HYPNOS: Interactive Data Lineage Tracing for Data Transformation Scripts

Xiwen Cai, Xiaodong Ge, Kai Xiong, Shuainan Ye, Di Weng, Ke Xu, Datong Wei, Jiang Long, and Yingcai Wu, *Senior Member*, *IEEE*

Abstract—In a formal data analysis workflow, data validation is a necessary step that helps data analysts verify the quality of the data and ensure the reliability of the results. Data analysts usually need to validate the result when encountering an unexpected result, such as an abnormal record in a table. In order to understand how a specific record is derived, they would backtrace it in the pipeline step by step via checking the code lines, exposing the intermediate tables, and finding the data records from which it is derived. However, manually reviewing code and backtracing data requires certain expertise, while inspecting the traced records in multiple tables and interpreting their relationships is tedious. In this work, we propose HYPNOS, a visualization system that supports interactive data lineage tracing for data transformation scripts. HYPNOS uses a lineage module for parsing and adapting code to capture both schema-level and instance-level data lineage from data transformation scripts. Then, it provides users with a lineage view for obtaining an overview of the data transformation process and a detail view for tracing instance-level data lineage and inspecting details. HYPNOS reveals different levels of data relationships and helps users with data lineage tracing. We demonstrate the usability and effectiveness of HYPNOS through a use case, interviews of four expert users, and a user study.

Index Terms—Data Transformation, Data Lineage Tracing, Program Debugging

1 Introduction

Data validation is a necessary step in a formal data analysis workflow that assists data analysts in ensuring data quality. Many data analysts use programming languages (e.g., R or Python) to process data and employ established validation methods (e.g., validation rules like range checking and wrangling routines like null filling) to ensure the quality of tabular data. However, due to the variety of data quality issues, they may still encounter abnormal results during analysis. Furthermore, distinguishing between programming errors, data quality issues, and unexpected data facts often requires manual judgment. In order to make sense of the analysis result, they need to verify both the data and the processing pipeline and explain how the result is derived.

To understand how a specific result is derived, data analysts usually need to perform ad-hoc (i.e., result-driven) data validation by backtracing it in the data pipeline. For example, in the analysis of the growing trend of COVID-19 deaths, a data analyst found that the growth rate of the weekly deaths in Honduras is significantly higher than that of the other countries. To understand how this outlier is derived, he checked his code for errors in data transformations (DTs). Then, in order to know how the growth rate was derived and which values it was derived from, he cached the intermediate results of previous steps and retrieved the records involved in the computation according to his knowledge. Since the dataflow between the original dataset and the final result contains multiple DT steps, backtracing within a single step did not answer his question. The analyst traced the DT operations and intermediate results step by step until he could understand why the growth rate of Honduras is extremely large. Ad-hoc data validation via data tracing is helpful and even necessary for data analysts. However, this process could be tedious and requires certain skills.

Data lineage (DL) methods are leveraged to help data analysts verify the data by capturing the process of transforming data. DL, or data provenance, describes "where data came from, how it was derived, and how it was updated over time" [1]. According to granularity, DL methods can be divided into two categories: schema-level (coarse-grained) DL, which helps track the datasets and data fields used to produce an output dataset; instance-level (fine-grained) lineage, which helps track the exact data items in datasets used to produce an output data item. In many cases, data analysts need DL methods, especially instance-level DL methods, to help them validate data details. Instance-level data lineage tracing (DLT) algorithms have been studied, but it requires certain expertise to perform queries, and it is tedious to interpret traced results step by step.

Researchers have proposed a number of visualization methods to help users analyze DL and understand DTs. Traditionally, node-link diagrams (e.g., VisTrails [2]) and Sankey diagrams (e.g., PROV-O-Viz [3]) are used to present DL graphs. These methods can be used to visualize data pipelines and the column-wise relationships between two tables, but they cannot effectively present the semantics

- *Xiwen Cai, Kai Xiong, Shuainan Ye, and Yingcai Wu are with the State Key Lab of CAD&CG, Zhejiang University, Hangzhou, China. E-mail: {xwcai, kaixiong, sn_ye, ycwu}@zju.edu.cn.*
- *Xiaodong Ge is with Zhejiang Lab, Hangzhou, China. E-mail: xdge1996@gmail.com.*
- *Di Weng is with School of Software Technology, Zhejiang University, Hangzhou, China. E-mail: dweng@zju.edu.cn.*
- *Ke Xu is with the School of Intelligence Science and Technology at Nanjing University, Nanjing, China. E-mail: xuke@nju.edu.cn*
- *Datong Wei is with the Data Intelligence Innovation Lab at HUAWEI Cloud, Hangzhou, China. E-mail: weidatong@huawei.com*
- *Jiang Long is with Zhejiang University, Hangzhou, China. E-mail: longjiang1@zju.edu.cn*
- *Di Weng and Ke Xu are the co-corresponding authors.*

(i.e., key information such as transformation type, involved columns, and row-wise changes) of DTs . Recent works, such as SOMNUS [4] and Datamations [5], explore glyph designs and animation techniques respectively for communicating DT semantics and probing data pipelines. These two methods are effective in presenting the semantics of a single DT step, but they merely support visualizing multi-step data relationships such as the lineage of a specific column. Moreover, none of the existing visualization methods supports both schema-level and instance-level DLT for DT scripts.

In this work, we explore how visualization can be utilized to support DLT tasks for DT scripts. During the long-term collaboration with two data analysts, we summarized the user tasks and proposed a series of design considerations. Based on the user tasks and design considerations, we propose a new visualization system, HYPNOS. HYPNOS consists of a lineage module and a visualization interface. The lineage module leverages the program adaptor from SOMNUS [4] to extract DT semantics from the code and captures both schema-level and instance-level DL through a lineage tracker. Following the mantra for visual information seeking by Shneiderman [6], the visualization interface adopts a coordinated dual-view design that includes a lineage view and a detail view for representing different levels of DL. Through the lineage view, users obtain an overview of the data pipeline, get access to the DT operations and the intermediate tables in the pipeline, and trace schema-level DL. Then, they can expose the intermediate tables, trace instance-level DL of data items, and inspect the details of the traced results on demand. HYPNOS can be applied to facilitating DLT tasks of DT scripts with tens of DT opertations and millions of data records. We present a use case, expert interviews, and a user study to demonstrate the usability of HYPNOS and discuss the derived implications.

The major contributions of this work are as follows:

- The problem characterization and design considerations for multi-level DL visualization.
- A novel visualization system named HYPNOS for supporting interactive DLT and a prototype implementation.
- A use case, expert interviews, and a user study for demonstrating the usability and effectiveness of HYPNOS, and several implications derived from them.

2 Related work

The related work mainly includes visualization of DL, visualization of DTs, and visualization for program debugging.

2.1 Visualization of Data Lineage

In recent years, researchers have proposed a series of visualization methods to help users analyze DL and gain insights for data governance [7]. A common solution is to represent DL graphs as node-link diagrams (e.g., VisTrails [2] and AVOCADO [8]). Like many other types of data (e.g., publication relationships [9], [10], [11], social networks [12], probability network models [13], [14]), data lineage is also in a graph format, which is very suitable for representation using node-link diagrams [15], [16]. Besides, visualizations such as Sankey diagrams (e.g., PROV-O-Viz [3]), radial-based tree layout [17], and Lamport diagrams [18] are leveraged to present DL. Although these methods can effectively visualize schema-level data lineage, they merely support instance-level DLT of tabular data across multiple steps of DTs. More recently, TACO [19] adopts matrices to help users inspect the detailed differences in pairs of homogeneous tables, but it does not support heterogeneous tables and instance-level drill-down operations. Datamations [5] uses animations to convey the exact transitions of data, but animations would result in challenges in recalling the relationships among the data items. In our work, we adopt a mixed approach, which combines a lineage view and a detail view to visualize different levels of DL. The lineage view uses a novel visual design that presents schema-level DL in a node-link style for providing users data pipeline overviews while presenting DT semantics and supporting schema-level DLT. The detailed view presents tables in a spreadsheet format and allows users to trace data items and inspect details on demand.

2.2 Visualization of Data Transformations

Visualization-based methods have been proposed for a wide range of tasks related to DT, such as data wrangling [20], [21] and data integration [22], [23]. Since the previous subsection has already covered DL visualizations, which involve visualizing DT pipelines, we mainly focus on those works that visualize DT semantics in this subsection.

A common method for representing DT semantics is glyphs in a grid style. For example, Kasica et al. [24] leverage colored grids to illustrate the taxonomy of DT operations in computational journalism. SOMNUS [4] presents a design space for visualizing the semantics of individual data transformations and a prototype system for demonstration. In addition to glyphs, animations are used to communicate algorithms and DTs. Animation-based methods such as Datamations [5] can well depict detailed information about data and clearly explain DT processes. However, as mentioned in the previous subsection, animation-based methods are ineffective in the scenario of instance-level DLT due to the challenges in recalling the DL. Glyph-based methods such as SOMNUS can effectively explain the overall DT process and the specific semantics of each DT step through a glyph-based graph. However, it is difficult to present column-level DL due to its inconsistent visual encoding across different DT steps. Generally, existing methods cannot well support scenarios that involve multi-step DTs. Furthermore, none of them can present multi-level DL.

In our work, we propose a novel visualization system named HYPNOS to support DLT tasks. HYPNOS consists of a lineage module (backend) and a visualization interface (frontend). The lineage module leverages the program adaptor from SOMNUS to parse code and uses a lineage tracker to capture DL. The visualization interface adopts a coordinated dual-view design, which consists of a lineage view and a detail view. For designing the lineage view, we explore the design alternatives and extend the glyph design of existing work to derive a concise and consistent DL representation, which is more suitable for DLT scenarios that involves tracing multi-steps of DT. The detail view supports instance-level DLT, which facilitates the process of fine-grained DLT and data validation. We connect the two views while ensuring the stability and independence of the contexts required for both coarse-grained and fine-grained tasks.

2.3 Visualization for Program Debugging

Program visualization leverages graphical representations to illustrate the developing idea, execution behavior, data, or structure of a program [25], [26]. Numerous attempts have been proposed to facilitate the debugging process. Some efforts have been mail to display code-related or task-specific information about the program execution, such as the runtime state [27], [28], interactive overviews of heaps [29], object mutation [30], or the resource behavior on control flow [31]. Some tools record a history of the execution or allow back-stepping in the debugger to provide additional context [32], [33]. Others can expose program behavior by utilizing in-situ visualizations within code [34], [35], [36], [37], or by introducing visualizations within integrated development environments [38], [39]. In addition, FireCrystal [40] and Timelapse [41] visualize and navigate program states to support recording, reproducing, and debugging interactive behavior in web applications. However, none of them can identify and fix bugs concerning the DT semantics in data wrangling programs. SOMNUS [4] is an exception, which visualizes the DT semantics by a glyph-based provenance graph, while it hardly helps in figuring out issues in data. In our work, we not only present DT semantics to help users check code, but also allow users to address data quality issues through tracing and inspecting data of interest.

3 User Tasks and Design Considerations

We were motivated to design HYPNOS by a data analysis project, in which we provided a government department with an analysis report of the Covid trend weekly for more than half a year. In this project, we worked closely with two data analysts who have more than five years of experience. First, the analysts processed data using Python scripts. Then, we generated visualizations and analysis reports in our system using the processed data. Ad-hoc data verification through DLT is an important part of their work. In the data analysis reports, we might find some unexpected results. These results might either reflect a data fact or result from quality issues in the data. To validate these results, we gave them feedback about the data they provided, and they located and solved the data quality problems, or provided us with explanations to prove that the data was reliable. In this process, they generally need to make sense of how the data is transformed, where the data is derived from, and what the original data is. In order to understand their needs, we observed their workflow and discussed with them during the cooperation. According to the observations and discussions, we summarized four important tasks of their DLT process. Based on these tasks, we propose four design considerations.

3.1 User Tasks

T1 Review the Data Pipeline. Before looking for specific issues, data analysts would first review the data pipeline to recall and check the overall data processing logic. They often maintain multiple scripts and switch between different contexts. Through reviewing the code, they recall the general process and the involved DT operations. This helps them obtain a rough overview of the data pipeline, which facilitates the downstream tasks.

T2 Check Data Transformations. To ensure the correctness of data processing, data analysts need to check the involved DTs. They need to locate the key data operations and check the code as well as the execution result. When they find the operations are incorrect, they would modify the code, rerun it, and check the result again. In other cases, they validate the DT operations and trace the DL to locate the problems in the data.

T3 Trace Data Lineage. To find the problems in data, data analysts would trace the DL of a target result. The target result consists of one or more data items in an output table. Data analysts need to find the DTs and the exact data items in the input tables that contribute to the result. From the data source to the final result, a data pipeline usually involves multiple DT steps, it is difficult for users to make sense of the process once. Therefore, users need to iteratively trace the DL by locating a DT operation and exposing the intermediate result step by step.

T4 Inspect Data Details. After the data is traced, data analysts would inspect the details to observe data patterns and determine whether there exist quality issues. For simple data, they would display the data through code functions (e.g., *print(df)* in Python). For complex data, they may export data and rely on external tools (e.g., Excel) to inspect the data. In this process, in addition to observing the traced data items, users may also inspect other data in the table, which serves as the context.

3.2 Design Considerations

C1 Concise Overview of Data Pipeline. In order to help users review the data pipeline (T1) and obtain an overview, a concise graph representation is necessary. Initially, to avoid being overwhelmed by too many details, users need an overview of DL to help them better accomplish the task of schema-level DLT (T3). Details can be exposed on demand through further interactions.

C2 Intuitive Representations of Transformation Semantics. Users need to verify whether they have performed the correct DTs (T2) according to the relationship between the input and output. To achieve this, they should be allowed to see the semantics of the DT intuitively. Intuitive representations of DT semantics would also help users summarize the data pipeline (T1) and locate the interested operations and data (T3).

C3 Explicit Revelation of Data Lineage. Explicitly revealing DL is necessary for supporting interactive DLT (T3). Users need column-level DL to help them validate the general DT process and row-level DL to help them locate specific data items. Explicit column-wise and row-wise data relationships also help users verify the correctness of DT operations (T2).

C4 Multi-scale Inspection of Traced Details. After data tracing, users should be able to see the details (T4), so that they can find problems. Users need to observe the pattern of the traced data to determine whether there are data quality issues. Since the amount of data could be large or small, we need to provide a multi-scale method to help users inspect the data.

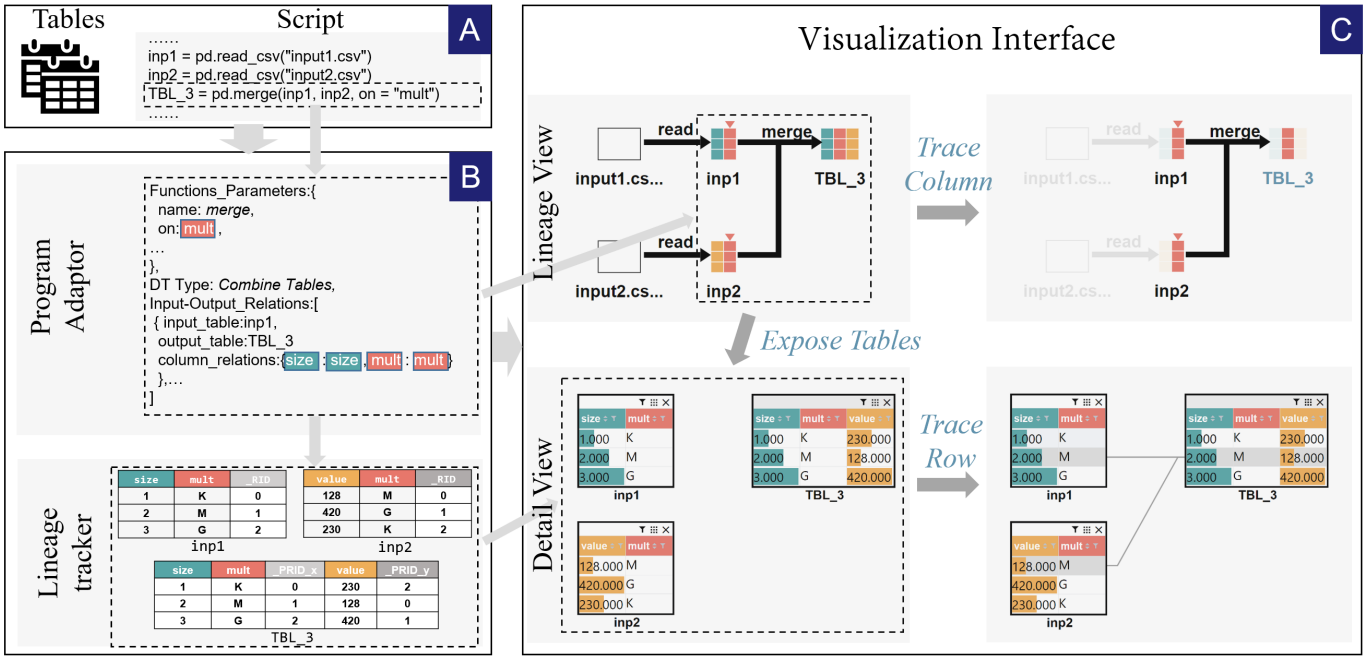

Fig. 1. The architecture of HYPNOS. HYPNOS takes a DT script and the underlining table files as input (A). The lineage module (B) consists of a program adaptor for adapting code and a lineage tracker for capturing DL. The visualization interface (C) consists of a lineage view and a detail view for supporting schema-level and instance-level DLT tasks, respectively.

4 Design of HYPNOS

The architecture of HYPNOS (Figure 1) consists of two parts: a lineage module and a visualization interface. The lineage module takes a script and one or more tables as input and extracts DT semantics of each line of code through a program adaptor from SOMNUS [4]. Then, it captures both schema-level and instance-level DL and provides DLT services through a lineage tracker. Based on the extracted DT semantics, the lineage view provides a lineage graph, which represents the DT pipeline of the script. Users can double-click the tables in the lineage graph to display them in the detail view. The lineage view and detail view support interactions for tracing columns and rows, respectively.

In this work, DL is obtained by parsing the input-output data relations in step-by-step DT operations. To facilitate illustration, we denote the input-output data relations between one step as "table/column/row-wise relationships" and the DL across multiple steps as "table/column/row-level DL".

4.1 Lineage Module

In the lineage module (Figure 1 B), HYPNOS leverages a program adaptor for parsing each line of code to 1) extract function parameters, 2) resolve input-output relations, and 3) inference DT type. Based on the result of the program adaptor, HYPNOS uses lineage tracker for capturing both schema-level and instance-level DL. Since the program adaptor is borrowed from SOMNUS [4], the paper only provides a brief introduction of it.

4.1.1 Program Adaptor

The work flow of the program adaptor consists three steps:
Program Execution. Given the input script and data files, the program adaptor first executes the script to cache the output tables (i.e., the intermediate results) of each line of code, which serve for extracting DT semantics.
Code Parsing. The program adaptor uses rule-based code parsers to extract function parameters and input-output relations of tables and columns from each line in the script. For a certain toolkit in a programming language (e.g., Pandas in Python), the code parser mainly relies on a set of regular expressions and rules to parse the code.
Transformation Inference. After parsing the code, the program adaptor would infer the transformation type of each DT function. The space of transformation types is summarized by Kasica et al. [24], which consists of two dimensions. The first dimension contains five types of operations: *Create* (0:1), *Delete* (1:0), *Transform* (1:1), *Separate* (1:N), and *Combine* (N:1). The second dimension contains three types of data: tables, columns, and rows. To complete the task of transformation inference, the program adaptor uses a manually crafted mapping between functions and transformation types.

4.1.2 Lineage Tracker

HYPNOS captures both schema-level and instance-level DL through a lineage tracker. Capturing schema-level DL is straightforward, which can be achieved through resolving the input-output relations of each line of code. To capture the instance-level DL, we design and implement a fine-grained DL component inspired by by Smoke [42], which adopts row indexes (RIDs) to track the row-level DL. According to the DT type, the lineage tracker either inserts RIDs into the table through revising the function parameters of a DT operation, or uses existing keys (e.g., group key) to record the DL.
Schema-level Data Lineage. Schema-level DL includes table-level DL and column-level DL. Table-level and column-level DL can be obtained from the table-wise and column-wise data relations obtained by the program adaptor.

TABLE 1
Summary of the DTs supported in our work.

Data Type	Operation	Col-Wise Value Changes	Example
Tables	Create	-	load table
	Delete	-	drop table
	Transform	No	fill_na; sort
	Separate	No	subset
	Combine	No	join; concate
Columns	Create	-	add columns
	Delete	-	delete columns
	Transform	Yes	transform
	Separate	Yes	split columns
	Combine	Yes	combine columns
Rows	Create	No	add rows
	Delete	No	filter
	Transform	No	edit
	Separate	No	duplicate
		Yes	split
	Combine	No	group
		Yes	sum

To better support column-level DLT, we categorize the columns involved in a DT operation into three types: lineage columns, directly involved in the transformation of the traced column; contextual columns, not directly involved in the calculation of the traced column but can impact the calculation result; and irrelevant columns, not involved in the transformation of the traced column and do not affect the calculation result. For example, consider a data analyst tracing the "value" column during the DT operation shown in Figure 1. Each cell in "value" in *TBL_3* is derived from the "value" column in *inp2*, making the "value" column a lineage column in *inp1*. Although the cells in the "mult" column in *inp1* does not directly participate in the calculation, any changes in these cells will affect those in the "value" column. Therefore, "mult" is a contextual column. Regardless of the changes in the cells of the "size" column in *inp1*, the result of "value" in *TBL_3* would not be affected. Thus, "size" is an irrelevant column when tracing "value" in this operation. The typical examples of contextual columns in DTs are *groupby* fields, *join* keys, and *filter* conditions.

Instance-level Data Lineage. Instance-level DL includes row-level DL and cell-level DL. Row-level DL is achieved mainly by inserting and preserving RIDs. For a pair of input and output tables *TBL1* and *TBL2*, the lineage tracker inserts an _RID column into *TBL1*, which contains the row indexes of *TBL1*. Then, it adapts the code (if necessary) to make ensure that *TBL1._RID* would be involved in the DT process and becomes a column in *TBL2*. Finally, it renames *_RID* in *TBL2* to *_PRID* (which means previous RID) and then obtain the row-wise relationships between *TBL1* and *TBL2* by mapping *TBL2._PRID* to *TBL1._RID*. After resolving both column-level and row-level DL, cell-level DL can be captured by combining them.

4.1.3 Data Transformation Space and Failure Cases

Table 1 shows the summary of the DTs supported in our work. As mentioned previously, Kasica et al. [24] present a space for characterizing transformation types, which consists of two dimensions (operation class and data type). We extend their space by characterizing whether it involves column-wise value changes in each DT type. During the DT process, some operations do not change the values of data items but rather result in the addition, duplication, deletion, or reordering of values. Common examples of these operations include *join*, *filter*, and *sort*. On the other hand, some operations do change the values, such as performing arithmetic operations on a column or calculating new values based on the content of two columns. We focus on the changes in columns (or fields) because columns are the most important data objects in the DT process. Not only are many DT operations performed on columns, but also many table-wise operations (such as *join*) and row-wise operations (such as *filter*, *sort*, and *aggregate*) often depend on the content of specific columns. In many cases, changes in column values often imply a change in their real-world semantics. For example, when we *aggregate* the daily new deaths, we obtain the weekly new deaths, which represents a different level of data granularity. It is important to note that besides column-wise operations, row-wise operations (*Separate Rows* and *Combine Rows*) can also lead to column-wise value changes, as exemplified by the aggregate operation mentioned earlier.

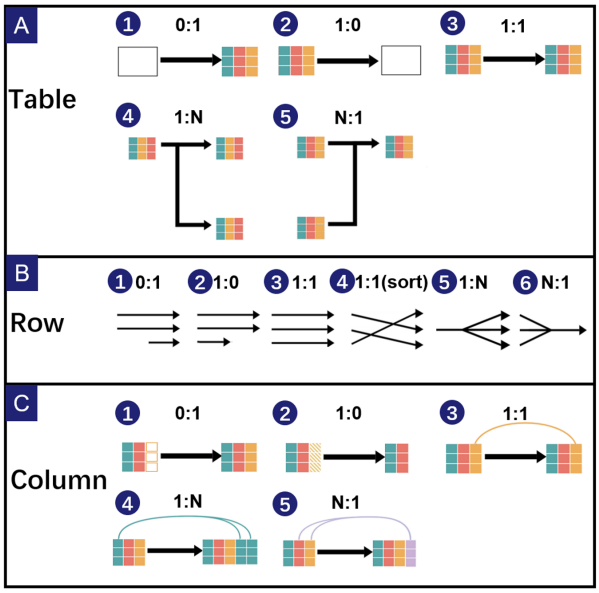

Fig. 2. Visual representations of different types of data transformations. We use rectangles and arrows to represent table-wise (A) and row-wise (B) data changes and relationships. We use curves to indicate column-wise value changes (C).

Similar to SOMNUS, HYPNOS would skip non-assignment statements (e.g., *if ... else ...*) and support only a limited set of DT operations. When encountering an unsupported DT operation, we regard it as a black box function and the lineage tracing would be interrupted. More details about the limitation and scalability issues can be found in the supplemental materials.

4.2 Visual Design

4.2.1 Lineage View

To support scheme-level DLT, we provide a lineage view to help users check the DT process and trace coarse-grained DL. The lineage view adopts a glyph-based method (Figure 2) to provide the context for supporting fine-grained DLT tasks.

The glyphs from existing methods (e.g., SOMNUS [4] and Table Scraps [24]) can effectively represent individual DT

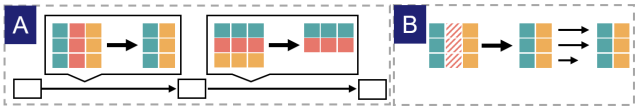

Fig. 3. Difference between the glyphs of existing methods (A) and HYPNOS (B) when applied to a lineage graph.

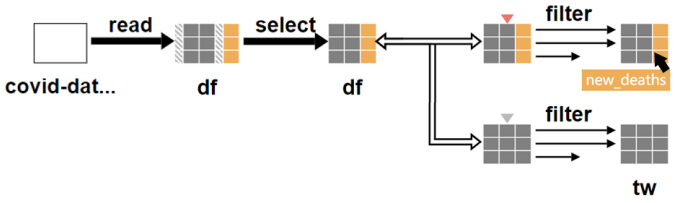

Fig. 4. Applying glyphs to the lineage graph. This figure shows the focus mode for coloring.

operations. However, they entail separate glyphs to represent consecutive steps (Figure 3 A), which causes redundancy and inconsistency: although the output table from the previous step and the input table for the next step are essentially the same table, two distinct table metaphors with inconsistent visual representations are redundantly used. This not only leads to a decrease in spatial efficiency, but also affects users' ability to trace column-level DL. In our method, we aim to combine consecutive steps (Figure 3 B) to simplify DL (**C1**) while better supporting column-level DLT (**C3**). In the following paragraphs, we first introduce the glyph design of HYPNOS (**Coloring**, **Column-wise Relationship**, and **Row-wise Relationship**) and then illustrate how we apply the glyphs to the lineage view (**Lineage Graph** and **Interaction**).

Coloring. In the glyphs used by SOMNUS [4] and Table Scraps [24], color is used to encode rows and columns respectively for operations that cause row changes and column changes (Figure 3 A). To avoid confusion (**C2**) as well as to support column-level DLT (**C3**), we use color to encode column-level DL (Figure 3 B). For a column C, if C has one parent (the column which it is derived from) or two parents with the same color, C's color would be identical with its parent(s). Otherwise, we would encode C with a different color (Figure 2 C5). In the default mode (Figure 1 C), the columns in a table loaded from external files are assigned different colors by a categorical color scale. When the number of columns in the data is limited, this mode can clearly display the column-level DL, allowing users to complete column-level DLT tasks without relying on interactions. However, when there are too many columns, this mode would probably result in visual clutter and color confusion in the lineage view. To address this issue, HYPNOS provides a focus mode, where the columns are gray at the beginning and would be colored when they are traced (Figure 4; note that the triangle marks on the context columns would also be colored). Based on the experience of our domain experts, DLT tasks do not involve many columns in most cases. Thus, the focus mode can be applicable to the majority of cases.

Column-wise Relationship. In subsubsection 4.1.3, we clarified column-wise value changes in different DTs. When tracing column-level lineage, user often pay more attention to the column-wise value changes. To make the column-wise value changes intuitive (**C2**), we highlight the input and output column pairs which involve column-wise value changes through linking them with curves (Figure 2 C3-C5). In representing 0:1 and 1:0 column-wise relationships (Figure 2 C1 and C2), we do not adopt use glyphs similar to the glyphs for representing 0:1 and 1:0 table-wise relationships (Figure 2 A1 and A2), for two main reasons. First, when representing tables, an empty rectangle representing 0 must be in a start or end position, with no preceding or following step, thus avoiding ambiguity. However, a column representing 0 can appear in intermediate tables, which may lead to confusion (it is unclear whether it represents 0 in the previous step or the next step). To address this issue, we have to introduce curves, which conflicts with our intention of using connections to emphasize column-wise value changes. Second, 0:1 and 1:0 column-wise relationships do not require emphasis since they are not prone to errors and are obvious during column-level DLT (as they are the starting and ending columns). Consequently, our primary consideration in designing these two types of operations is to differentiate them to avoid confusion. To achieve this goal, we use empty grids (Figure 2 C1) to represent 0:1 relationship and shadow (Figure 2 C2) to represent 1:0 relationship.

Row-wise Relationship. Our visual design of row-wise transformations (Figure 2 B) is intended to make it intuitive (**C2**) to find that an operation involves row-wise changes and which type of row-wise relationship is involved while reading the lineage graph. Due to the occupation of the color visual channel and the need to combine operations for consecutive steps, we cannot use color and the number of rows like existing methods. Instead, we use arrows to represent the quantity relationship between inputs and outputs. Arrows are commonly used and provide an intuitive depiction of quantity relationships.

Lineage Graph. Figure 4 shows how we apply the glyph design to the lineage graph. We present the function name on the edge (the arrows) of each DT to help users obtain the specific function type. We add triangles above the contextual columns (see subsubsection 4.1.2) in the input table. When there are too many columns in the table, displaying all the columns will take up too much space. In a table, if a column is not involved (as an output/input column) in both the preceding and subsequent operations and has not changed, we will narrow its width to save space. We do not hide it to avoid confusion. For two consecutive DT steps, we connected them by combining the output table of the previous step with the input table of the next step. The combination of two steps of DTs causes no conflict. However, we may encounter the situation that an output table is the input of two or more DTs. In these case, we do not combine these representations to avoid conflict. We use double arrows to indicate that the connected nodes actually represent the same table (Figure 4).

Interaction. When users hover over a column, HYPNOS would show the column name (Figure 4). In the lineage view, we mainly support two types of interactions: trace and expose. Users are allowed to trace the DL of a table or a column (Figure 8). The traced tables or lineage columns are highlighted by making the other tables and columns transparent. When users trace column-level DL, the triangles

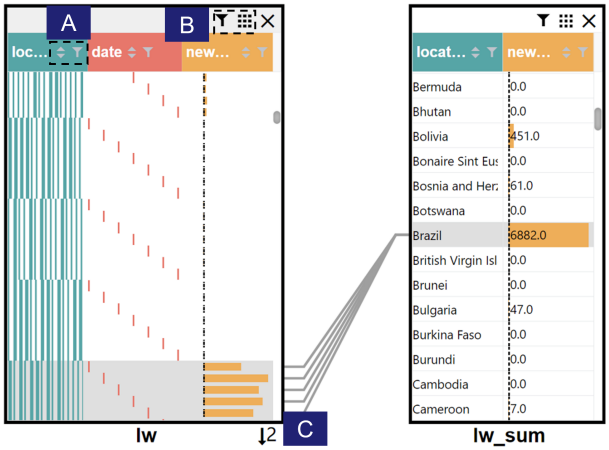

Fig. 5. Visual encoding for different data types in the detail view and the result of tracing data items.

Fig. 6. The Comparison of various approaches for showing categorical data. (A) a larger font size results in less items displayed; (B) a smaller font size results in poorer readability; (C) a normal font size with less row height results in occlusion; (D) barcodes show grouped identical items.

above the context columns would not be transparent, as to indicate that these columns also have an impact on the traced columns. Through this design, we allow users to focus on the traced column without being distracted by irrelevant columns, while also being aware of the presence of context columns. When users double-click a node, the system exposes it in the detail view to show its details. We apply black borders to the exposed tables to help users associate tables in different views. Note that if a user first traces a column and then expose a table, only the traced columns would be displayed in the table; but if a user first expose a table and then trace a column, the content of the exposed table would not be changed for the stability of the context.

4.2.2 *Detail View*

To support instance-level DLT, we provide a detail view to help users trace fine-grained DL and inspect details (**C4**).

Visual Encoding of Tabular Data. To fit the habits of users, we present tables in a spreadsheet form (Figure 5). The color encoding is consistent with that in the lineage view. We use bars (length) to present numeric data and ticks (position) to present ordinal data. When users zoom out and the text cannot be clearly presented, we use barcodes (shape) to present categorical data (Figure 6). We do not use colors to encode categorical data primarily because colors were already occupied, and colors can hardly deal with a large number of categories. The advantage of barcodes is that they are scalable both in quantity (i.e., we can easily design a set of barcodes to encode thousands of categories) and in space (i.e., the height of a barcode can be compressed to 1 pixel). In addition, when multiple same barcodes are vertically juxtaposed, they can form a whole in space. This feature help users distinguish groups of identical items and identify whether a group of data items encoded in barcode are the same (Figure 6). The drawback of barcodes is that they are difficult for users to remember and search, and thus it is difficult for identifying or comparing items across views. Considering this drawback, we only use barcodes when the text is not legible.

Interaction In the detail view, the user may zoom a table to control the number of rows displayed. We support sorting and filtering (Figure 5 A), two common tabular data operations. When users perform column-level DLT, the columns displayed by default are consistent with the traced columns in the lineage view, to help users focus on the traced columns. The core functionality we support in this view is instance-level DLT. When users click a row, we will add links to the rows used to generate the row or cell in the previous table and highlight them through changing the background color to gray. Besides, users can focus on the traced rows by hiding the irrelevant data and display the columns needed for context through interaction (Figure 5 B). Users are allowed to trace data across multiple steps in a skipping manner. Rows that are not displayed are indicated by numbers and direction arrows (Figure 5 C).

4.3 Justification for Multi-View Design Choice

In our work, users view the DT process and trace column-level DL in the lineage view, and trace fine-grained DL and examine data details in the detail view. In addition to the current approach, there are two design alternatives.

One approach is to use a single detail view, where users are initially presented with data tables and encode the information from the lineage view into the detail view. The problem with this design is that it can easily overwhelm users with too much information at once, leading to information overload and violating the mantra for visual information seeking by Shneiderman [6].

Another design choice is to allow users to combine Focus + Context in the lineage view, directly expanding the tables and tracing the details. However, we did not adopt this design primarily due to the following considerations. First, DLT is a highly iterative process, and users require a stable lineage view. After completing a fine-grained DLT task, users need to return to the coarse-grained pipeline to find the next target table or column for tracing. Users require a stable lineage view as the context for completing coarse-grained DLT tasks, and adopting the Focus + Context approach makes it challenging to maintain the stability of the context since the expanded tables require considerable space. Second, users often jump during the process of DLT. They do not have to examine DT operations and intermediate tables one by one. Instead, they often jump several steps to check an operation or table they suspect, based on their experience

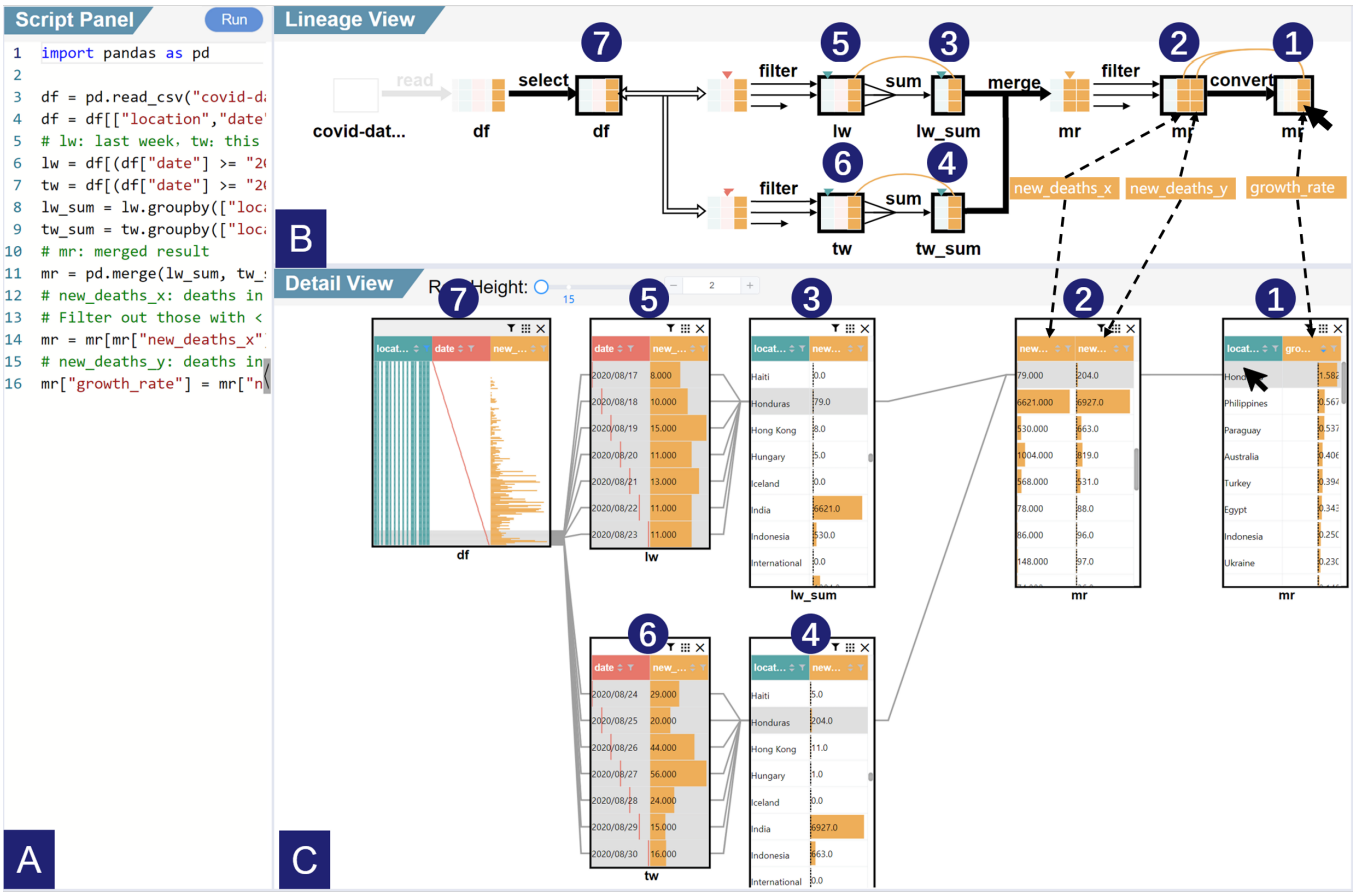

Fig. 7. HYPNOS presents the DL extracted from a data transformation script for analyzing the trend of Coronavirus deaths at different levels of detail. The interface consists of (A) a script panel which shows the script, and two visualization components: (B) a lineage view for presenting the data pipeline, which supports table-level and column-level DLT; (C) a detail view for presenting the intermediate tables in the data pipeline, which supports row-level DLT and allows users to inspect details.

or intuitiveness. To display the fine-grained lineage between two tables that are several steps apart, it is necessary to hide the intermediate tables and operations. However, this can disrupt the contextual information in the lineage view.

Our current approach ensures the stability and independence of the contexts required for both coarse-grained and fine-grained tasks, preventing them from interfering with each other. The drawback is that both views require space, and it requires some effort to connect the information between the two views.

4.4 Implementation

HYPNOS is implemented as a web application with a Python server. The backend server is built with Flask [43]. The program adaptor is from SOMNUS [4]. The frontend is built in Vue.js [44]. In the script panel, we utilize Monaco Editor [45] to present the code. In the lineage view, we use Eclipse Layout Kernel [46] for generating node positions and D3.js [47] for drawing visual elements. In the detail view, we use vxe-table [48] for displaying tabular data.

5 Evaluation

In this section, we present a case study, an expert interview, and a user study to illustrate the usability and effectiveness of HYPNOS. Due to the limitation of space, additional cases and details are placed in the supplementary materials.

5.1 Use Case: COVID Analysis

A data analyst analyzes the trend of COVID-19 in various countries and locations. He uses HYPNOS to help him verify an abnormal result derived from his data processing script.

The script for weekly analyzing the trend of COVID-19 deaths in various countries and regions around the world. After the output table is used in a downstream data analysis task, his collaborators ask him to check one particular record in the data: *growth_rate* (of weekly deaths) of Honduras is significantly larger than that of other countries (Figure 7 C1). The collaborators want him to validate his data processing and explain to them how the record is generated.

In order to solve the concerns of the collaborators, the user needs to verify the data. Traditionally, he would briefly review the data pipeline and then figure out how *growth_rate* is generated in the code. He would check the key DT operations to ensure there are no errors and examine data details by displaying intermediate results, such as executing "print(df)" after specific operations. With HYPNOS, he could do this in a more intuitive way. He uploads the data and script to HYPNOS and clicks Run button. The system then generates a lineage graph (Figure 7 B) that represents the data pipeline. He briefly reviews the pipeline by reading the lineage graph from left to right to see whether there is any obvious mistakes. The script first creates a table (*df*) by

reading an external file, and selects three columns (see the leftmost two steps in Figure B). Then, it performs two *filter* operations on *df* and gets two different output tables (*lw* and *tw*). Afterwards, it performs *sum* on each of the two tables and gets *lw_sum* and *tw_sum*. Then, it merges the two output tables to get *mr*. After that, it performs a filter operation on *mr* and finally combines two columns to get the output.

After reviewing the pipeline, he traces the DL of *growth_rate* (Figure 8 A) to focus on how it is derived. Hovering on the edge of the last operation (Figure 8 A), he finds that *growth_rate* is generated from *new_deaths_x* / *new_deaths_y* − 1. He further traces *new_deaths_x* and *new_deaths_y* to see how they are derived (Figure 8 B and C). He could see that both *new_deaths_x* and *new_deaths_y* have gone through a *filter* operation and then a *sum* operation. He decides to check the two *filter* operations since they are critical and prone to errors. By hovering on the edges of the *filter* operations, he checks them and makes sure that both filter conditions are correct.

After checking the overall pipeline and the critical DT operations, the user starts to check the specific data. Note that in this case the user performs some column selections, which are omitted for the simplicity of illustration. He exposes *mr* (Figure 7 B1) and sorts it according to *growth_rate* (Figure 7 C1). Then, he finds that the *growth_rate* of Honduras is indeed significantly greater than that of other countries. To know how this value is derived, he exposes its input table (Figure 7 B2) and trace the row-level data lineage (Figure 7 C2). Bob finds that the *new_deaths_x* and *new_deaths_y* of Honduras are 79 and 204 respectively. Unlike those countries which have no more than 10 deaths in the last week, 79 is quite a normal number for a country like Honduras. He cannot conclude that the extremely large *growth_rate* of Honduras is due to a small denominator. In order to know how these two values are derived, he exposes *lw_sum* (Figure 7 B3), *tw_sum* (Figure 7 B4), *lw* (Figure 7 B5) and *tw* (Figure 7 B6). He then clicks on the row containing Honduras in *mr* and traces the row-level lineage in the other tables. Since it is not necessary to compare the daily new deaths of Honduras to that of the other countries, he only keeps the traced data in Figure 7 C5 and C6. Then, he checks the daily new deaths of Honduras in the last week (Figure 7 C5) and this week (Figure 7 C6). He finds that *new_deaths* of Honduras in 2020/08/26 and 2020/08/27 are comparatively large. In order to see the trend of Honduras in a longer period, he exposes *df* (Figure 7 B7), filters out the other countries, and scales the table (Figure 7 C7). From Figure 7 C7, he ensures that these two values are normal according to the history of Honduras. After figuring out why the *growth_rate* of Honduras is large, he makes an explanation to his collaborators. He provides the snapshot of the detail view as evidence to support his explanation.

In general, the data analyst benefits from:

- An intuitive overview of the script (C1), which helps them review the data pipeline (T1) and provides the context for schema-level DLT tasks (T3).
- Clear representations of DT semantic (C2), which helps them locate and check interested DT operations (T2).
- Explicit revelation of different levels of DL (C3), which helps them trace DL (T3) and focus on key imformation.
- Convenient access to the tables and traced data items (C4), which helps them inspect data details (T4).

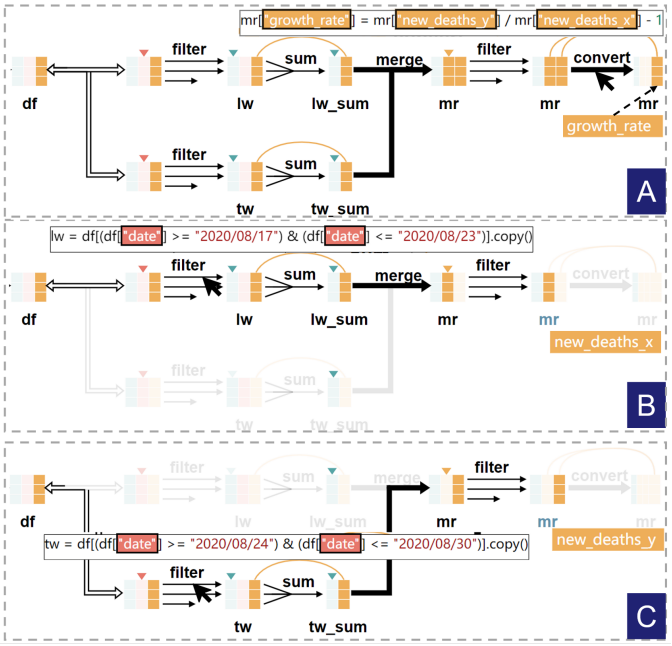

Fig. 8. Column tracing of growth_rate (A), new_deaths_x (B), and new_deaths_y (C). Hovering on the edges, users can see the code while highlighting the critical columns.

5.2 Expert Interview

To test the usability and effectiveness of our system, we interviewed four data analysts with more than five years of experience: two data analysts (E1 and E2) working in a research institute and a company, respectively; one postdoctoral researcher (E3) and one research fellow (E4), both in the field of data science. Among them, E1 and E2 had collaborated with us in several data analysis project (but were not co-authors of the paper) and used HYPNOS to complete cases. E3 and E4 were external experts who were not familiar with HYPNOS. To allow them better evaluate our system, we introduced HYPNOS and three alternatives: Jupyter Notebook [49], HEX [50], and SOMNUS [4]. Jupyter Notebook is a commonly used IDE and a regular baseline. HEX provides functionality similar to Jupyter Notebook, while also providing a code-based lineage graph feature. SOMNUS is a visualization system that provides a glyph-based lineage graph. Snapshots of these tools can be found in the supplemental materials.

We first introduced to the experts the background of our research and the scenario we are focusing on: DLT for post-hoc data validation. Subsequently, for each of the tool, we demonstrated the use of the tool through an identical case. Then we provided the experts with several scripts and datasets, allowing them to freely explore and experience the tools. After they had used all of the tools, we conducted a semi-structured interview with them. We mainly focus on the effectiveness of HYPNOS in presenting the data flow (C1) and the semantics of data transformations (C2), and revealing data lineage (C3) and traced data details (C4). The entire process lasted approximately one and a half hours.

Visualization. All four experts agreed that 1) glyph-based methods (SOMNUS and HYPNOS) are more intuitive than code-based methods (Jupyter Notebook and HEX) in show-

TABLE 2
The comparison of Jupyter Notebook, HEX, SOMNUS, and HYPNOS.

System	Lineage Graph	Table Lineage	Column/Row Lineage
Jupyter	None	No	No
HEX	Code-Based	Yes	No
SOMNUS	Glyph-Based	Yes	No
HYPNOs	Glyph-Based	Yes	Yes

ing the data pipeline; 2) the lineage graph of HYPNOS is most intuitive and helpful for understanding data lineage. Furthermore, experts commented that "HYPNOS is more intuitive than SOMNUS" due to the consistent color encoding (E1, E2, and E4) and conciseness (E2 and E3). The following comments from E2 illustrate his need and why he believed that HYPNOS is more suitable for DL scenarios:

> *Reviewing an entire script and understanding its logic is not easy, but understanding a single line of code is not difficult. The focus (of glyphs) should not be on explaining a single line of code, but rather on helping locate the data and critical code. ... The lineage graph in HYPNOS is more concise and highlights the key points better. The colors cause no confusion. It more suitable for data lineage scenarios.*

Lineage Tracing. All four data analysts acknowledged the effectiveness of interactive DLT functionalities in HYPNOS, which simplify the DLT tasks (E1-3). Coarse-grained DLT helps them locate/focus on key data and DT operations (E1-4) and understand the context (E1 and E3). Fine-grained DLT provides convenient data retrieval (E1, E2, and E4) and intuitive data relationships (E1 and E3). We cite E1's comments to show the effectiveness of HYPNOS:

> *Interactive data lineage tracing is indeed helpful. Coarse-grained tracing is valuable for understanding the broader context (beyond a single step). Fine-grained tracing provides convenient data retrieval and intuitive relationships between different data items. This level of detail facilitates precise troubleshooting, as I can trace the lineage of specific data items across various stages.*

Reviewing Code. HYPNOS supports displaying the code of a DT operation in the lineage view and detail view. In scenarios where a user's focus is to trace the lineage of specific data items, this feature maybe sufficient. However, when the user's focus is on checking and debugging a snippet of code rather than tracing data, our approach may not be as effective (E3 and E4). We cite E3's comments to illustrate the scenarios where HYPNOS fits and where it falls short:

> *Displaying the code (for a particular step) in the graph is a nice feature, as it avoids switching between different views. It may be sufficient for such cases, but when I find numerous issues within the data, my task is no longer to trace specific data items. Instead, I want to thoroughly check the code. I would need to look at both the code and the lineage graph, switching between them, which may not be convenient.*

Data Details. In the detail view, we provided some basic functionalities (sorting and filtering) for the expanded tables, which are sufficient in many scenarios (E1-3). However, there is a desire for more advanced features, such as expanding the table in a new window, allowing users to perform further operations using code, or providing profiling functionalities. The following comments from E4 show the desire for the support of further analysis:

> *In some cases, checking the data alone may not be enough to make a judgment. Further analyses, such as examining the data distribution, may be necessary to better determine if a value is an anomaly. I hope that it (HYPNOS) could offer additional support for these analyses. For example, it could allow users to perform further analysis through coding.*

5.3 User Study: HYPNOS vs SOMNUS

This study compares the effectivenss of HYPNOS and SOMNUS in the DLT scenario. Since SOMNUS does not support instance-level DL and DLT-related interactions, we focused on the comparison of different glyph designs and the common visualization component (the lineage view in HYPNOS and the graph panel in SOMNUS). To make a fair comparison, 1) we disabled the tracing functions in HYPNOS which make the DLT tasks too straightforward; and 2) we did not include tasks that require fine-grained information (T4) since the lineage view of HYPNOS and the graph panel in SOMNUS are not designed for these tasks.

5.3.1 Participants and Experiment Design

Participants. We recruited 12 participants (7 males and 5 females, aged 23 to 29, and all reported normal or correct-to-normal vision) in this study. 10 of them were graduate students who majored in Data Science (6) or Artificial Intelligence (4). 2 of them were full-time data analysts who worked in a research institution. They all reported to have experience in using Pandas for processing data (2 reported 0-0.5 year, 4 reported 0.5-1 year, 4 reported 1-3 years, and 2 reported more than 3 years). All reported normal or correct-to-normal vision. Each of the participants received 50 Chinese Yuan as a reward.

Tasks and Materials. We adopted a within-subject design to test how HYPNOS and SOMNUS help users in summarizing the data pipeline (T1), checking DT semantics (T2), and tracing columns (T3). According to whether participants need to pay attention to multiple steps of DTs or a single step of DT, we characterized these tasks as single-step tasks (T2) and multi-step tasks (T1 and T3). In order to balance the learning and sequence effects, we constructed two sets of tasks, each includes 7 single-step (semantic) tasks and 3 multi-step tasks (1 summarization task and 2 tracing tasks). To craft the single-step tasks, we designed 14 questions based on the glyphs generated by the two systems. These questions cover all 15 (5 *Operation Classes* $*$ 3 *Data Types*) of DT types characterized by Kasica et al. [24] except *Transform Rows*, which requires fine-grained information to answer the related questions. To craft the multi-step tasks, we used two datasets, one used in the usage scenario of the present paper and one from SOMNUS. The details of these tasks can be found in supplemental materials. We denote the two task sets as TS1 and TS2 and the two systems as H and S. The participants were divided into four groups and each group was assigned one of the following conditions: [TS1H, TS2S], [TS1S, TS2H], [TS2H, TS2S], [TS2S, TS2H].

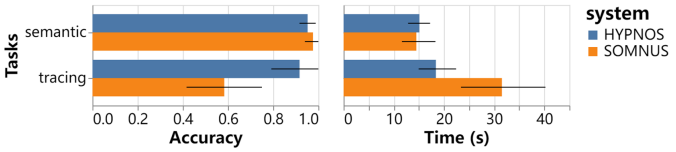

Fig. 9. Average accuracies and time with 95% confidence intervals.

Procedure. The experiment can be divided into three phases: test of one system, test of the other system, the questionnaire and the interview. The procedure of the test of one system is as follows: [Introduction of the experiment $->$ Introduction of the glyph design $->$ Training $->$ Single-step Tasks $->$ Introduction of the system $->$ Training $->$ Multi-step Tasks]. After they finished the two test phases, they were asked to rate on the two systems through a questionnaire (Table 3). The questionnaire is a seven-point Likert scale (1 - strongly disagree, 7 - strongly agree). In the process of rating, they were encouraged to explore these two systems to better evaluate them. Finally, we made a short interview with each participant to collect their feedbacks. We recorded the answers and the time for further analysis. The entire experiment lasted about 45 minutes.

5.3.2 Result

The average accuracies and time are presented in Figure 9. We did not analyze the accuracies and time for summarizing the pipeline (T1) since we found that many participants are not adept at describing DT procedures (even though they might have a clear understanding internally). We used Student's t test to examine the result after ensuring the homogeneity of variance is not violated. In terms of semantic checking tasks (T2), there is no significant difference between the average accuracies ($t = 0.920$, $p = 0.368$) and time ($t = 0.272$, $p = 0.788$) of the two methods. In terms of lineage tracing tasks, the average accuracy of HYPNOS is significantly higher than that of SOMNUS ($t = 3.316$, $p < 0.01$) while the average time of HYPNOS is significantly less than that of SOMNUS ($t = 2.646$, $p < 0.05$). This indicates that HYPNOS is close to SOMNUS in presenting the semantics of a DT operation while it is superior to SOMNUS in presenting DL. The result is consistent with our observation. During the experiment, a number of participants could quickly locate the key DT operations with HYPNOS to answer multi-step questions while they seemed to be struggling with SOMNUS.

The ratings are presented in Figure 10. HYPNOS is rated higher on Q1 (summarizing pipeline) and Q3 (tracing lineage) and has a similar rating with SOMNUS on Q2 (presenting semantics). The ratings on Q2 and Q3 are aligned with the performance of semantic tasks and tracing tasks, which indicates that HYPNOS is similar to SOMNUS in presenting DT semantics and the superior of HYPNOS in DLT scenarios.

5.3.3 Feedback

From user feedback, we identified a series of commonly used keywords and summarized three main dimensions compare and evaluate two systems: consistency, simplicity, and intuitiveness. We further categorized the participants' statements into positive and negative comments of two systems across these dimensions.

TABLE 3
Questionnaire.

Q1	The system is helpful in summarizing data transformation pipeline.
Q2	The visual design well presents the semantics of data transformations.
Q3	The system is helpful in tracing data lineage.

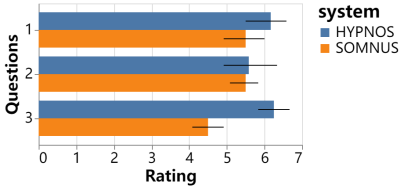

Fig. 10. Average ratings with 95% confidence intervals.

Consistency. 9 (of the 12) participants gave positive comments about the consistency of HYPNOS (7) or gave negative comments about that in SOMNUS (4). Among them, 6 participants mentioned the consistent colors in HYPNOS (6) or the inconsistent colors in SOMNUS (3). 4 participants mentioned the inconsistent visual representations of data objects in SOMNUS (e.g., "It is confusing that the same table looks so different before and after").

Simplicity. 6 participants gave positive comments about the simplicity of HYPNOS (4) or gave negative comments about that in SOMNUS (6). Among them, 4 participants commented that there are "too many visual elements" or "too many details" in SOMNUS; 2 participants commented that the details in SOMNUS are "hardly available" or "difficult to read". 2 participants mentioned that the glyph designs of both system are simple. 1 participant commented that the glyph design of SOMNUS is simpler and easier to learner.

Intuitiveness. 6 participants mentioned that HYPNOS is more intuitive or SOMNUS is less intuitive. Among them, 5 participants attributed the differences to color consistency (5), the amount of visual elements or details (3), and the graph design (2). 3 participants mentioned that the glyph designs of both systems are intuitive. In addition, 1 participant believe that SOMNUS is more intuitive than HYPNOS. The participant commented "The design of first system (SOMNUS) is more intuitive. I did not need to learn it at all."

6 Discussion

HYPNOS distinguishes itself from existing work primarily through its visualization and interface design. In terms of visualization design, we have adopted a glyph design that are more suited for DLT scenarios. We employ a consistent color scheme and utilize arrows and connections to represent the relationships between data. Meanwhile, we have strived to simplified the design while highlighting key DT operations according to the needs of data analysts. In terms of interface design, we have adopted two separate but coordinated views for supporting schema-level and instance-level DLT. This design primarily takes into account that users will alternate between scheme-level and instance-level DLT tasks, and having two views provides the context needed for each task without interference. HYPNOS offers users an

intuitive representation of data pipeline and DT semantics, and provides interactive DLT functions, allowing users to present data details as needed. HYPNOS provides users with the context necessary for completing DLT tasks, helping them focus on key DT operations while intuitively understanding the relationships between data items. In the following subsections, we further discuss 1) the implications obtained from the iterative process of designing, the observation of our users, and their feedbacks; 2) the limitations.

6.1 Implications

Keep the Overview Simple to Achieve Preferable Design. Initially, we intended to provide an overview with more detailed information such as the sizes of the tables and the parameters of the DT operations. However, during the iterative discussions with the two data analysts involved in our study, we realized that more details might lead to information overload and a concise overview is preferable. For this reason, we have decided to simplify the design of the lineage view while providing users with useful information and interactive features. The feedback from users confirms the preference and benefits of a concise design, which highlighted the importance of simplicity in overview designs.

Support Navigation in a Skipping Manner to Improve Efficiency. As mentioned previously, data analysts do not necessarily inspect DT operations and intermediate tables sequentially. Instead, they frequently skip multiple steps to directly navigate to the key information most relevant to their interest. To support such navigation, we design a concise lineage graph with consistent color encoding, which highlights the critical DT operations they may be interested in. In the user study, we observed that users were able to locate key information in a skipping manner with HYPNOS. The findings from our study indicates that supporting navigation in a skipping manner helps improve the efficiency of visual analysis.

Keep the Context Stable to Avoid Context Loss. During our collaboration with data analysts, we observed that they often encountered the issue of losing context while using Jupyter Notebook due to context changes. For instance, after locating a a table, a data analyst needed to view its details. However, exposing data details in Jupyter Notebook would cause the position of the code blocks to change. This alteration disrupted his previous working context. To address this issue, we ensure the stability of the contexts required for both locating and inspection tasks. We suggest that the context loss due to the unstable context can impose unnecessary load, which should be avoided.

Use Color Encoding Appropriately to Ensure Effectiveness. In HYPNOS, we use colors to indicate column-level DL. In those uncomplicated cases, users can even complete column-level DLT tasks without the interaction of tracing columns, which shows the effectiveness of this color encoding method. However, the space of color for clearly encoding categorical data is quite limited. When there are numerous columns, colors can indeed become easily confusing. To address this issue, we provide a focus mode for coloring that only assign colors to the traced columns to address this issue. Our research suggests that it is important to prepare alternative solutions (e.g., interaction) to address color conflicts and ensure the effectiveness of color.

6.2 Limitations

Code Parsing. The program adaptor of SOMNUS [4] is customized for several popular Python and R packages (e.g., Pandas and tidyr). Since we used this program adaptor, HYPNOS inherits the limitations of SOMNUS in terms of code parsing. We do not support other programming languages (e.g., SQL) and syntaxes unrelated to table transforms (e.g., *if ... else ...*). To improve the generalizability and the capability, we would explore more advanced techniques (e.g., large language models) to enhance the ability in parsing code.

Scalability. The scalability of HYPNOS is limited by the lineage module as well as the visual design. First, the current lineage module can hardly support big data scenarios. It entails caching all the intermediate tables in the pipeline. When the volume of data exceeds the memory capacity, it necessitates fetching data from external storage, which consequently incurs significant latency. Second, our visual design is hardly applicable to wide tables and high dimensional data. To ensure visibility, the width of the columns should not be too narrow. Based on our current settings (minimum width of 8 pixels and 1 pixel offset), a maximum of approximately 400 columns can be displayed horizontally in the lineage view on a 4K monitor. The amount of data that can be displayed in the detail view is smaller by one order of magnitude.

To make HYPNOS more scalable, we plan to explore more scalable lineage modules and visual designs in the future. An interesting idea is to integrate multi-step DT operations into one step to facilitate both storage and rendering. More detailed discussion can be found in supplemental materials.

Further Analysis on the Traced Data. HYPNOS provides a detail view to allow users to inspect table details in a spreadsheet style. It can help users check data in a single dimension, or combine category and time information to do some simple analysis. The it does not support post-processing procedures (e.g., anomaly detection algorithms or plotting), and users can only export the traced data and rely on external tools for further analysis. Presently, we have not yet summarized the diversified requirements of data analysts from a wide range of cases. We look forward to completing this in the future.

Dependence on Interactions. In our work, we use a series of interactions to help users trace DL and inspect details. The effectiveness of our visual design significantly relies on these interactions. Our lineage graph can be applied in an interactive visualization system for exploring data pipeline. However, it is probably not effective for illustrating the DL of columns in a statistic image. In the future, we would provide more efficient interactions to make HYPNOS easier to use. Besides, we would explore visual designs that less rely on interactions, such as using labels and additional links.

7 Conclusion

We present HYPNOS, a visualization system that supports interactive DLT for DT scripts. HYPNOS consists of a lineage module and a visualization interface. The lineage module employs a program adaptor to extract the key information from code and uses a lineage tracker to capture both schema-level and instance-level DL. The visualization interface provides users with an overview of the data pipeline by a lineage view and supports fine-grained data lineage tracing

by a detail view. We demonstrate the usability of HYPNOS through a use case, expert interviews, and a user study.

In the future, we plan to extend HYPNOS by employing an advanced lineage module that supports a wider range of DTs in different programming languages and captures DL more efficiently. Besides, we would explore visual designs that effectively visualize the DL of wide tables and rely less on interactions. Moreover, we attempt to provide means for supporting further analysis of the traced data.

ACKNOWLEDGMENTS

The work was supported by National Key R&D Program of China (2022YFE0137800), Key "Pioneer" R&D Projects of Zhejiang Province (2023C01120), and NSFC (U22A2032, 62402421).

REFERENCES

[1] R. Ikeda and J. Widom, "Data lineage: A survey," Stanford InfoLab, Tech. Rep., 2009.
[2] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "Vistrails: visualization meets data management," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2006, pp. 745–747.
[3] R. Hoekstra and P. Groth, "Prov-o-viz-understanding the role of activities in provenance," in *International Provenance and Annotation Workshop*. Springer, 2014, pp. 215–220.
[4] K. Xiong, S. Fu, G. Ding, Z. Luo, R. Yu, W. Chen, H. Bao, and Y. Wu, "Visualizing the scripts of data wrangling with SOMNUS," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2022.
[5] X. Pu, S. Kross, J. M. Hofman, and D. G. Goldstein, "Datamations: Animated explanations of data analysis pipelines," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–14.
[6] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *The Craft of Information Visualization*. Elsevier, 2003, pp. 364–371.
[7] Y. Chen, Y. Zhao, X. Li, J. Zhang, J. Long, and F. Zhou, "An open dataset of data lineage graphs for data governance research," *Visual Informatics*, vol. 8, no. 1, pp. 1–5, 2024.
[8] H. Stitz, S. Luger, M. Streit, and N. Gehlenborg, "Avocado: Visualization of workflow-derived data provenance for reproducible biomedical research," in *Computer Graphics Forum*, vol. 35, no. 3. Wiley Online Library, 2016, pp. 481–490.
[9] M. Yu, Y. Wang, X. Yu, G. Shan, and Z. Jin, "Pubexplorer: An interactive analytical system for visualizing publication data," *Visual Informatics*, vol. 7, no. 3, pp. 65–74, 2023.
[10] M. Tian, G. Li, and X. Yuan, "Litvis: a visual analytics approach for managing and exploring literature," *Journal of Visualization*, vol. 26, no. 6, pp. 1445–1458, 2023.
[11] H. Lv, Z. Niu, W. Han, and X. Li, "Can gpt embeddings enhance visual exploration of literature datasets? a case study on isostatic pressing research," *Journal of Visualization*, pp. 1–14, 2024.
[12] R. Pan, Y. Wang, J. Sun, H. Liu, Y. Zhao, J. Xia, and W. Chen, "Simplifying social networks via triangle-based cohesive subgraphs," *Visual Informatics*, vol. 7, no. 4, pp. 84–94, 2023.
[13] J. Klaus, M. Blacher, A. Goral, P. Lucas, and J. Giesen, "A visual analytics workflow for probabilistic modeling," *Visual Informatics*, vol. 7, no. 2, pp. 72–84, 2023.
[14] W. Gao, S. Liu, Y. Zhou, F. Wang, F. Zhou, and M. Zhu, "Gbdt4ctrvis: visual analytics of gradient boosting decision tree for advertisement click-through rate prediction," *Journal of Visualization*, pp. 1–21, 2024.
[15] O. Akbulut, L. McLaughlin, T. Xin, M. Forshaw, and N. S. Holliman, "Visualizing ordered bivariate data on node-link diagrams," *Visual Informatics*, vol. 7, no. 3, pp. 22–36, 2023.
[16] Y.-H. Kuo, D. Liu, and K.-L. Ma, "Spreadline: Visualizing egocentric dynamic influence," *IEEE Transactions on Visualization and Computer Graphics*, 2024.
[17] M. A. Borkin, C. S. Yeh, M. Boyd, P. Macko, K. Z. Gajos, M. Seltzer, and H. Pfister, "Evaluation of filesystem provenance visualization tools," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 2476–2485, 2013.
[18] P. Alvaro, J. Rosen, and J. M. Hellerstein, "Lineage-driven fault injection," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015, pp. 331–346.
[19] C. Niederer, H. Stitz, R. Hourieh, F. Grassinger, W. Aigner, and M. Streit, "TACO: visualizing changes in tables over time," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 677–686, 2018.
[20] Z. Luo, K. Xiong, J. Zhu, R. Chen, X. Shu, D. Weng, and Y. Wu, "Ferry: Toward better understanding of input/output space for data wrangling scripts," *IEEE Transactions on Visualization and Computer Graphics*, 2024.
[21] K. Xiong, X. Xu, S. Fu, D. Weng, Y. Wang, and Y. Wu, "Jsoncurer: Data quality management for json based on an aggregated schema," *IEEE Transactions on Visualization and Computer Graphics*, 2024.
[22] A. Coscia, A. Suh, R. Chang, and A. Endert, "Preliminary guidelines for combining data integration and visual data analysis," *IEEE Transactions on Visualization and Computer Graphics*, 2023.
[23] X. Cai, D. Weng, T. Fu, S. Fu, Y. Wang, and Y. Wu, "Linking text and visualizations via contextual knowledge graph," *IEEE Transactions on Visualization and Computer Graphics*, 2024.
[24] S. Kasica, C. Berret, and T. Munzner, "Table Scraps: An actionable framework for multi-table data wrangling from an artifact study of computational journalism," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 957–966, 2021.
[25] G.-C. Roman and K. C. Cox, "Program visualization: The art of mapping programs to pictures," in *Proceedings of the International Conference on Software Engineering*, 1992, pp. 412–420.
[26] K. E. Isaacs, A. Giménez, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, and P.-T. Bremer, "State of the art of performance visualization." *EuroVis (STARs)*, vol. 3, p. 6, 2014.
[27] J. Sundararaman and G. Back, "Hdpv: Interactive, faithful, in-vivo runtime state visualization for c/c++ and java," in *Proceedings of the ACM Symposium on Software Visualization*, 2008, pp. 47–56.
[28] R. Faust, K. Isaacs, W. Z. Bernstein, M. Sharp, and C. Scheidegger, "Anteater: Interactive visualization of program execution values in context," *arXiv preprint arXiv:1907.02872*, 2019.
[29] E. E. Aftandilian, S. Kelley, C. Gramazio, N. Ricci, S. L. Su, and S. Z. Guyer, "Heapviz: Interactive heap visualization for program understanding and debugging," in *Proceedings of the International Symposium on Software Visualization*, 2010, pp. 53–62.
[30] R. Schulz, F. Beck, J. W. C. Felipez, and A. Bergel, "Visually exploring bbject mutation," in *IEEE Working Conference on Software Visualization*, 2016, pp. 21–25.
[31] T. Ohmann, R. Stanley, I. Beschastnikh, and Y. Brun, "Visually reasoning about system and resource behavior," in *Proceedings of the International Conference on Software Engineering Companion*, 2016, pp. 601–604.
[32] H. Lieberman and C. Fry, "Bridging the gulf between code and behavior in programming," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1995, pp. 480–486.
[33] S. P. Reiss, "The challenge of helping the programmer during debugging," in *IEEE Working Conference on Software Visualization*, 2014, pp. 112–116.
[34] J. Hoffswell, A. Satyanarayan, and J. Heer, "Augmenting code with in situ visualizations to aid program understanding," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.
[35] F. Beck, O. Moseler, S. Diehl, and G. D. Rey, "In situ understanding of performance bottlenecks through visually augmented code," in *Proceedings of International Conference on Program Comprehension*, 2013, pp. 63–72.
[36] M. Harward, W. Irwin, and N. Churcher, "In situ software visualisation," in *Proceedings of Australian Software Engineering Conference*, 2010, pp. 171–180.
[37] B. Swift, A. Sorensen, H. Gardner, and J. Hosking, "Visual code annotations for cyberphysical programming," in *Proceedings of International Workshop on Live Programming*, 2013, pp. 27–30.
[38] A. J. Ko and B. A. Myers, "Designing the whyline: A debugging interface for asking questions about program behavior," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 151–158.
[39] T. Lieber, J. R. Brandt, and R. C. Miller, "Addressing misconceptions about code with always-on programming visualizations," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 2481–2490.
[40] S. Oney and B. Myers, "FireCrystal: Understanding interactive behaviors in dynamic web pages," in *Proceedings of IEEE Symposium*

on Visual Languages and Human-Centric Computing, 2009, pp. 105–108.
[41] B. Burg, R. Bailey, A. J. Ko, and M. D. Ernst, "Interactive record/replay for web application debugging," in *Proceedings of Annual ACM Symposium on User Interface Software and Technology*, 2013, pp. 473–484.
[42] F. Psallidas and E. Wu, "Smoke: Fine-grained lineage at interactive speed," *Proceedings of the VLDB Endowment*, vol. 11, no. 6, pp. 719–732, 2018.
[43] Flask, http://flask.pocoo.org/.
[44] Vue.js, https://vuejs.org/.
[45] Monaco Editor, https://microsoft.github.io/monaco-editor/.
[46] Eclipse Layout Kernel, https://www.eclipse.org/elk/.
[47] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.
[48] vxe-table, https://github.com/x-extends/vxe-table.
[49] "Jupyter Notebook," https://jupyter.org, accessed: Feb 28, 2024.
[50] "Hex," https://hex.tech, accessed: Feb 28, 2024.

Xiwen Cai is currently a Ph.D. candidate in the State Key Lab of CAD&CG, Zhejiang University in China. He received his B.Sc. degree in Psychology and M.Sc degree in Computer Science from Zhejiang University. His research interests include human computer interaction and visual analytics.

Xiaodong Ge received the master's degree in computer science from Zhejiang University, China. He is currently working in Zhejiang Lab, China. His research interests include visualization and visual analysis.

Kai Xiong is a Ph.D. student at the State Key Laboratory of CAD&CG, Zhejiang University, and works under the supervision of Prof. Yingcai Wu. He holds a bachelor's degree in Computer Science from Xidian University. His main research interests center on visual analytics and data wrangling. For more information, please visit https://xkkevin.github.io/.

Dr. Di Weng is a ZJU100 Young Professor at School of Software Technology, Zhejiang University. His main research interest lies in information visualization and visual analytics, focusing on interactive data transformation and spatiotemporal data analysis. He received his Ph.D. degree in Computer Science from State Key Lab of CAD&CG, Zhejiang University. Prior to his current position, Dr. Weng was a researcher at Microsoft Research Asia from 2022 to 2023. For more information, please visit https://dwe.ng.

Shuainan Ye received his Ph.D. degree in Computer Science from Zhejiang University, China in 2022. He works as a postdoctoral researcher in the State Key Lab of CAD&CG, Zhejiang University. His main research interests include data visualization, human-computer interaction, and immersive analytics.

Ke Xu is currently a tenure-track associate professor at Nanjing University. Prior to this, he was an scientist in the Data Intelligence Lab at HUAWEI Cloud and a postdoctoral research associate in the Visualization and Data Analytics Research Center (VIDA) at New York University (NYU). He obtained his Ph.D. in the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology in 2019, and B.S. in Electronic Science and Engineering from Nanjing University, China in 2015. His research interests include data visualization, human-computer interaction, with focus on visual anomaly detection and explainable AI.

Datong Wei is currently a researcher in the Data Intelligence Innovation Lab at HUAWEI Cloud. He received his B.Sc. degree in Intelligence Science and Technology and the M.Sc. degree in Software Engineering from Peking University. His research interests include data visualization and human-computer interaction.

Jiang Long is a Ph.D. from ZheJiang University and a technical expert at Huawei. Since 2005, he has worked in Huawei, Alibaba, and Tencent. He has served as the chief architect of Huawei cloud data platform and head of Huawei data intelligent innovation lab. He has also served as a senior product expert of Ali Group, senior operation expert of Alipay and Tencent data center O&M expert.

Dr. Yingcai Wu is a Professor at the State Key Lab of CAD&CG, Zhejiang University. His main research interests are information visualization and visual analytics, with focuses on urban computing, sports science, immersive visualization, and social media analysis. He received his Ph.D. degree in Computer Science from the Hong Kong University of Science and Technology. Prior to his current position, Dr. Wu was a postdoctoral researcher in the University of California, Davis from 2010 to 2012, and a researcher in Microsoft Research Asia from 2012 to 2015. For more information, please visit http://www.ycwu.org.