* (Boston, MA) *(TaPP'12)*. USENIX Association, USA, 7.

[11] 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* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 2097–2100. https://doi.org/10.1145/2882903.2899389

[12] Alon Halevy, Flip Korn, Natalya F Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, New York, NY, USA, 795–806.

[13] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. 2013. Scalable discovery of unique column combinations. *Proceedings of the VLDB Endowment* 7, 4 (2013), 301–312.

[14] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. 2017. A survey on provenance: What for? What form? What from? *The VLDB Journal* 26, 6 (2017), 881–906.

[15] Silu Huang, Liqi Xu, Jialin Liu, Aaron J Elmore, and Aditya Parameswaran. 2017. Orpheus DB: bolt-on versioning for relational databases. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1130–1141.

[16] Dmitri V. Kalashnikov, Laks V.S. Lakshmanan, and Divesh Srivastava. 2018. FastQRE: Fast Query Reverse Engineering. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. ACM, New York, NY, USA, 337–350. https://doi.org/10.1145/3183713.3183727

[17] Hui Miao, Amit Chavan, and Amol Deshpande. 2017. ProvDB: Lifecycle Management of Collaborative Analysis Workflows. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*. ACM, New York, NY, USA, 7.

[18] Netflix. 2020. Metacat Project. https://github.com/Netflix/metacat. [Online; accessed 12/21/2020].

[19] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-Grained Lineage at Interactive Speed. *Proc. VLDB Endow.* 11, 6 (Feb. 2018), 719–732. https://doi.org/10.14778/3184470.3184475

[20] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.

[21] Mohammed Suhail Rehman, Aaron J. Elmore, Silu Huang, and Aditya Parameswaran. 2021. RELIC: REtrospective Lineage InferenCe. *(Under Preparation)* (2021).

[22] Adam Rule, Aurélien Tabard, and James D Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 32.

[23] Eric Sun. 2020. Open Sourcing WhereHows: A Data Discovery and Lineage Portal. https://engineering.linkedin.com/blog/2016/03/open-sourcing-wherehows--a-data-discovery-and-lineage-portal. [Online; accessed 12/21/2020].