# DataLoom: Simplifying Data Loading with LLMs

Alexander van Renen
UTN
alexander.van.renen@utn.de

Mihail Stoian
UTN
mihail.stoian@utn.de

Andreas Kipf
UTN
andreas.kipf@utn.de

## ABSTRACT

Schema discovery and data loading is a crucial step in any data analysis pipeline. While this used to be a rare task, in the highly dynamic field of machine learning and modern business intelligence on top of data lakes, today it has become a frequent, but often underestimated, activity. Existing tools often focus on single files, presume prior knowledge of the data on the user's side or a significant amount of manual labor.

In this paper, we improve the process of mapping a "chaotic" set of files to an initial database schema that can then be iteratively refined and loaded. The idea is to take the previously tedious parts of this process and automate them through the use of Large Language Models (LLMs) while leaving already well-understood problems such as constraint discovery to existing algorithms. We thus carefully orchestrate the use of LLMs for the "soft" problems and traditional algorithms for the "hard" problems. This creates a more seamless schema discovery and data loading experience that minimizes the time to first insight for users. We show this vision on modern schema discovery and data loading in our web-based prototype called DataLoom that serves as our demonstration.

**PVLDB Reference Format:**
Alexander van Renen, Mihail Stoian, and Andreas Kipf. DataLoom: Simplifying Data Loading with LLMs. PVLDB, 17(12): 4449 - 4452, 2024. doi:10.14778/3685800.3685897

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/utndatasystems/data-loom-vldb-demo.

## 1 INTRODUCTION

**Motivation.** Before data scientists can create insights or machine learning algorithms can start training, the underlying data needs to be prepared. This usually involves finding the data, creating a schema, doing some cleaning and, finally, loading it into a database system. This process can often be tedious and time-consuming, yet it is extremely important because the quality of the results of the later stages largely depends on the quality of the input data (in short: garbage in, garbage out). Lowering the *time to first insight*, i.e., the time it takes from starting a project to getting the first results, is crucial in a fast-paced environment, such as data science. More precisely, we define the problem of schema discovery as follows: *Given a set of files, create a schema fitting the contained data.*

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685897

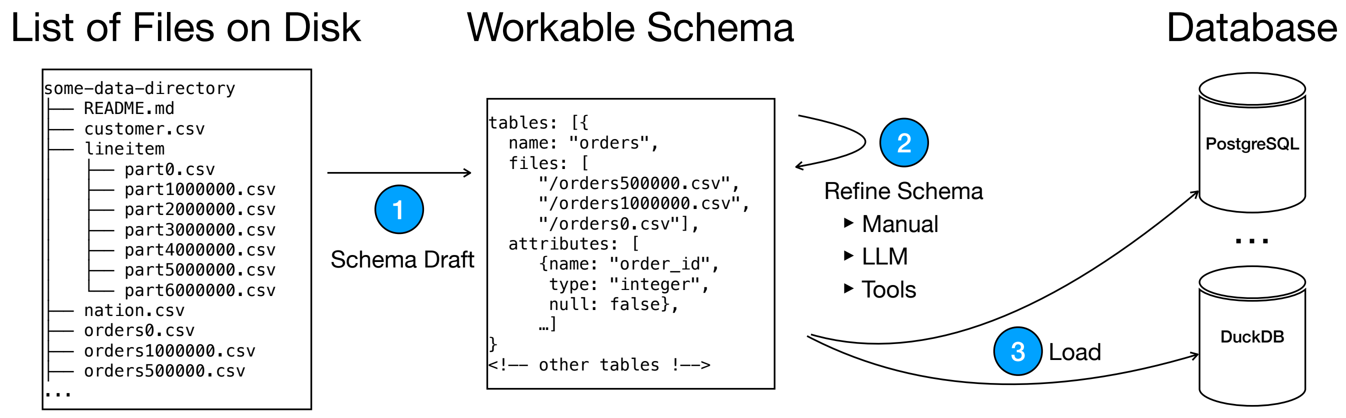
**Existing Approaches.** While there exists a number of algorithms for schema discovery (cf. Section 3), we found that they are falling short in a number of ways: Tools from academia [11] usually focus on an algorithmically complex or interesting part of schema discovery, such as deriving constraints on a given table or finding join keys on a set of tables. Industry-based solutions are often tied to a specific database product, are not extensible, and still require significant manual work, especially in the early stages where data needs to be discovered. In contrast, the goal of this work is to showcase an end-to-end approach with as little work of the user as possible.
**DataLoom.** Our approach, called *DataLoom*, presents a vision on how manual effort can be reduced for schema discovery, refinement, and loading of small to medium sized datasets (i.e., hundreds of tables). It combines existing algorithms from academia to solve the algorithmically well defined problems and leverages Large Language Models (LLMs) to solve the soft problems that used to require manual labor. For instance, constraint discovery is a well-defined problem and can be left to traditional algorithms. Mapping a "chaotic" set of files to a database schema, can be seen as a "soft" problem as it requires domain knowledge and intuition that traditional algorithms lack. The novelty in our approach comes from introducing LLMs into the process and integrating everything into an end-to-end solution to demonstrate how seamless data loading could work. We show that LLMs are effective in creating an initial file to table mapping, allow users to easily perform incremental update on the schema in a safe fashion, and aid in fixing problems during the data ingestion process.
**Example Scenario.** The motivation for DataLoom arose during the practical part of a Master's course on data engineering at our university. Students were given a directory of files containing logging information in `csv` files and were asked to explore the data and analyze the logs. We observed students struggling to import data into a database for analysis and many falling back to manually typing SQL schema files (i.e., `create table [...]`). To show how DataLoom solves this problem, we include the exact same exercise as one of our demonstration scenarios (see Section 4).

## 2 SYSTEM DESCRIPTION

DataLoom is divided into three phases: (1) an automatic, initial schema discovery, (2) a user-driven refinement, and (3) the data loading itself. The goal is to have a fast and seamless process that minimizes tedious work and the *time to first insight* for users. In the following, we describe these three phases, alongside a running example (see Figure 1) that shows an overview of the process with some sample data.

### 2.1 Phase 1: Schema Discovery

**Initial State.** At the beginning of the schema discovery process, we have a directory filled with potential table files on a local or remote drive (e.g., a data lake such as Amazon S3). This is depicted



```
some-data-directory
├── README.md
├── customer.csv
├── lineitem
│   ├── part0.csv
│   ├── part1000000.csv
│   ├── part2000000.csv
│   ├── part3000000.csv
│   ├── part4000000.csv
│   ├── part5000000.csv
│   └── part6000000.csv
├── nation.csv
├── orders0.csv
├── orders1000000.csv
├── orders500000.csv
...
```

```
tables: [{
  name: "orders",
  files: [
     "/orders500000.csv",
     "/orders1000000.csv",
     "/orders0.csv"],
  attributes: [
    {name: "order_id",
     type: "integer",
     null: false},
    …]
}
<!-- other tables !-->
```

**Figure 1: Workflow:** *DataLoom takes a set of files as input (left hand side) and creates an initial schema draft (middle) using classical inference techniques combined with LLMs. Next, the user can inspect the schema in our graphical user interface and incrementally refine manually, with the help of LLMs, or classical algorithms for constraint discovery and data cleaning. Finally, the data can be loaded into a database system (right hand side) and further analyzed and refined.*

on the left hand side of Figure 1; in this case DataLoom is faced with a somewhat irregular version of TPC-H [9]. DataLoom supports comma separated value (CSV) files, but the system design allows to easily extend to other file formats such as Parquet[1] or JSON.

**Files → Tables.** As a first step, DataLoom maps the given set of files to table names. With the unknown and potentially chaotic structure of the input directory layout, we leverage an out-of-the-box Large Language Model (LLM) (namely ChatGPT 3.5 turbo[2]) to obtain an initial mapping from files to tables. In the running example, the LLM associated all `order`-files with an `orders` table. Files that can not be mapped to a table are discarded, this is currently done by the LLM, but in the future we could pre-filter common non-relational files (e.g., `README.md`).

**Column Types.** Given the initial set of tables with their associated files, DataLoom now infers possible data types for each column. This is done by sampling rows from each file mapped to a given table. For each column we pick the most restrictive type that can represent all sampled values. If there is a mismatch in the number of columns, we drop the offending file and add it to the list of discarded files. Whenever data types across files for a single table do not match they are generalized to the most common type, in the worst case this would be `varchar`, as all data can be stored as text.

**Column Names.** Lastly, if the input files for a table do not contain a header row that specifies the column names, DataLoom automatically infers the column names for that table. To do this, we ask the LLM to generate column names, given the table name, the column types, and a sample of the data. In the example, the LLM inferred the first column name of the `orders` table as `order_id`, which is similar to `l_orderkey`, the column's actual name in TPC-H.

**Bias.** Obviously, there might be a slight bias towards the TPC-H schema, as the LLM likely saw TPC-H in its training data. However, the LLM likely saw most common column names for many typical tables that are being used at companies, hence this form of overfitting it not necessarily an issue. In our demonstration, we will also show some lesser known datasets to show that the LLM is not simply memorizing TPC-H. However, a well-known dataset makes for a good example in this presentation.

**Fuzziness.** None of the steps described so far necessarily lead to a correct or optimal schema. The goal of this phase of DataLoom is to provide the user with an initial schema draft upon which to iterate. We argue that an initial, even non-perfect schema, can be greatly beneficial in getting started with large unknown datasets.

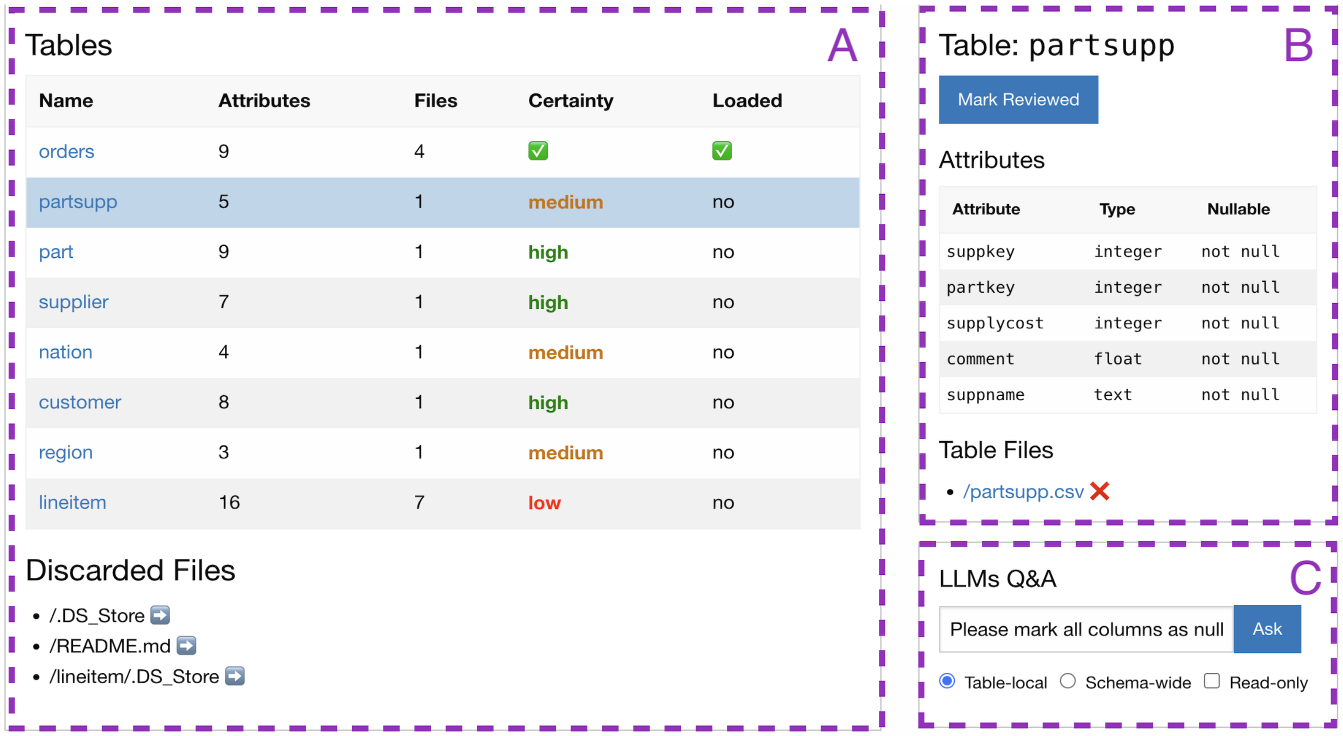
## 2.2 Phase 2: Schema Refinement

**Overview.** Having an initial schema draft, we now describe how DataLoom can be used to refine the schema and correct errors. Figure 2 shows a screenshot of DataLoom's dashboard. The area marked with `A` has a list of the inferred tables. It shows the number of files associated with the table, how many attributes where discovered, and an estimated certainty of the inference process. We calculate the certainty, as a combination of the LLM's confidence in the file to table mapping, the attribute naming, and the attribute type inference. Table details are shown on the right side (area `B`). A user can interactively refine the table either manually, by use of the LLM, or with traditional algorithms.

**Manual Refinement.** First and foremost, DataLoom allows the user to manually update the underlying schema. While this has no scientific novelty, it is an important part of the demonstration, as it gives direct control to the user. We include all standard tasks a user might need, like modifying table names, column names and types, altering the file mapping, or inspecting table files.

**LLM Refinement.** Next, DataLoom allows the user to modify the schema using the LLM (area `C`). This is especially helpful for users less familiar with SQL or for doing larger modifications quickly. To prevent unwanted changes, we allow to limit the scope for LLM inquiries to either a single table or the whole schema. In

[1]github.com/apache/parquet-format [Accessed: 2024-07-17]
[2]platform.openai.com/docs/models/gpt-3-5-turbo [Accessed: 2024-07-17]



Tables

| Name | Attributes | Files | Certainty | Loaded |
|---|---|---|---|---|
| orders | 9 | 4 | ✓ | ✓ |
| partsupp | 5 | 1 | medium | no |
| part | 9 | 1 | high | no |
| supplier | 7 | 1 | high | no |
| nation | 4 | 1 | medium | no |
| customer | 8 | 1 | high | no |
| region | 3 | 1 | medium | no |
| lineitem | 16 | 7 | low | no |

Discarded Files

- /.DS_Store ➡
- /README.md ➡
- /lineitem/.DS_Store ➡

Table: `partsupp`

Attributes

| Attribute | Type | Nullable |
|---|---|---|
| `suppkey` | `integer` | `not null` |
| `partkey` | `integer` | `not null` |
| `supplycost` | `integer` | `not null` |
| `comment` | `float` | `not null` |
| `suppname` | `text` | `not null` |

Table Files

- /partsupp.csv ✗

LLMs Q&A

Please mark all columns as null

[✓ Table-local] Schema-wide Read-only

**Figure 2: DataLoom Dashboard:** *A screenshot of the main dashboard of DataLoom. The user can see the discovered tables, associated files, and attributes. Further, they can use the UI to incrementally refine the schema.*

addition, we implemented a read-only mode, where the user can ask questions about the schema. In the backend, the user's question is sent to the LLM together with the selected part of the schema. If the LLM's confidence in the answer is below a threshold, the user is presented with a diff between the original state and the LLM's proposed updates, which can then be accepted or rejected.
**Constraint Inference.** Lastly, DataLoom implements an interface to traditional algorithms for constraint discovery. Here we mainly build on the work of Naumann et al. [6, 7], who maintain an extensive library of constraint discovery algorithms with a common interface. In DataLoom, users can utilize these algorithms to add constraints to their schema, such as unique constraints, check constraints, or foreign key constraints.

## 2.3 Phase 3: Data Loading

Once a user has inspected, validated, and refined a table, they can flag it as "reviewed", which unlocks the data loading phase for that table. To load a table, we generate SQL code, which can be inspected and then send to a database on our backend. We implemented the connection to the database with psql, allowing for easy extensibility to other database systems[3]. We continue our theme of LLM-aided data loading, by using the LLM to fix errors that might occur during the loading process: Loading errors are sent to the LLM together with the loading query and table's schema. Using prompt engineering, we generate a helpful comment with concrete changes to the SQL statement. The user can inspect the proposed changes before resubmitting the query.

[3]Adding new database backends might also require to adopt our SQL code generator to the specific SQL dialect.

## 2.4 Scaling

DataLoom's interactions (i.e., the prompts that are sent) with the LLM are limited in size or can be partitioned: Column name inference is limited to a single table and thus depends on the number of columns. Similar, the initial file to table mapping depends on the number of files/tables in the dataset. In both cases, we can split the input data (the list of files, of columns) into multiple requests and then merge the results.

## 2.5 Summary

We have shown how DataLoom can interweave LLMs, manual refinement, and classical algorithms to enhance the schema discovery and data loading process. The idea is to utilize LLMs to support users with the "soft" tasks that are traditionally done by humans and leave the "hard" problems (constraint discovery, number crunching) to traditional algorithms. In summary, we explained how LLMs can be used to map files to tables, infer column names, help with schema refinement, and fix loading problems. We present this vision of seamless data loading in our prototype, DataLoom.

## 3 RELATED SYSTEMS AND NOVELTY

Schema discovery and data loading are both well studied problems. In this section, we summarize the most relevant works for this paper. We distinguish ourselves from these existing techniques by adding LLMs to previous pain-points and by combining existing approaches into one end-to-end solution.

**Academia.** Nargesian et al. [5] summarize important problems in the field of schema and constraint discovery, data loading, and data cleaning for data lakes. For constraint discovery, the Metanome platform [7], by Neumann et al. [6], provides a common platform with a generic interface to compare and evaluate competing algorithms. Notably, it ships with a large library of state-of-the-art algorithms, allowing researchers to easily compare results. To aid the user with data cleaning, the Wrangler [3] project explores graphical solutions to specify data transformation rules. In schema discovery, Trummer et al. [10] started exploring the applicability of LLMs for finding column correlations. The JOSIE system [11] uses set similarity search to discover possible join columns in data lakes. Our DataLoom demonstration shows how these point solutions can be combined into a seamless end-to-end solution. An example of a more user-focused approach is the Data Civilizer [2], which implements an end-to-end solution for discovery, governance, and processing on very large schemas. DataLoom, in contrast, improves parts of this process by aiding the user with the use of LLMs.

**Industry.** Given of the importance of schema discovery and data loading, many companies have developed tools to aid in this process. For instance, Amazon, representing cloud platforms, offers AWS Glue[4], which is an umbrella application covering various data engineering tasks with a focus on exchanging data between products from the AWS ecosystem. AWS DataBrew[5], as part of AWS Glue, takes care of analyzing and cleaning data. However, it works on one table at a time and requires manual dataset definitions. Another relevant tool is AWS DataZone[6], which catalogs data from various sources and manages governance. In contrast to those industry solutions, DataLoom shows how many of the manual steps can be aided by using LLMs. Modern database systems, like Snowflake [1] and DuckDB [8], already allow users to directly query or import CSV files without prior explicit schema definition. However, users still have to map CSV files to tables, need to perform data cleaning, and manually fix errors during the loading process. In DataLoom we show how these steps can be aided by LLMs, and data cleaning and constraint discovery can be done with state-of-the-art algorithms. Lastly, Starburst[7] allows users to import multiple tables at a time from a directory. However, the file to table mapping is done statically and expects a fixed schema on disk. In contrast, DataLoom implements an iterative workflow with an LLM backend that captures human intuition and intent.

## 4 DEMONSTRATION

In the following we describe how users can interact with DataLoom during the demonstration (for user interface details please refer to Section 2). Users can either explore the schema discovery and data loading process freely on their own or with a guided set of instructions. Either way, they will move through the three phases of DataLoom: schema discovery, schema refinement, and data loading. They will be able to iteratively refine the schema, load and reload the data. To facility the demonstration of various scenarios covered by DataLoom, we have prepared a number of data sets. Users will be allowed to compare their experience with DuckDB.

**Familiar Dataset.** To allow users to test various features we have prepared TPC-H as a datasets most attendees are familiar with. Here users can see how DataLoom seamlessly maps files to tables, infers column names and types, and loads data, allowing for a running start. Using a familiar datasets allows users to compare the results of DataLoom with their own knowledge of the dataset.

**Unknown Dataset.** In this scenario, users will be able to try DataLoom with two unfamiliar datasets. First, we prepared the SciSciNet dataset [4], which consists of 22 tables and represents a database of scientific publications. And, second, we revisit the dataset that motivated this (cf. Section 1) from a course at our university, where students where tasked to explore a dataset.

**Risk Management.** To avoid networking issues or potential problems with unexpected changes of LLM endpoints, we have an option for DataLoom to work on cached results completely offline. This will assure that we can have a successful demonstration.

[4]AWS Glue https://aws.amazon.com/glue/ [Accessed: 2024-07-17]
[5]AWS DataBrew aws.amazon.com/glue/features/databrew/ [Accessed: 2024-07-17]
[6]AWS DataZone aws.amazon.com/datazone/ [Accessed: 2024-07-17]
[7]Starburst Loading docs.starburst.io/starburst-galaxy/working-with-data/explore-data/schema-discovery.html [Accessed: 2024-07-17]

## REFERENCES

[1] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 215–226. https://doi.org/10.1145/2882903.2903741

[2] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p44-deng-cidr17.pdf

[3] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Wrangler: interactive visual specification of data transformation scripts. In *Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011*, Desney S. Tan, Saleema Amershi, Bo Begole, Wendy A. Kellogg, and Manas Tungare (Eds.). ACM, 3363–3372. https://doi.org/10.1145/1978942.1979444

[4] Zihang Lin, Yian Yin, Lu Liu, and Dashun Wang. 2023. SciSciNet: A large-scale open data lake for the science of science research. *Scientific Data* 10, 1 (2023), 315.

[5] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. https://doi.org/10.14778/3352063.3352116

[6] Felix Naumann. 2013. Data profiling revisited. *SIGMOD Rec.* 42, 4 (2013), 40–49. https://doi.org/10.1145/2590989.2590995

[7] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. *Proc. VLDB Endow.* 8, 12 (2015), 1860–1863. https://doi.org/10.14778/2824032.2824086

[8] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 1981–1984. https://doi.org/10.1145/3299869.3320212

[9] Transaction Processing Performance Council (TPC). 2022. TPC BENCHMARK™ H Standard Specification Revision 3.0.1. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf. [Accessed 28-11-2023].

[10] Immanuel Trummer. 2023. Can Large Language Models Predict Data Correlations from Column Names? *Proc. VLDB Endow.* 16, 13 (2023), 4310–4323. https://www.vldb.org/pvldb/vol16/p4310-trummer.pdf

[11] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 847–864. https://doi.org/10.1145/3299869.3300065