Thesis Proposal: Reconstructing the Lineage of Artifacts in Data Lakes

Mohammed Suhail Rehman

February 5, 2022

1 Introduction

Data lineage, also called provenance or pedigree, is information about data's origin and creation process. Lineage information is used to understand the semantics of processed data, tracing errors from the result of a transformation back to its input data, and estimating the quality of derived data[28]. Data lineage is a crucial component that enables error tracing, troubleshooting, quality estimation, and knowledge building in a variety of data systems ranging from relational databases[27], OLAP warehousing systems[11], NoSQL systems[50] and systems for the Machine Learning Lifecycle[49]. Quality lineage is crucial for building productive and sustainable data lakes; the lack of lineage information can affect the quality of data processed and the insights derived from data lakes[34].

Prior work in database lineage deals with lineage has always been discussed from a *capture*, *record* or *annotate* perspective[27, 8, 35, 22, 49, 20]. The assumption is that the lineage management system is running in conjunction with the core database software, monitoring accesses and logs or allowing for API calls to record the lineage information for every artifact accessed during each data life-cycle activity. These systems effectively solve the lineage problem going forward; none of the proposed solutions allow for retrospective tagging/analysis of data artifacts generated in the past.

On the other hand, existing data discovery solutions[9, 53, 51] do not consider lineage at all; they use techniques from information retrieval to process data from a query-focused perspective. These systems are designed to handle queries such as (*Retrieve all artifacts that match a set of keywords or is a likely join or union candidate for a specific input table*).

To our knowledge, there has not been any work on inferring the lineage of data artifacts in a *retrospective* manner, i.e., long after they have been created, without any supporting metadata about the artifacts that could help with the inference procedure. Such a scenario is common within poorly maintained and governed data lakes, and is often cited as an impediment to the overall vision of reducing the time to insight from vast amounts of data organized in a central data lake[34].

This thesis aims to be the seminal retrospective data lineage thesis - we would like to outline the problem statement, its feasibility, applications, and solutions for the problem. Our two-pronged approach for the thesis aims to organize data artifacts in a lake into their generating workflows; then, lineage can be ascertained within the workflows.

2 Background

2.1 Motivation

Relational databases have been the mainstay of data processing systems since the advent of System R[6]. Within the relational model, operational data collection and storage were typically handled by transaction-focused OLTP systems, and analytical queries, reporting, and data exploration tasks were handled by read-query-optimized OLAP (data warehousing) systems. Decades of database research have helped in the fine-tuned performance optimizations of both these types of systems, as well as the development of hybrid engines that can be used to handle both types of workloads for organized, schema-defined data[26].

However, the explosion in data as part of the expansion of the Internet, smartphones, and sensor networks along with the ever-increasing requirements for complex analyses, machine learning, and analytics have resulted in a shift away from these traditional, rigid, schema-first relational databases[7].

Data analysts and machine learning practitioners often have to process large amounts of data from disparate sources, using multiple tools and processes and collaboratively, often using *lowest common denominator* data interchange formats such as CSV, Apache Parquet, and so on. The resulting decentralization of data away from relational database systems has prompted the industry to converge around the idea of *data lakes* – centralized, shared data repositories that allow for the storage of both raw data from multiple sources, as well as cleaned, transformed data and machine learning models[]. The availability of cheap, easy to integrate distributed file systems such as HDFS and cloud storage such as S3 hastened the move towards this paradigm, making it easy to set up an internet-connected data collection and retention system that is instantly scalable on-demand. According to [5], most Fortune 500 companies operate a two-tiered architecture where a data lake is used to ingest and feed non-relational analytics and machine learning pipelines, while some data is ETL-ed into warehouse systems for more traditional OLAP workloads.

While the data lake vision promises to enable data discovery, reduce time-to-insight for real-time analytics, and enable quick machine learning training and inference turnarounds; more often than not, the data lake reality is that of a *data swamp*, a so-called data dumping ground with data curation tasks taking a back seat. In data swamps, the data quality ends up being suspect and only gets worse over time.

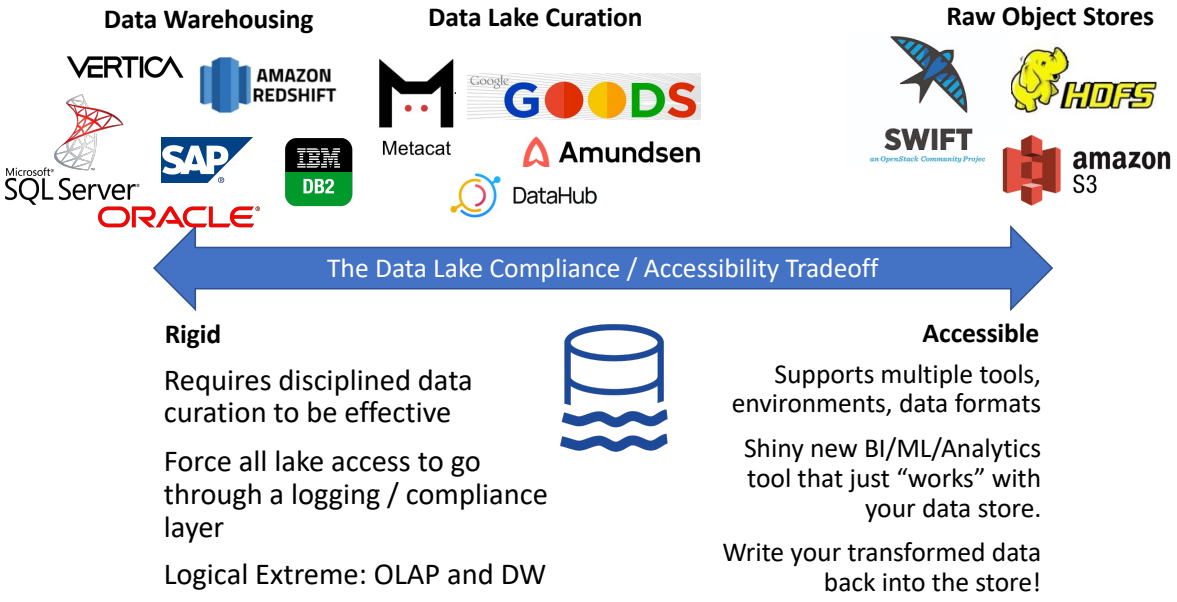


Figure 1: The Compliance/Accessibility Trade-off in Data Lakes

This situation is in part due to a rigidity/accessibility trade-off (Figure 1) that encompasses database systems, data lakes, and other data operations. Traditional relational databases and data warehouses figure on the left, with rigid schema acting as a compliance layer which forces data to be well-structured and enough metadata (in the form of logging/access control/schema and constraint management) to discern lineage information[].

On the other hand, Big data architectures and data lakes are an evolving market with many emerging technologies that are vying for market share. In order to build a customer base and ease customer adoption, there will be a natural tendency to skew right in Figure1 (towards less rigid and accessible solutions that fit into the existing data systems). The resulting provenance crisis in data lakes is a significant problem that impedes the original data lake vision. As we will see in Section 2.2, existing solutions are insufficient to discern lineage in such data swamps in a retrospective manner and require new tools and techniques to aid in the organization of said swamps. This thesis will focus on inferring the lineage of these data artifacts in a retrospective manner; using tools and techniques adapted and modified from various related fields as follows:

2.2 Related Work

Work related to this thesis proposal can be broadly classified into the following themes:

Research in Data Lineage Ikedaet al. have summarized traditional lineage research in a survey[28], while providing a taxonomy of lineage systems for relational and probabilistic databases. Cuiet al. [11] proposed a taxonomy of transformations for the purposes of lineage tracing. The categorization is based on the number of input and output tuples involved in transformation; dispatchers (1:N), aggregators(N:1), and black-box transformations; the paper provides the formalism and tracing procedures for each of these transformation types assuming that the lineage tracing procedure is capturing both the input and output tuples as they are being processed during query execution. A more recent survey by Herschelet al. [24] provides an application-centric overview of all of the workflow lineage solutions proposed in the domains of science, business, data analytics, and general programming.

Online Lineage Capture / Curation / Version Control: Systems such as OrpheusDB [27] and ProvDB [33] provide (git-style) version control for relational databases, they have substantial barriers to adoption—and are unlikely to be used in unstructured, ad-hoc data exploration settings like in data lakes. For data lakes, lineage capture systems both from the industry [20, 21, 8, 5, 35, 1] and academia [8, 27, 9, 4, 52] either require access to code or explicit API calls to register and link artifacts in a curated fashion, which requires compliance and human effort to document lineage while artifacts are being created and offer no help in a retrospective analysis of said artifacts. At the time of writing, the Lakehouse architecture described in [5] is gaining traction among the plethora of solutions being proposed by companies in managing data lake complexity going forward.

Data Discovery: ReConnect [2] and Rediscover [3] attempt to discover the relationship for a given dataset pair. These papers define a space of relevant relationships, generate the conditions for each relationship based on row and column statistics, and then suggest a relationship for a given dataset pair by examining the conditions. ReConnect relies on user feedback to validate candidate relationships, while ReDiscover uses a machine learning model to predict the relationship. However, both ReConnect and ReDiscover only consider a limited relationship set, i.e., containment, augmentation, complementation, template, and incompatible, and cannot handle any mix of them. Systems such as Aurum [9] use sampling and estimation techniques to build, maintain, and query datasets in an Enterprise Knowledge Graph (EKG). The system uses similarity metrics like Jaccard distance and containment to find tables with columns that are most similar to a source table; JOISE[53] speeds up searches for top-k joinable tables in data lakes using a novel set overlap similarity search system. Aurum and JOISE assist in query-centric similarity search; neither system makes any inferences about lineage.

Query Synthesis: A long line of work in Query Reverse Engineering (QRE) and Query By Example (QBE) [48, 44, 12, 45, 29, 18] focuses on reverse-engineering SQL queries that are used to transform one artifact to another. This approach has several issues that make it challenging to apply in our problem context. Firstly, most of this work constrains the inferred SQL queries to some subset of Select-Project-Join-Aggregate (SPJA) and assumes a perfect solution exists in that search space. Second, the input and output artifacts are explicitly labeled, usually within the context of a normalized database schema with established join paths via PK/FK constraints. We argue that artifacts generated by modern data analytics systems do not satisfy either of these conditions. Besides SQL queries, manual edits, scripts, and programs can also be involved in data curation, transformation, and feature engineering. We also note that inferring a concise SQL query itself is a computationally hard problem [48]. We may complicate the problem further if we try to formulate the delta between two artifacts as SQL queries. Finally, current work focuses on a single pair of datasets. Instead, we aim to summarize the relationship among a collection of datasets, which poses additional challenges since it involves all possible dataset pairs.

Document Provenance Systems A separate line of work explores retrospective lineage using similarity scores in information retrieval for plain text documents. Deolalikar et al. [15] demonstrate a system that combines content analysis (cosine similarity over TF-IDF vectors) with filesystem timestamps to generate a list of documents, ordered by the time that is most relevant to a given document. Similarity [13, 14] uses TF-IDF over-extracted named entities to discover provenance using semantic similarity over a set of documents. These approaches are fine-tuned for documents and are unlikely to yield good results over large data tables, as shown by [46], which uses a complex scheme to recover

the semantic meaning of tables present in the web corpus and utilize the semantic keywords and relevant information to power a table search system. In our context, we deal with multiple tables with varying degrees of semantic information or named entities (e.g., a time-series table of sensor values may have little semantic information or named entities that can be extracted from the table). Since the tables are assumed to have some derivation or transformation relationship to one another, they are likely to saturate these semantic information signals to the extent that makes it difficult to assess the fine-grained differences between versions of these individual tables. Table similarity is discussed in [36], which uses a combined Schema and Data similarity metric to measure the distance between two web tables and is an approach that is most closely related to our proposal.

3 Thesis Proposal

3.1 Thesis Statement

We propose a method that retrospectively uses sketch and estimation techniques as well as similarity functions applied at various levels of granularity to:

- Organize data artifacts into the individual workflows that generated them, and,
- Infer a lineage tree for said artifacts that most closely resembles their ground truth derivations.

Our technique is specifically designed to operate with tabular artifacts generated by data wrangling, cleaning, analysis and machine learning preparation operations. Our technique relies solely on the contents of the data artifacts to infer their lineage within the data lake - we operate under the assumption that file system metadata is either absent or unreliable.

3.2 Scope of the Proposal

Data Lakes are designed to be a single unified location for all the organization's data - including raw data ingested from multiple sources as well as the cleaned, transformed intermediate and output data as well as the resulting machine learning models, reports and visualizations.

Inferring the raw data that was responsible for a visualization[] or trained ML model[] is out of the scope for this proposal and is covered in part by prior work[]. Likewise, images, videos, documents, raw text files are also not within the scope of this proposal, as they are well represented within the existing information retrieval literature[].

In our proposed work, however, we focus solely on the *tabular artifacts* and attempt to organize them by workflow and infer the lineage within each workflow. These tabular artifacts are stored in some materialized order, along with column labels.

3.3 Generating the Output Lineage

Given that our starting point is just the artifacts from a data lake without any additional metadata or human input to the inference process, we have decided to infer a derivation *tree* of all the artifacts within the data lake. We plan to show in future work that this may be further loosened with additional input; given timestamps or a general assumption about the data, we could infer directionality;

We shall continue with this simplifying assumption as we first discuss our system, Relic, to infer the lineage of a single workflow:

3.3.1 The RELIC System

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].

[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.

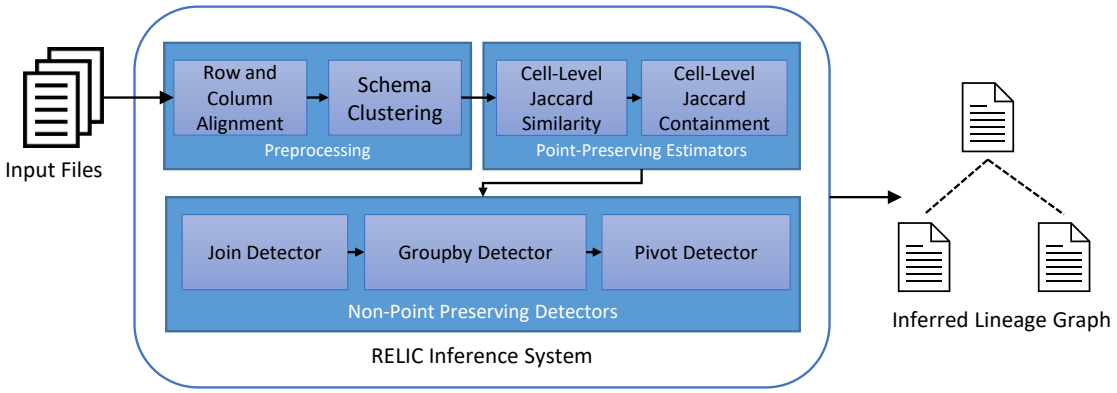
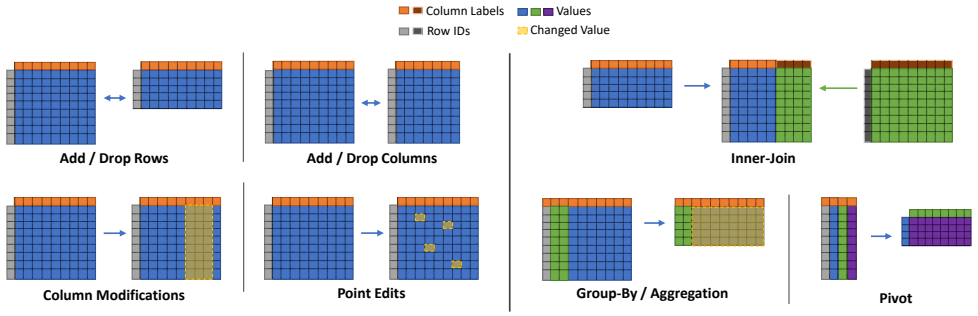


Figure 2: Overall Architecture of RELIC



(a) Point Preserving Transformations (PPT)

(b) Non- Point Preserving Transformations (NPPT)

Figure 3: Types of transformations and how row-ids, columns labels and values change between these transformations.

RELIC focuses on two-types of transformations, namely *point-preserving transformations* (PPTs), and non-point-preserving transformations (NPPTs) (Figure 3. PPTs are transformations with 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 [23]

and column alignment using schema matching techniques [39]. 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 1):

$$\delta_{cell}(F_i, F_j) = \frac{|V_{F_i} \cap V_{F_j}|}{|V_{F_i} \cup V_{F_j}|} \tag{1a}$$

$$\delta_{contain}(F_i, F_j) = \frac{|V_{F_i} \cap V_{F_j}|}{\min(|V_{F_i}|, |V_{F_j}|)} \tag{1b}$$

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 [42].

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 [29]. 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. 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

[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.

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 [42].

3.4 Experimental Evaluation of RELIC

Our evaluation methodology has been designed with the following goals in mind:

- **Overall Accuracy**: How accurate are the inferred lineage graphs are compared to the ground truth?
- **Variation with Workflow Configuration**: How does the accuracy change with workflow configuration (number of artifact files, size of the original artifacts, and number of operations between materialization of artifact files)?
- **Individual Detector Performance**: What were the edge contributions from each detector (cell, containment, join, groupby etc.) and how accurate were they?
- **Runtime Performance**: What is the overall time taken? What part of our technique takes the longest time to run?

3.4.1 Datasets Used

To the best of our knowledge, there are no standardized data analysis benchmarks or workloads that can be used for evaluating our lineage inference technique. Hence, we rely on a combination of workflows derived from Jupyter notebooks published as a corpus [43] and synthetically generated workflows.

Workflows in the Wild To evaluate our technique on realistic workloads, we use a variety of workflows sourced from a notebook corpus [43], AzureML, and Kaggle. Jupyter notebooks are used extensively for data analysis and exploration, and the linear nature of notebook code allows us to automatically execute code, observe outputs, and construct ground-truth lineage in a semi-automatic fashion [41]. However, as flexible as Jupyter notebooks are, they are also notoriously messy [43], as they primarily contain experimental, ad-hoc, and exploratory code that may be manipulated and executed out of order. Additionally, the notebook corpus does not contain any associated metadata or contextual information. We sifted through the corpus to find a sample of notebooks that primarily use pandas for data preparation, load data from valid public URLs, generate at-least 5 dataframes, and belong to a single workflow that deals with data analysis or ML prep (as opposed to homework assignments or tutorial/example notebooks). These notebooks were then executed, verified, and hand-annotated to produce the artifact files and ground truth lineage graphs. These workflows are outlined in Table 1.

Name	**Description**	$(\|\mathcal{F}\|, \|R_{F_0}\|, \|C_{F_0}\|)$
agri-mex	Data Analysis Workflow*	(9, 1300, 9)
churn	Computing Customer Churn*	(5, 3333, 21)
githubviz	Github Repository Visualizations*	(6, 8697,5)
london-crime	Analysis of Crime in London*	(11, 446975, 5)
nyc-cab	Analysis of Cab Rides in NYC*	(17, 150000, 21)
nyc-noise	Analysis of 311 noise complaints*	(24, 136080, 51)
nyc-property	Analysis of NYC property taxes*	(15, 13060, 11)
prop-64	California prop-64 donors*	(10, 56379,8)
retail	Bike rental ML prep†	(21, 17379,16)
titanic	Titanic survivor ML prep‡	(13, 891, 12)

Table 1: List of workflows obtained from Jupyter corpus(*), Azure ML(†), Kaggle(‡).

Synthetic Workflow Generator Our synthetic workflow generator can generate pandas dataframes using the generation parameters listed in Table 2. To generate realistic datasets, we used the Faker [17] python library to generate values in different types of columns. Columns are categorized into different types, i.e., numeric, string, group-able. Group-able columns typically have lower cardinally (such as

Parameter	Description
$\|\mathcal{F}\|$	Number of Artifacts
$\|R_{F_0}\|$	Base Artifact Number of Rows
$\|C_{F_0}\|$	Base Artifact Number of Columns
ν	Materialization Frequency

Table 2: Parameters for generating synthetic workflows.

country or state), which allow for meaningful group-by operations to be performed. *Base artifacts* are artifacts that do not have any ancestors in the workflow. Based on the statistics of scraped dataframes [43], we vary the cardinality of columns for each of the base tables as a function of the row-size of the table, to capture the properties of tables found "in the wild".

After generating a base artifact with the specified number of rows and columns, a synthetic workflow is generated by perturbing the artifact using a randomly selected pandas operation[4] with random parameters. For example, a random column may be selected and a random value within that column maybe selected to be replaced with a new one. Our generator performs quality checks to keep the operations meaningful (e.g., it does not generate empty tables or tables with NaNs, we also limit the number of pivots and groupbys performed in a chain). The out degree of each artifact in the synthetic workflows is also carefully controlled in order to generate a tree-like workflow that is roughly between straight line path and a star-like workflow.

3.4.2 Configurations to be Evaluated

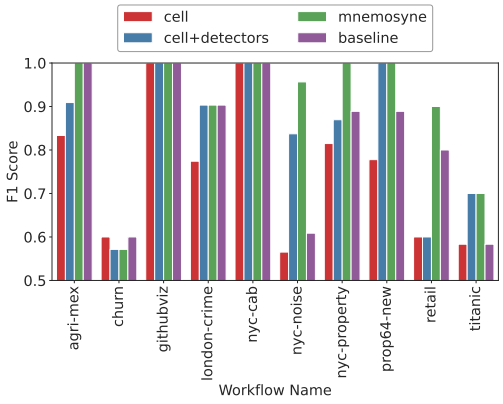
We have tested the following configurations in this paper:

- **cell**: Constructs a spanning tree consisting of edges in decreasing order of δ_{cell}, with no threshold on the lowest edge score.
- **cell+detectors**: Constructs a spanning tree consisting of δ_{cell} edges followed by δ_{join}, $\delta_{groupby}$, δ_{pivot} edges, each in decreasing order. We use $\epsilon_{cell} = 0.1$, $\epsilon_{join} = 0.9$, $\epsilon_{groupby} = 1.0$ and $\epsilon_{pivot} = 0.99$ to determine the minimum edge inclusion threshold for each detector.
- RELIC: Constructs a tree as defined in Section 3.3.1. Artifacts are first clustered by schema, then δ_{cell} edges are added within each cluster. This is followed by δ_{join}, $\delta_{containment}$, $\delta_{groupby}$, δ_{pivot} edges. We use $\epsilon_{intra\_cell} = \epsilon_{inter\_cell} = 0.1$, $\epsilon_{contain} = 0.99$, $\epsilon_{join} = 0.9$, $\epsilon_{groupby} = 1.0$ and $\epsilon_{pivot} = 1.0$. This is the finalized configuration for RELIC.
- **column**: Constructs a spanning tree consisting of edges in decreasing order of column-level jaccard similarity (δ_{column}) edges (as defined in Equation 2), with no threshold on lowest edge score.

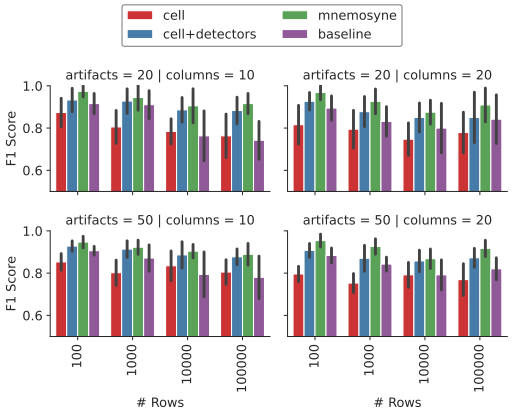
$$\delta_{column}(F_i, F_j) = \frac{\sum\limits_{k \in C_{F_i} \cap C_{F_j}} \left(\underset{jaccard}{\Delta} \left(\pi_{C_k}(V_{F_i}), \pi_{C_k}(V_{F_j}) \right) \right)}{|C_{F_i} \cup C_{F_j}|} \tag{2}$$

All the thresholds mentioned in this section were determined using sensitivity analysis of RELIC on our datasets and found to be the best for both our real-world and synthetic benchmarks. The **column** configuration is adapted from the most closely related work at the time of writing, Aurum [9], which is used for dataset similarity search. δ_{column} is a column-oriented similarity function that returns the average column jaccard similarity between two artifacts. Aurum uses approximate similarity search techniques such as LSH [40], trading accuracy for a reduction of search space, which we have not implemented in the interest of fairness to the **column** configuration.

[4]The operations are point-edits, row sampling, column drops, new derived columns, groupby, merge and pivot.



(a) Real Workflows



(b) Synthetic Workflows. Each bar represents the mean F1 score of 10 separate workflows generated using the (artifact, column, row) configuration. Error bars indicate standard deviation.

Figure 4: Lineage recovery accuracy (F1 Score) for the workflows described in Section 3.4.1

3.4.3 Overall Accuracy

We first evaluate how effective RELIC and other configurations are at recreating lineage for a set of artifacts. Figure 4a shows our results for the 10 real workflows described in Section 3.4.1. The average F1 score of our technique RELIC across all the real workflows is around 0.90. We notice that the **column** configuration performs quite well in a number of workflows (`agri-mex, githubviz, nyc-cab`), but fails to produce sufficiently good results for other workflows. Specifically, `nyc-noise` contains multiple `groupby` and `pivot` operations that are better handled in RELIC via the respective detectors than the generalized column baseline. Similarly, `nyc-property` contains `sample` and `concat` operations that were handled by the clustering and containment scoring in RELIC well. RELIC's performance on `retail` highlights the importance of clustering; the `retail` workflow consists of multiple feature selection and sampling, as well as test/train splits, causing erroneous edges to be inferred between feature splits in the baseline configurations (which did not perform the initial clustering).

Finally, `churn` and `titanic` are the primary outliers in terms of accuracy; this can be attributed to a few operations that are not supported, with RELIC such as `crosstab`, and value scaling/normalization. RELIC still infers the lineage correctly for the rest of the edges in those workflows.

Figure 4b illustrates the accuracy of our method on synthetic workflows with varying number of artifacts, rows, and columns. For each configuration, we generated ten random workflows. The F1 score for RELIC is favorable with an average score of ~ 0.91, and is the best performer on average for all the workflow configurations. Overall, RELIC is able to recover lineage for a wide variety of real and synthetic workflows with reasonable accuracy. The addition of clustering and specialized detectors allows RELIC to infer edges more accurately than the other methods.

3.4.4 Individual Detector Performance

We now assess the contributions of the individual detectors towards our overall accuracy. Since each stage of RELIC uses a different similarity score or detector, we label edges as being inferred at that specific stage, using a specific detector. Grouping the inferred edges by detector, we can compute the detector-specific precision and recall, presented in Table 3. Note that the edges under consideration at each stage decrease as the graph is built, since we add edges between pairs (or triples) of disconnected components. We see the precision of the individual stages are quite good, with groupby and pivot detectors having the lowest precision. We also see that cell level detectors pick up a majority of the edges with a recall rate of around 0.65. The recall scores for $\delta_{contain}$ are low as well, as this detector is invoked after δ_{cell} and δ_{join} detectors to capture `sample` and `concat`-style operations, which is a small fraction of the total edges in the workflow.

Additionally, we found that the groupby and pivot detectors often produce an "equivalent edge" for

Score / Detector	real		synthetic	
	precision	**recall**	**precision**	**recall**
δ_{cell}	0.96	0.64	0.95	0.65
δ_{join}	1.00	0.86*	0.99	0.99*
$\delta_{contain}$	1.00	0.05	0.67	0.02
$\delta_{groupby}$	0.83	0.83*	0.69	0.37*
δ_{pivot}	1.00	0.67*	0.38	0.32*
Total	**0.92**	**0.89**	**0.87**	**0.87**

Table 3: Individual Stage Performance in Relic. The ground truth edge space is all edges in the workflow, except for (*), which indicates the recall score specific to the edges for which that specific detector was designed for.

that operation, which is the same operation applied to a different source dataframe to generate the same result. This is plausible since we found that for every synthetically generated artifact produced by a groupby or pivot in the ground truth, there are, on average, approximately 2 alternate groupby sources and 6 alternate pivot sources that could have produced the same result. If these "equivalent-edges" are taken into consideration, our synthetic workflow (precision, recall) numbers improve significantly to (0.92, 0.50) for the groupby detector and (0.99, 0.84) for the pivot detector. Thus, the stage ordering and thresholds used in Relic provides good opportunities for each of the detectors to find the respective edges, and is reflected in the stage-wise performance metrics presented.

3.4.5 Effect of Materialization Rate on Accuracy

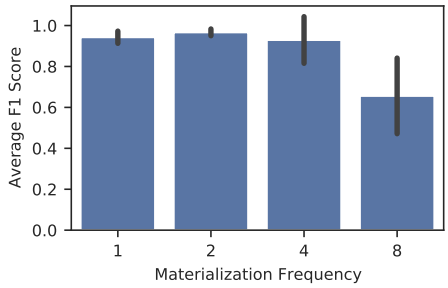
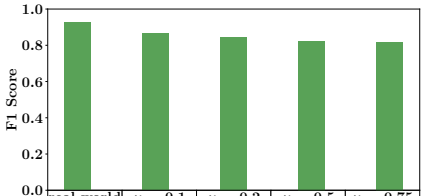


Figure 5: Effect on materialization rate on Accuracy. Each bar represents the average of 5 workflows generated using 50 operations with a base table size of 1000x20. Error bars represent the standard deviation.

In addition to our overall results, we would like to see what effect materialization rate has on accuracy for Relic. Artifacts that are materialized less frequently should exhibit less similarity and may affect many of the conditions that we use in our NPPT detectors, impacting accuracy. Setting a fixed number of operations, rows, and columns (50,1000,20), we vary the materialization rates from 1 (which generates an artifact after every operation) and 8 (which generates an artifact after every 8 operations). We refrain from generating join operations in this experiment as it is hard to keep the number of artifacts fixed as a join requires two inputs. Figure 5 plots the average F1 score for each materialization rate. We see that as the materialization rates increase the accuracy of our technique decreases, especially for non-point preserving operations as the detectors may not correctly capture the containment metrics that we target when several operations are stacked upon each other before an artifact is materialized. However, even with 4 operations between materialization, we have a reasonable accuracy range, which suggests that Relic could be used in more general lineage recovery scenarios.

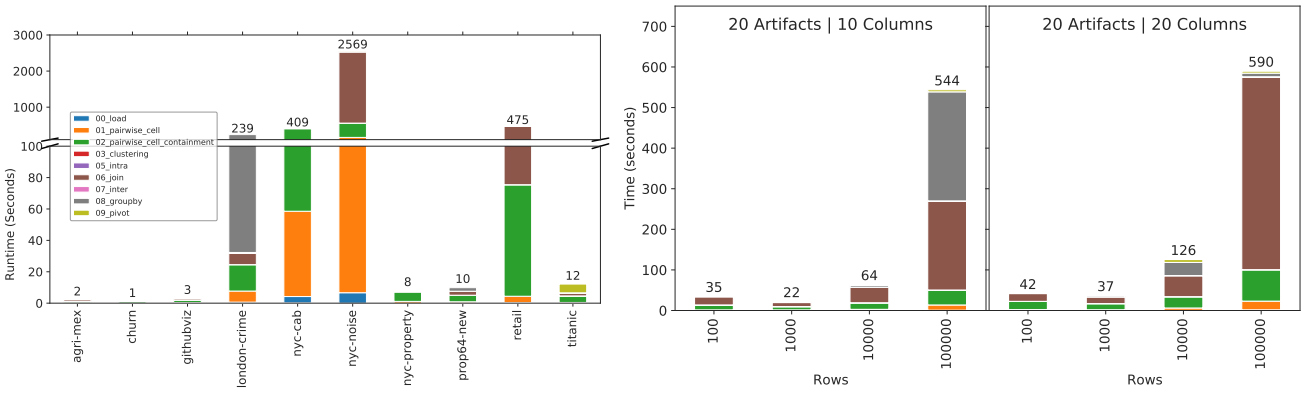
3.4.6 Multiple Workflows

Finally, we evaluate the efficacy of Relic in inferring lineage in a data-lake style setting with artifacts from multiple workflows mixed together in the same directory. Our first mixed workflow (`real-world`) consists of all the artifacts from the workflows in the wild (Section 3.4.1). We also found that the



	real_world	$\mu = 0.1$	$\mu = 0.2$	$\mu = 0.5$	$\mu = 0.75$
# Artifacts	135	2156	2376	2249	2253
F1 Score	0.926	0.864	0.846	0.824	0.818
Cross Workflow FP Edges	0	0	0	2	5

Figure 6: Accuracy of Relic when inferring multiple mixed workflows as part of different data lake configurations. The `real-world` configuration consists of all the artifacts from Section 3.4.1 mixed together, while the others consist of synthetically generated data lakes, with 100 workflows each with varying column overlap (μ), as described in Section 3.4.6.



(a) Real Workflows

(b) Synthetic Workflows

Figure 7: Time to run the lineage inference technique, broken down into individual stages

accuracy and run-time of Relic is primarily dependent on how similar the schema of the artifacts in the data lake is to one another; Evaluating Relic in such an environment requires generating multiple synthetic workflows with controllable schema overlap. This experiment allows us to emulate differing data reuse and similarity levels, typical in lakes with multiple datasets with similar attributes being used simultaneously in different workflows. To synthesize a specific data lake with n workflows, we first generate s distinct *seed workflows* using the techniques described in Section 3.4.1. We then generate $n-s$ *overlapping workflows* in which every base artifact has c columns that overlap with some randomly selected seed workflow. c is chosen at random from a Gaussian distribution $\mathcal{N} = (\mu, \sigma^2)$. We set $n = 100$, $\sigma = 0.1$, $s = 25$ and vary μ from 0.1 to 0.75 to generate four different synthetic data lakes with increasing schema overlap.

Figure 6 shows the performance of Relic on these multiple workflows. We find that Relic can infer edges with an average F1 score of 0.926 on the combined real-world artifacts described in Section 3.4.1. In the synthetic data lakes, we observe that as the column overlap increases, we find a reduction in accuracy and a slight increase in false-positive *cross-workflow edges*, (i.e., false-positive edges inferred by Relic which cross the original ground truth workflow boundaries). Here we find that Relic's performance degrades as the datasets overlap, but suggests that Relic can be used in a multi-workflow data-lake environment to help reconstruct the lineage of a data repository or at least help in clustering artifacts by workflow. For future work we plan to investigate how Relic can be extended for better support of multiple workflow lineage inference.

3.4.7 Time to Infer Workflows

We empirically show how Relic scales in terms of wall-clock time. *Time-to-infer* in our context includes the wall-clock time to load all the artifact tables to memory, perform clustering, compute pair-wise similarity scores and run the individual detectors. All of the experiments were run on a

desktop with a Intel Core i7-8700K CPU, 16 GB of RAM and SSD storage. Figure 7 breaks down the time-to-infer using RELIC for the real and synthetic workflows under consideration. Our method scales in time linear to the table size and quadratic to the number of artifacts, due to the increase in the number of pairwise similarities that need to be computed. The join detector forms the bulk of the timing breakdown, especially for larger workflows, as the Join detector looks at artifact triples and is thus cubic in time complexity. The time taken by individual detectors is dependent on the number of components that remain to be connected at the instant the detector is invoked. Small workflows such as `agri-mex`, `chrun`, `githubviz`, `nyc-property`, `pro64-new` and `titanic` have their workflows inferred within 15 seconds, making interactive usage of RELIC plausible. `london-crime`, `nyc-cab`, `nyc-noise` and `retail` all have large artifacts and multiple tied joins and groupbys, which contributes to the long time taken by those detectors.

3.5 Extending RELIC

The RELIC system presented in section 3.3.1 works well with pandas-centric workflows derived from Jupyter notebooks. We were also able to show the system performing on a very limited set of mixed workflows; Additional techniques will have utilized in order to generalize RELIC for different types of workflows; improvements in efficiency will be required if RELIC is to scale to the size of larger data lakes; they will be discussed next.

3.6 Relaxing RELIC's assumptions

3.6.1 Lineage Graph structure

RELIC infers $n-1$ number of edges to be inferred from the given input set of artifacts, with the goal of creating, at most a single, connected graph of artifacts that represents the lineage, subject to the availability of enough edges that meet the connectivity and similarity detector thresholds.

However, this is a strong assumption. The original lineage may have multiple edges between workflows, as multiple workflows may share the same set of input artifacts. Workflows could have branches and merges, or may have cycles.

We plan to extend our framework in multiple ways to tackle this problem; a number of additional parameters could be introduced to aid in the edge inference and selection process. We can set the total number of edges to be inferred, or set constraints on the in or out-degrees for artifacts to steer the edge selection process towards certain graph architectures.

3.6.2 Schema and Row Matching

RELIC's similarity metrics and detectors heavily rely on schema-matched and row-matched artifacts; Within our pandas workflows, we leaned on the availability of consistently named columns and indices that are matched for accurately computing scores such as the δ_{cell}. This is not always true; columns can be arbitrarily renamed, rows can be sorted and indices can be dropped. For RELIC to be useful in arbitrary data-lake environments, we will need to include some form of schema matching to allow for these deviations from our assumptions.

Schema Matching Valentine [30] is the most-up-to-date benchmarking survey of the state of the art methods in schema matching, and an implementation was readily available for experimenting with pandas dataframes[38]. Our initial experiments have shown that schema-based methods (column names, types and relations) are strike a good balance between speed and accuracy for the retrospective lineage inference application. Fig 8 shows the mean F1 accuracy of schema matching against the synthetic dataset for 4 different schema matching systems for a sample of synthetically generated workflows, organized by operation type:

- **coma**: COMA[16] combines multiple schema-based matchers. Schemata are represented as rooted directed acyclic graphs, where the associated elements are graph nodes con- nected by edges of different types (e.g. containment). The match result is a set of element pairs and their corresponding similarity score.

- **cupid**: Cupid[31] is a schema-based approach. Schemata are translated into tree structures representing the hierarchy of different elements (relations, attributes etc.). The overall similarity of two elements is the weighted similarity of i) Linguistic Matching and ii) Structural Matching.
- **jlm**: A naive instance-based schmea matcher that uses jaccard-levenshtien distance of column pairs, as implemented in [38]
- **sf**: Similarity Flooding[32] is a schema- based matching approach that relies on graphs, and outputs correspondence between any kind of elements (relations, at- tributes, data types) of two given schemata.

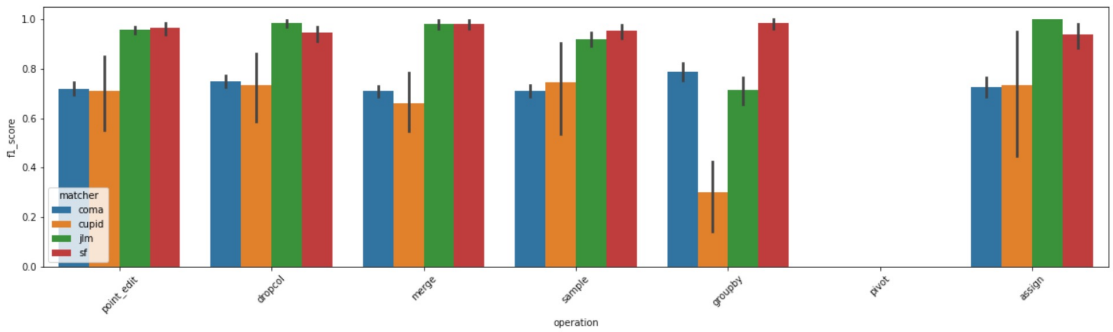


Figure 8: Mean F1 score accuracy of various schema matching techniques grouped by operation type for different synthetic workflows. All of the techniques failed to produce a schema match for the `pivot` operation. Error bars indicate standard deviation.

	matcher	coma						
	operation	assign	dropcol	groupby	merge	pivot	point_edit	sample
rows	columns							
100	10	1.834844	2.000040	2.011723	1.876946	1.921165	1.890412	1.901333
1000	10	2.339861	2.094186	2.104516	2.346695	2.041479	2.263345	1.994538
10000	10	1.775740	1.701545	1.759409	1.847649	1.994599	1.784585	1.626239
50000	10	2.667499	2.446591	2.494659	3.327126	9.587094	2.426881	2.366859

	matcher	jlm						
	operation	assign	dropcol	groupby	merge	pivot	point_edit	sample
rows	columns							
100	10	1.249355	1.426147	0.282732	1.823604	0.036348	1.153427	0.774592
1000	10	11.332245	8.492683	5.345921	17.725999	0.055421	6.027436	5.181377
10000	10	31.966751	15.258460	0.766628	110.787817	0.837367	15.926263	11.893593
50000	10	215.284125	311.198752	77.048847	2109.402365	11.762795	207.477518	134.312776

	matcher	cupid						
	operation	assign	dropcol	groupby	merge	pivot	point_edit	sample
rows	columns							
100	10	915.598904	766.366788	644.769046	688.757686	438.85402	847.549806	1044.24912
1000	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10000	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50000	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN

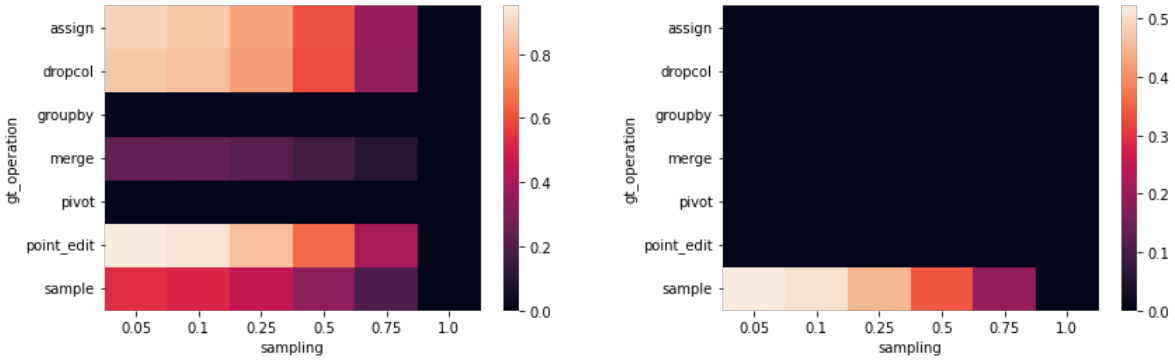
	matcher	sf						
	operation	assign	dropcol	groupby	merge	pivot	point_edit	sample
rows	columns							
100	10	1.078496	1.002309	0.614660	1.309511	0.408079	1.015945	1.092419
1000	10	1.885527	0.774439	0.601465	1.813812	0.835091	0.786582	0.722595
10000	10	0.453799	0.366846	0.452213	1.466936	4.641430	0.447957	0.469665
50000	10	1.663635	1.210670	0.962867	2.395658	33.189649	1.081412	1.746460

Figure 9: Mean run-time (in seconds) to infer the schema among a single pair of tables. Results are organized by table size, schema matcher and operation. `NaN` indicates that the matcher took longer than 3600 seconds to run (DNF).

Figure 9 shows the number of seconds elapsed when inferring the schema between a single pair of artifacts organized by table size and operation. Our initial results show that Similarity Flooding (sf) is the most promising in terms of both accuracy and performance for our application. We plan to further experiment with this technique to find its accuracy among a broader set of operations and within multiple workflows.

3.7 Scaling the Framework

In our previous work, we were able to show that retrospective lineage for single, small workflows can be generated with reasonable accuracy. However, our initial framework scales poorly. Workflows with a large number of artifacts, as well as artifacts that have a large number of rows and columns often taken multiple hours to process on a single machine. Such an approach is not useful for a data lake



(a) Simple Random Sampling

(b) Index-Aware Sampling

Figure 10: Effect of sampling on the δ_{cell} similarity score grouped by operation. The heatmap shows the mean difference in similarity score for each operation (y-axis) and sampling rate (x-axis)

setting - pairwise (or triple-wise) assessment of similarity metrics for hundreds of thousands or millions of artifacts is not practical.

Prior work in databases and information retrieval fields have successfully used sampling and estimation techniques to reduce the computational burden of searching through large datasets to find the most relevant items for a given query. In my thesis I intend to explore two specific avenues for scaling our technique - designing *sampling strategies* to reduce the data scanning involved with the various scoring and detectors, as well as *indexing techniques* specifically designed to reduce the pair and triple-wise comparisons involved in organizing the artifacts into workflows and to automatically prune out non-viable candidate pairs and triples for our detectors.

3.7.1 Sampling Schemes

As seen in Figure 7a, within single-workflow scenarios, RELIC scales poorly when used with artifacts that have a large number of rows and columns. A typical optimization for similar problems is to minimize the amount of data that needs to be scanned, typically by some sampling process.

Sampling for Cell-Level Jaccard Similarity Estimation Our initial experiments with sampling techniques has yielded interesting results, specifically for Point-Preserving Transformations. We have found that random sampling of rows results in a proportionate loss in precision for all of our scoring metrics Figure 10a.

We have since developed a sampling scheme, known as *index-aware sampling* that ensures that the same rows (in their materialised order) are sampled across all the artifacts. This increases the cell-level jaccard similarity scores significantly, as well as the final F1 score for edge inference. In Figure 10b, we can see the effect of index-aware sampling for the cell-level jaccard similarity scores grouped by operation. We will continue to investigate sampling schemes that employ a robust efficiency/accuracy tradeoff in terms of both pairwise accuracy as well as for use in the indexing techniques that we intend to explore in Section 3.7.2.

Detector	Jaccard Similarity	Value Containment	Schema Containment	Cardinality Estimation
δ_{cell}	✓	-	-	-
$\delta_{containment}$	-	✓	-	-
δ_{join}	-	✓	✓	✓
$\delta_{groupby}$	-	✓	-	✓
δ_{pivot}	-	✓	✓	-

Table 4: Estimation components used in each of the similarity functions and detectors in RELIC

Sampling for Detectors We are now looking at leveraging prior work in database sampling to design detector-specific sampling schemes which can strike a balance between the number of rows read and detector accuracy. Table 4 shows the requirements for each of the detectors; all of our detectors

designed thus far have a containment estimation component, while the group-by detector requires a cardinality estimation component.

For value and schema containment, starting with independent Bernoulli sampling, we can arrive at unbiased estimation expressions for a containment query as described in [47]. Given a set of records X and a set containment query Q. If $|X| = m$, sample size b, true set containment cardinality t, and a random variable n_i which indicates that a specific record i is contained in set or not, an unbiased estimate of the containment of Q in X is (Equation 3):

$$\hat{t} = \frac{m}{b} \sum_{i=1}^{b} n_i \tag{3}$$

We further expect to enhance our search into and adapting prior work for cardinality estimation for join queries, and expect to be able to derive a similar sampling strategy which can target our join detector.

For the groupby detector, are exploring prior work in count distinct sketches, histograms and wavelets to find promising directions for accelerating the time taken to score a given artifact pair for groupby operations.

With a combination of index-aware sampling as well as specially-designed unbiased estimators for containment and keyness checks, we expect to be able to significantly improve the scalability of our scheme while reducing the effort required when scoring a given pair or triple of artifacts using our distance metrics and custom detectors.

3.7.2 Indexing Techniques

Locality Sensitive Hashing (LSH) is a common technique used to prune the number of candidate pairs to be evaluated in a search or information retrieval context[54, 9]. We plan to implement an LSH scheme to aid in the lineage inference process; this involves:

- Designing an appropriate vector space (with and without shingling) for each of the similarity scores we plan to use;
- Evaluating a MinHash-based signature scheme for cell-level jaccard similarity along with analysis of the appropriate banding parameters to for lineage inference;
- Evaluating containment sketches to be used for join, groupby and pivot detectors.

Our plan is to first formalize the space and evaluate existing techniques [54, 9] for speeding up Relic. We forsee the use of a combination of LSH indices to cover various similarity score requirements similar to LSH Ensemble [54] in order to arrive at the appropriate pruned index for our application.

3.8 Workflow Generator and Benchmarking Tool

While designing the experiments for Relic, we fashioned a custom synthetic workflow generator that generates realistic tabular artifacts that are typical in data analysis and machine-learning workloads. Our generator is also able to manipulate these artifacts using typical data analysis operations such as select, project and join in order to mimic the type of workflows that we have seen in the Github corpus.

We have found that the core logic in our generator can be retrofitted into a dataframe benchmarking tool (which we call FuzzyData) – similar to YCSB [10]. Our tool can be used to target tabular dataframe systems and generate complex workflows of artifacts that are modified using select, project, join, groupby and pivot operations. We have designed our workflow generator to be extensible – if a user wants to benchmark a dataframe manipulation system for both performance and correctness, they can implement a client which extends our DSL's abstract base classes representing an artifact, operation and workflow. The client thus plugs-in to the core logic of our dataframe benchmarking system. A user can hence either generate a brand-new workflow or replay a previously generator workflow to compare the results of the manipulation across these dataframe systems.

We have implemented clients for both pandas and SQL. We plan to implement and test our dataframe generation system on a variety of dataframe systems such as Pandas, Modin[37], Koalas/SparkDataframes[19], SQLite[25].

4 Research Plan

Table 5, contains research goals and milestones for this thesis. As described in section3.4, we have demonstrated the ability to infer lineage of smaller sized workflows and for data lakes with upto 100 mixed workflows and approximately 3000-4000 artifacts. In order to claim that the lineage inference problem to be solved for data lakes, we need to be able to expand our inference capabilities to much larger data lakes. We plan to do this using sampling and estimation techniques which will reduce the pairwise scoring overheads and the number of pairwise candidates to be evaluated.

4.1 Pending Goals

- Detector Design when sampling artifacts
- LSH Design for grouping artifacts by workflow
- LSH Design for PPTs
- LSH Design for NPPTs
- Randomized re-target-able workflow generator and benchmarking tool
- Implementation and Evaluation

Goal	Expected Completion
Workflow Generator Complete	Jan 2022
Sampling Experiments Complete	Feb 2022
Basic LSH Experiments for Lineage	Feb - March 2022
LSH Design for Detectors	Apr - May 2021
System Integration & Experimentation	June - July 2022
...	...

Table 5: Timeline

References

[1] AirBnB. Dataportal Project, 2020.

[2] Alawini, A., Maier, D., Tufte, K., and Howe, B. Helping scientists reconnect their datasets. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management* (New York, NY, USA, June 2014), SSDBM '14, Association for Computing Machinery, pp. 1–12.

[3] Alawini, A., Maier, D., Tufte, K., Howe, B., and Nandikur, R. Towards automated prediction of relationships among scientific datasets. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management - SSDBM '15* (La Jolla, California, 2015), ACM Press, pp. 1–5.

[4] Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., and Mock, S. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* (2004), IEEE, pp. 423–424.

[5] Armbrust, M., Das, T., Sun, L., Yavuz, B., Zhu, S., Murthy, M., Torres, J., van Hovell, H., Ionescu, A., Łuszczak, A., Świtakowski, M., Szafrański, M., Li, X., Ueshin, T., Mokhtar, M., Boncz, P., Ghodsi, A., Paranjpye, S., Senster, P., Xin, R., and Zaharia, M. Delta lake: high-performance ACID table storage over cloud object stores. *Proceedings of the VLDB Endowment 13*, 12 (Aug. 2020), 3411–3424.

[6] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. System R: relational approach to database management. *ACM Transactions on Database Systems 1*, 2 (June 1976), 97–137.

[7] Bailis, P., Hellerstein, J. M., and Stonebraker, M. Readings in Database Systems, 5th Edition, 2015.

[8] Bhardwaj, A., Bhattacherjee, S., Chavan, A., Deshpande, A., Elmore, A. J., Madden, S., and Parameswaran, A. G. DataHub: Collaborative Data Science & Dataset Version Management at Scale. *arXiv:1409.0798 [cs]* (Sept. 2014). arXiv: 1409.0798.

[9] Castro Fernandez, R., Abedjan, Z., Koko, F., Yuan, G., Madden, S., and Stonebraker, M. Aurum: A Data Discovery System. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (Paris, Apr. 2018), IEEE, pp. 1001–1012.

[10] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10* (Indianapolis, Indiana, USA, 2010), ACM Press, p. 143.

[11] Cui, Y., and Widom, J. Lineage tracing for general data warehouse transformations. *The VLDB Journal The International Journal on Very Large Data Bases 12*, 1 (May 2003), 41–58.

[12] Das Sarma, A., Parameswaran, A., Garcia-Molina, H., and Widom, J. Synthesizing view definitions from data. In *Proceedings of the 13th International Conference on Database Theory - ICDT '10* (Lausanne, Switzerland, 2010), ACM Press, p. 89.

[13] De Nies, T., Coppens, S., Van Deursen, D., Mannens, E., and Van de Walle, R. Automatic Discovery of High-Level Provenance Using Semantic Similarity. In *Provenance and Annotation of Data and Processes*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, P. Groth, and J. Frew, Eds., vol. 7525. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 97–110. Series Title: Lecture Notes in Computer Science.

[14] De Nies, T., Mannens, E., and Van de Walle, R. Reconstructing Human-Generated Provenance Through Similarity-Based Clustering. In *Provenance and Annotation of Data and Processes* (Cham, 2016), M. Mattoso and B. Glavic, Eds., Lecture Notes in Computer Science, Springer International Publishing, pp. 191–194.

[15] Deolalikar, V., and Laffitte, H. Provenance as data mining: combining file system metadata with content analysis. In *First workshop on on Theory and practice of provenance* (USA, Feb. 2009), TAPP'09, USENIX Association, pp. 1–10.

[16] Do, H.-H., and Rahm, E. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases* (Hong Kong, China, Aug. 2002), VLDB '02, VLDB Endowment, pp. 610–621.

[17] Faraglia, D., and Other Contributors. Faker, Feb. 2022. original-date: 2012-11-12T23:00:09Z.

[18] Fariha, A., and Meliou, A. Example-driven query intent discovery: abductive reasoning using semantic similarity. *Proceedings of the VLDB Endowment 12*, 11 (July 2019), 1262–1275.

[19] Foundation, A. S. PySpark Documentation — PySpark 3.2.1 documentation.

[20] Foundation, L. A. . D. amundsen-io/amundsen, Feb. 2022. original-date: 2019-05-14T15:12:40Z.

[21] Foundation, L. A. . D. MarquezProject/marquez, Feb. 2022. original-date: 2018-07-05T22:43:20Z.

[22] Halevy, A., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., and Whang, S. E. Goods: Organizing Google's Datasets. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco California USA, June 2016), ACM, pp. 795–806.

[23] Heise, A., Quiané-Ruiz, J.-A., Abedjan, Z., Jentzsch, A., and Naumann, F. Scalable discovery of unique column combinations. *Proceedings of the VLDB Endowment 7*, 4 (Dec. 2013), 301–312.

[24] Herschel, M., Diestelkämper, R., and Lahmar, H. B. A survey on provenance: What for? What form? What from? *The VLDB Journal 26*, 6 (2017), 881–906. Publisher: Springer.

[25] Hipp, D. R. SQLite Home Page.

[26] Huang, D., Liu, Q., Cui, Q., Fang, Z., Ma, X., Xu, F., Shen, L., Tang, L., Zhou, Y., Huang, M., Wei, W., Liu, C., Zhang, J., Li, J., Wu, X., Song, L., Sun, R., Yu, S., Zhao, L., Cameron, N., Pei, L., and Tang, X. TiDB: a Raft-based HTAP database. *Proceedings of the VLDB Endowment 13*, 12 (Aug. 2020), 3072–3084.

[27] Huang, S., Xu, L., Liu, J., Elmore, A. J., and Parameswaran, A. Orpheus DB: bolt-on versioning for relational databases. *Proceedings of the VLDB Endowment 10*, 10 (2017), 1130–1141. Publisher: VLDB Endowment.

[28] Ikeda, R., and Widom, J. Data lineage: A survey. Tech. rep., Stanford InfoLab, 2009.

[29] Kalashnikov, D. V., Lakshmanan, L. V., and Srivastava, D. FastQRE: Fast Query Reverse Engineering. In *Proceedings of the 2018 International Conference on Management of Data* (New York, NY, USA, May 2018), SIGMOD '18, Association for Computing Machinery, pp. 337–350.

[30] Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., and Katsifodimos, A. Valentine: Evaluating Matching Techniques for Dataset Discovery. *arXiv:2010.07386 [cs]* (Feb. 2021). arXiv: 2010.07386.

[31] Madhavan, J., Bernstein, P. A., and Rahm, E. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases* (San Francisco, CA, USA, Sept. 2001), VLDB '01, Morgan Kaufmann Publishers Inc., pp. 49–58.

[32] Melnik, S., Garcia-Molina, H., and Rahm, E. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th International Conference on Data Engineering* (Feb. 2002), pp. 117–128. ISSN: 1063-6382.

[33] Miao, H., Chavan, A., and Deshpande, A. ProvDB: Lifecycle Management of Collaborative Analysis Workflows. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics* (New York, NY, USA, May 2017), HILDA'17, Association for Computing Machinery, pp. 1–6.

[34] Nargesian, F., Zhu, E., Miller, R. J., Pu, K. Q., and Arocena, P. C. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment 12*, 12 (Aug. 2019), 1986–1989.

[35] Netflix. Metacat, Feb. 2022. original-date: 2016-03-19T20:06:34Z.

[36] Nguyen, T. T., Hung Nguyen, Q. V., Weidlich, M., and Aberer, K. Result selection and summarization for Web Table search. In *2015 IEEE 31st International Conference on Data Engineering* (Apr. 2015), pp. 231–242. ISSN: 2375-026X.

[37] Petersohn, D., Macke, S., Xin, D., Ma, W., Lee, D., Mo, X., Gonzalez, J. E., Hellerstein, J. M., Joseph, A. D., and Parameswaran, A. Towards scalable dataframe systems. *Proceedings of the VLDB Endowment 13*, 12 (Aug. 2020), 2033–2046.

[38] Psarakis, K. Valentine: Matching DataFrames Easily, Jan. 2022. original-date: 2019-06-28T13:50:14Z.

[39] Rahm, E., and Bernstein, P. A. A survey of approaches to automatic schema matching. *The VLDB Journal 10*, 4 (Dec. 2001), 334–350.

[40] Rajaraman, A., and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2011.

[41] Rehman, M. S. Towards Understanding Data Analysis Workflows using a Large Notebook Corpus. In *Proceedings of the 2019 International Conference on Management of Data* (New York, NY, USA, June 2019), SIGMOD '19, Association for Computing Machinery, pp. 1841–1843.

[42] Rehman, M. S., Elmore, A. J., Huang, S., and Parameswaran, A. RELIC: REtrospective lineage InferenCe. *(Under Preparation)* (2021).

[43] Rule, A., Tabard, A., and Hollan, J. D. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, Apr. 2018), CHI '18, Association for Computing Machinery, pp. 1–12.

[44] Tan, W. C., Zhang, M., Elmeleegy, H., and Srivastava, D. Reverse engineering aggregation queries. *Proceedings of the VLDB Endowment 10*, 11 (Aug. 2017), 1394–1405.

[45] Tran, Q. T., Chan, C.-Y., and Parthasarathy, S. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* (New York, NY, USA, 2009), ACM, pp. 535–548.

[46] Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., and Wu, C. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment 4*, 9 (June 2011), 528–538.

[47] Yang, Y., Zhang, W., Zhang, Y., Lin, X., and Wang, L. Selectivity Estimation on Set Containment Search. *Data Science and Engineering 4*, 3 (Sept. 2019), 254–268.

[48] Yilmaz, G. S., Wattanawaroon, T., Xu, L., Nigam, A., Elmore, A. J., and Parameswaran, A. DataDiff: User-Interpretable Data Transformation Summaries for Collaborative Data Analysis. In *Proceedings of the 2018 International Conference on Management of Data* (New York, NY, USA, May 2018), SIGMOD '18, Association for Computing Machinery, pp. 1769–1772.

[49] Zaharia, M., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., and Zumar, C. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Engineering Bulletin 41* (2018), 39–45.

[50] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. In *2nd USENIX workshop on hot topics in cloud computing (HotCloud 10)* (2010).

[51] Zhang, Y., and Ives, Z. G. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, New York, NY, USA, June 2020, pp. 1951–1966.

[52] Zhao, J., Goble, C., Stevens, R., and Turi, D. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 463–472. Publisher: Wiley Online Library.

[53] Zhu, E., Deng, D., Nargesian, F., and Miller, R. J. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data* (New York, NY, USA, June 2019), SIGMOD '19, Association for Computing Machinery, pp. 847–864.

[54] Zhu, E., Nargesian, F., Pu, K. Q., and Miller, R. J. LSH Ensemble: Internet-Scale Domain Search. *arXiv:1603.07410 [cs]* (July 2016). arXiv: 1603.07410.