Urania : An Intelligent Authoring Tool for Creating Datamations via Data Query Decomposition

Yi Guo, Nan Cao, Ligan Cai, Yanqiu Wu, Daniel Weiskopf, Danqing Shi and Qing Chen

***Abstract*—Datamation is designed to animate an analysis pipeline step by step, which is an intuitive and effective way to interpret the results from data analysis. However, creating a datamation is not easy. A qualified datamation needs to not only provide a correct analysis result but also ensure that the data flow and animation are coherent. Existing animation authoring tools focus on either leveraging algorithms to automatically generate an animation based on user-provided charts or building graphical user interfaces to provide a programming-free authoring environment for users. None of them are able to help users translate an analysis task into a series of data operations to form an analysis pipeline and visualize them as a datamation. To fill this gap, we introduce Urania , an intelligent authoring tool developed to support datamation design and generation. It leverages a novel data query decomposition model to allow users to generate an initial datamation by simply inputting a data query in natural language. The initial datamation can be refined via rich interactions and a feedback mechanism is utilized to update the decomposition model based on user knowledge and preferences. Our system produces an animated sequence of visualizations driven by a set of low-level data actions. It supports unit visualizations, which provide a mapping from each data item to a unique visual mark. We demonstrate the effectiveness of Urania via a series of evaluations including case studies, performance validation, and a controlled user study.**

***Index Terms*—Natural Language Interface, Datamation**

I. Introduction

A datamation [1] is designed to help interpret the results from data analysis by animating the detailed analysis pipeline step by step. Although it has been demonstrated to be an intuitive and effective way of interpreting the analysis, creating a datamation is not easy. Users need to decompose a complex data analysis task into a sequence of fundamental data operations and represent the intermediate results via appropriate visualizations that could be smoothly connected through animations. To this end, one needs to acquire multiple skills, including data analysis, visualization, and animation design, which will be challenging for ordinary users.

During the past decades, in the field of data visualization, techniques for creating insightful animations have been extensively studied [2]–[5] and a number of authoring tools [6]–[8] have also been developed for helping users create smooth transitions between charts. These techniques and tools greatly lower the technical barriers to designing meaningful animated transitions in data visualization. However, none of them aim to help users translate an analysis task into a series of data operations and visualize them as a datamation.

Designing such a datamation authoring tool is not simple and a number of challenges exist. First, it is usually difficult to clearly and precisely describe an analysis task. In most cases, users can only tell what they would like to find from the data in natural language. Second, selecting a series of data operations to create a datamation is hard because the right solution should not only provide the correct analysis result but also needs to ensure a continuous data flow (i.e., the output of the current step is the input of the next step) so that the changes can be smoothly represented in animation. Third, generating a meaningful animation of the analysis pipeline is difficult. Even though the intermediate analysis results may be shown in different forms of visualizations, the animation should be able to smoothly connect them without losing focus or increasing cognitive load.

To address the above issues, in this paper, we introduce Urania , the first authoring tool developed for creating datamations. Urania employs an advanced deep learning model that can directly and automatically decompose a data query described in natural language into a sequence of pre-defined operations. Each operation corresponds to a series of low-level actions that drive a captioned unit visualization to visualize the change of the intermediate analysis results through animated transitions. A preview of the generated datamation is displayed in an editor in which users can edit the analysis pipeline and the details of each of its analysis actions. The system also incorporates a deep knowledge editing network to help improve the generation results based on users' feedback. In particular, users can calibrate the decomposition results in the datamation editor and feed their modifications back into the decomposition model to improve it.

The contributions of the paper are as follows:

- **System.** We introduce the first, to the best of our knowledge, an intelligent authoring tool that is developed to support datamation design and generation. With this system, a user can easily generate an initial datamation by simply specifying a data query in natural language and then editing to refine it via rich interactions and a feedback mechanism based on his/her preferences.
- **Data Query Decomposition with Calibration.** We introduce a data query decomposition model that automatically converts a natural language data query into a sequence of data analysis operations for generating a datamation. The model integrates a knowledge calibration network to support an efficient feedback mechanism.
- **Action-Oriented Unit Visualization.** We introduce a unit visualization driven by a set of low-level data actions designed for illustrating datamations. The visualization uses a caption to illustrate the semantics of each action and show the effects of the action via animated transitions.

Here we adopt the term "unit visualization" from Drucker and Fernandez [9], which describes the class of visualization techniques that have a direct mapping between individual data items and corresponding unique visual marks.

We evaluate Urania via quantitative experiments of the performance of the query decomposition model, a controlled user study to assess the quality of the generated animations,

Yi Guo, Nan Cao, Ligan Cai, Yanqiu Wu, Danqing Shi and Qing Chen are with Intelligent Big Data Visualization Lab, Tongji University. E-mail: {2010937, nan.cao, tsailgan, 1941923, sdq, qingchen}@tongji.edu.cn.
Nan Cao & Qing Chen are the corresponding authors. .
Daniel Weiskopf is with University of Stuttgart. E-mail: { Daniel.Weiskopf@visus.uni-stuttgart.de}

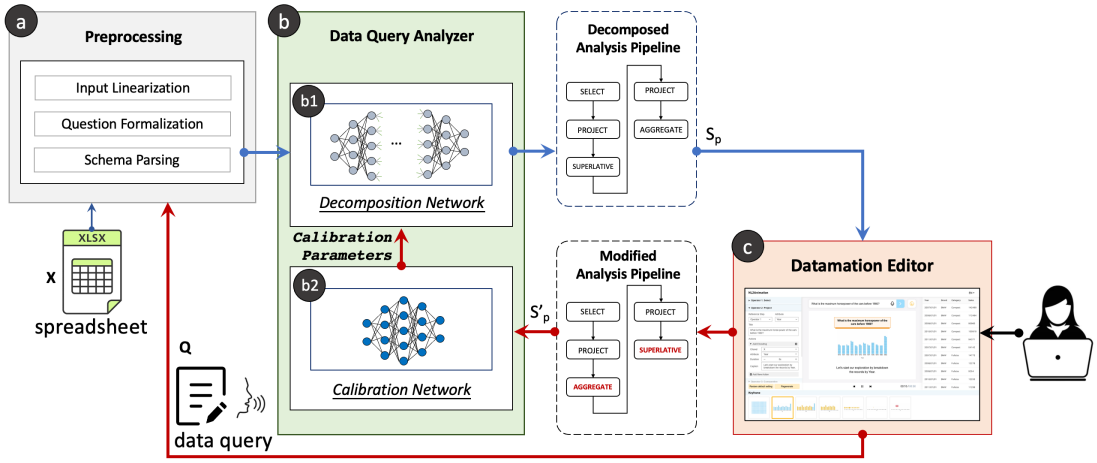

Fig. 1. The architecture design of Urania system consists of three major modules: (a) preprocessing, (b) data query analyzer, (c) datamation editor.

and interviews with two expert users to verify the usability of the system.

II. Related Work

In this section, we review the recent studies that are most relevant to our work: animation in data visualization, animation generation, and natural language interfaces (NLIs) for data visualization.

A. Animation in Data Visualization

In the field of data visualization, animations are usually used for illustrating the change of data [10], showing the transitions between visualization views [11], highlighting relationships [6], [12], and catching attentions [10]. It has also been used for supporting data analysis. Some works focus on using animation for highlighting [13] or adding information shown in static plots to boost reading comprehension. Hypothetical outcome plots (HOPs) [14], for example, augment static visualizations (e.g., error bars) with animated frames of random draws from the underlying sampling distribution to convey uncertainty. The most recent work by Pu *et al.* [1] introduces the idea of datamation, which is designed to support users in interpreting the results from complex data analysis by animating the detailed analysis pipeline step by step. The datamation enhances a static visualization with details from the data analysis phase, which can convey important insights and help people understand specific analysis results in everyday settings.

Following the idea of datamation [1], we have developed Urania , an intelligent authoring tool that supports datamation design and generation. We introduce a unit visualization driven by a set of low-level data actions designed for illustrating datamations. The visualization uses captions to illustrate the semantics of each action and show the effects of the action via animated transitions of units.

B. Animation Generation

Creating animated visualization can be difficult and time-consuming. A range of tools has been introduced to help users create animated transitions. Comprehensive libraries, such as D3 [5], allow flexible creation and great expressiveness but require significant effort. Users need to write program code to calculate and assign values for low-level components, impairing the ease of use. High-level grammar can help balance the trade-off between flexibility and ease of use. One example of this is Gemini and Gemini2 [3], [15], which suggest and execute animated transitions between two Vega-Lite charts. While these grammars avoid imperative programming, it is still challenging for ordinary users with little programming background to operate the grammars.

To provide a programming-free environment, existing approaches choose to either automatically generate animations by algorithms [4], [16] or provide graphical interfaces for authoring [7], [8], [17]. For instances, Data Animator [8] and CAST [7] allow users to create animations using a GUI. Users can create keyframes by importing Data Illustrator projects or selecting graphic components, and then design animations with timing parameters and data mappings between adjacent keyframes. While the aforementioned tools have eased the authoring process, creating a datamation is still not easy. Users have to prepare the keyframes or the data facts that are used to synthesize animations.

Unit visualization [9] facilitates smooth transitions between different visualization views. It presents data items as units and transforms them into various visual forms via animated transitions. Recent studies have utilized it for creating animated transitions [7] and data stories [18]. However, direct authoring of datamations, which requires careful consideration of the analysis pipeline and data flows, is not supported in these works. Urania also leverages the flexibility of the unit visualization design to animate an analysis pipeline. Moreover, we introduced a set of low-level actions that precisely control the existence, appearance, and layout of the units to help with the datamation authoring.

C. NLIs for Data Visualization

To lower the barriers of creating a visualization chart, there are techniques that can automatically generate or recommend visualizations of tabular data input. Various NLIs for data visualization have been explored within the research community [19]–[27] and industry [28], [29].

While NLIs provide flexibility in posing data-related questions, inherent characteristics of natural language such as ambiguity and under-specification make precisely understanding user intentions a challenging task. To overcome this obstacle, NLIs are designed to either guide users to provide a more concrete nature language query [21], [25], [30] or untangle ambiguities in the query [19], [20], [26] to capture user intent.

One approach to parsing natural language uses pre-defined grammars. Flowsense [30] and Eviza [21] depend on a pre-defined grammar to capture query patterns. Articulate [19]

allows people to generate visualizations by deriving mappings between tasks and data attributes in user queries, the translation of imprecise user specification is based on a natural language parser enriched with machine learning algorithms that can make reasoned decisions. The most recent work, ncNet [31], builds a Transformer model to translate natural language queries to visualization specifications.

Most of the existing NLIs are designed to translate the natural language to one or more static visualizations. How to appropriately generate animation based on natural language has yet to be studied. We address this lack in the literature by introducing a language-driven authoring tool developed to support datamation design and generation.

III. System Overview

In this section, we first describe the design requirements of the Urania system, followed by an introduction of the system architecture and running pipeline. In the end, we briefly review the definition of QDMR operations.

A. Design Requirements

The proposed Urania system has been designed to meet several real-world requirements for creating an interpretable data analysis pipeline and showing it via data animations. The design requirements were derived by reviewing existing literature and discussing with domain experts. Below, we describe the most critical requirements that motivate the design adopted in our work.

- **R1 Supporting data queries in natural language.** The system should allow users to input queries about the data in natural language to facilitate data analysis and datamation authoring.
- **R2 Uncovering the analysis pipeline for interpretation.** The system should not only retrieve the desired results based on the input query, but also uncover the underlying analysis pipeline, aiding result interpretation.
- **R3 Creating comprehensible animations.** The system should be able to generate meaningful animations that are easy to follow and understand, illustrating the changes between any of the two succeeding steps in the analysis pipeline.
- **R4 Learning from users' feedback.** To ensure the quality of the generated datamations, the system should allow users to refine the generation results.

B. System Architecture and Running Pipeline

To fulfill the above requirements, we introduce the Urania system. As shown in Fig. 1, it consists of three major modules: (a) the *Preprocessing Module*, (b) the *Data Query Analyzer*, and (c) the *Datamation Editor*. Specifically, when a user uploads a tabular dataset X and input a data query q in natural language (**R1**), the *Preprocessing Module* (Fig. 1(a)) parses X and q into a word sequence q_x to facilitate the calculations in the following steps.

In the next step, the *Data Query Analyzer* translates the q_x into a sequence of Question Decomposition Meaning Representation (QDMR) operations [32]:

$$S_{op} = [op_1, op_2, ..., op_n] \leftarrow Decompose(q_x) \quad (1)$$

where each operation op_i indicates a simple calculation (e.g., filter or aggregate) on the input data or the outputs of the previous operation(s). Ideally, executing these operations in order will provide the desired results regarding the data query. In this case, the above operation sequence will indicate a valid analysis pipeline (**R2**).

Finally, each operation in S_p will be used to drive an action-oriented dynamic unit visualization to generate a datamation (**R3**). The generated datamation will be displayed in the *Datamation Editor*, in which users can interactively remove an operator, change an operator's order, or add new operators in S_{op} to refine or fix the generation results. The datamation will be updated accordingly in real-time during the editing process. Users' modifications will be recorded and fed back into the *Data Query Analyzer* (**R4**), in which a pre-trained deep knowledge editing network (Fig. 1(b2)) takes the modified sequence S'_p to calculate a set of parameters to calibrate the decomposition model.

C. QDMR

QDMR is an approach to representing a question by a sequence of operations that can be executed to answer the question [32]. Each operation is responsible for querying the source data or analyzing the outputs of the previous operation(s). It is a data-independent representation that can be applied to many NLP benchmarks. Wolfson *et al.* [32] released Break, a question decomposition dataset of 83,978 questions over ten NLP benchmarks, such as data-related questions from Spider [33], document-related questions from HotPotQA [34], and image-related questions from CLEVR [35]. In our work, we focus on ten types of operations employed to represent the data-related questions in Spider [33], and use them to drive the datamation generation. The formalization and detailed explanation of each operation are provided in Table I(a).

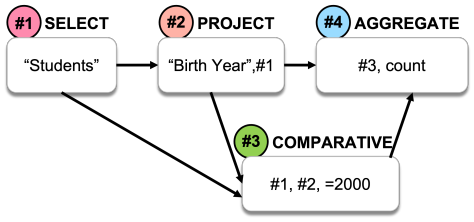

Fig. 2. QDMR operations chain to answer the question: "how many students were born in 2000?"

Fig. 2 demonstrates an example of using QDMR to resolve the question: *how many students were born in 2000?* A sequence of four operations is used to answer this question:

- Step 1: `SELECT` ("Students"). `SELECT` is an operation used to retrieve records from the data based on certain conditions. It is similar to the "select" statement in the structured query language (SQL). The argument *"Students"* corresponds to a table name in the dataset. In step 1, the intent of the operation is to select all the student records from the data for further exploration.
- Step 2: `PROJECT` ("Birth Year", #1). `PROJECT` is an operation that retrieves a certain attribute from the source/input records. The argument *"Birth Year"* specifies the attribute to be retrieved, which is usually a data column; *#1* corresponds to the output of step 1. In step 2, the intent of the operation is to retrieve the year of birth of previously selected students.
- Step 3: `COMPARATIVE` (#1, #2, "=2000"). The `COMPARATIVE` operation uses a comparative condition to filter records based on a specified attribute. The argument *#1* denotes the records that need to be filtered; the second argument *#2* is the attribute based on which the

TABLE I
THE QDMR OPERATIONS AND THE CORRESPONDING ACTIONS DESIGNED TO DRIVE A DYNAMIC UNIT VISUALIZATION.

(a) QDMR OPERATIONS			(b) UNITVIS ACTIONS		
Operator	**Arguments**	**Description**	**Data**	**Visual**	**Annotation**
SELECT	table/column, condition	Select data records from a data source (i.e., table/column) for the given *condition*	select	layout	/
PROJECT	records, attribute	Retrieve the *attribute* values from the *records*	/	size [numerical] color [categorical] x-axis [temporal]	/
COMPARATIVE	records, attribute, condition	Filter out the *records* whose *attribute* value satisfies $(=, \neq, >, <, \geq, \leq)$ a condition	filter	/	highlight,hide
SUPERLATIVE	records, attribute, superlative condition	Find a record from the *records* whose *attribute* has the maximum/minimum value	filter	/	highlight,hide
AGGREGATE	records, agg methods	Compute the *max/min/sum/count/avg* value of the records	aggregate	/	annotate
GROUP	records, attribute agg methods	Group *records* by the *attribute* and compute the *max/min/sum/count/avg* value of each group	/	x-axis[temporal] y-axis[categorical]	annotate
UNION	$records_1$, $records_2$	Combine $records_1$ and $records_2$	union	/	/
DISCARD	$records_1$, $records_2$	Find the instances in $records_1$ but not in $records_2$.	filter	/	hide
INTERSECTION	$records_1$, $records_2$	Find the instances belonging to both $records_1$ and $records_2$	intersection	/	hide
SORT	records, attribute, *asc/desc*	Sort the *records* by *attribute* in a *asc/desc* order	sort	/	/

data will be filtered (here, the birth year of students). The last argument, *"=2000"*, is the comparative condition based on which the data will be filtered. Step 3 aims to find out the students born in 2000.

- Step 4: `AGGREGATE` (#3, count). `AGGREGATE` computes the aggregated value of the records. QDMR supports six frequently used aggregation methods: count, max, min, sum, avg, and median. The first argument refers to the data records to be aggregated. The second argument indicates the aggregation method. Step 4 calculates the number of students in the filtered records.

QDMR has been used in various tasks, such as open domain QA [32], natural language for executable database queries [36], and contrast set generation [37]. In our work, we use QDMR as a structured intermediate representation of questions, connecting to datamation generation. Each type of QDMR operation maps to a series of low-level actions that drive a captioned unit visualization for the animation. After resolving a question into a sequence of QDMR operations, these operations can be used to generate a datamation to visualize the changes of the intermediate analysis results through animated transitions.

IV. DATA QUERY ANALYZER

The *Data Query Analyzer* is designed to resolve an input natural language data query in the context of a given dataset and decompose it into a sequence of data operations in the form of QDMR [32]. When a decomposition error occurs, it will modify the result based on users' feedback. Specifically, as shown in Fig. 3, the design of the *Data Query Analyzer* consists of two parts: the decomposition network ($\mathcal{D}$) and the calibration network ($\mathcal{C}$). When $\mathcal{D}$ wrongly resolves a data query q_x into a problematic operation sequence s_d (Fig. 3(a)), the user can modify it to provide a calibration s_c. $\mathcal{C}$ takes (q_x, s_c) as the input and produces a parameter calibration $\Delta\theta$ for $\mathcal{D}$ (Fig. 3(b)). By adding $\Delta\theta$ to $\mathcal{D}$'s parameter θ, the decomposition network will be able to generate correct results given q_x or the questions similar to q_x (Fig. 3(c)) without affecting the decomposition results of other questions (Fig. 3(d)). Next, we will introduce the technical details of the proposed decomposition network and the corresponding feedback mechanism.

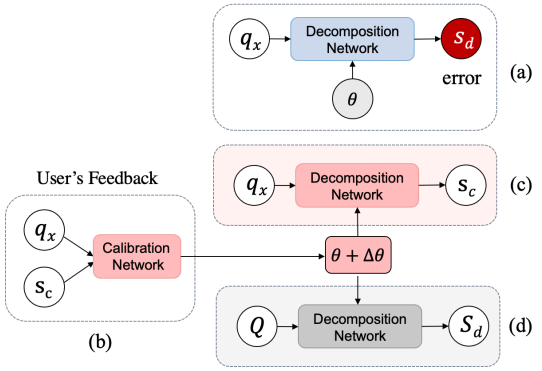

Fig. 3. Data query decomposition with online knowledge calibration.

A. Decomposition Network ($\mathcal{D}$)

We adopt and fine-tune a pre-trained natural language model T5 [38] that was developed based on the Transformer architecture [39] to translate the input data query (and the corresponding data scheme) to a sequence of QDMR operations. T5 is used due to its many advantages shown in a wide range of translation-related tasks, such as natural language translation [40], text2sql [41], and text summarization [42]. We fine-tune the model based on the aforementioned QDMR data corpus. To make it simple, we present the conceptual calculation steps in the context of our problem to show a brief idea about the model but leave the mathematical details to Vaswani *et al.* [39].

Generally, a Transformer follows the encoder-decoder architecture. Given a preprocessed data query q_x, the model first embeds q_x into a vector representation v. Then, it encodes v into a latent vector h_e that captures the semantics of the data query q and the corresponding tabular data X:

$$h_e = encode(v), \quad v = embed(q_x) \tag{2}$$

Later, the decoder takes h_e to compute a decoded latent vector h_d and then uses it to compute the output probabilities of the

tokens in the vocabulary given by the training samples via a softmax layer:

$$h_d = decode(h_e) \tag{3}$$

$$P_{voc} = softmax(W h_d^\top) \tag{4}$$

where W is a weight matrix to be trained. In each round, the token in the vocabulary having the highest probability is chosen as the output of the model. In this way, the model generates a QDMR operation sequence token by token in the following form:

$$[op_1\{\mathtt{t}_{11}, \ldots, \mathtt{t}_{1j}\}, \ldots, op_n\{\mathtt{t}_{n1}, \ldots, \mathtt{t}_{nk}\}] \leftarrow decode(h_e) \tag{5}$$

where $\mathtt{t}_{ij}$ indicates the j-th token of the i-th operator.

a) **Loss Function:** To encourage the output of the decomposition network as identical as possible with the target analysis pipeline in our training corpus, the model is trained by using cross-entropy loss between generated operation sequence and ground-truth:

$$L = -\sum_{i=1}^{n} \log p(t_i \mid t_1, \ldots, t_{i-1}, q_x) \tag{6}$$

where t_i is the current reference token in the target pipeline. Given the previous tokens $t_1, \ldots, t_{i-1}$ and the input data query q_x, this loss function tends to maximize the probability of the reference token t_i as the prediction in the current step.

b) **Training Corpus:** We adopt the dataset introduced in [36] to train our decomposition network. It was generated by manually annotating the Spider dataset [33] based on the QDMR operations. In particular, it contains 7,423 natural language data queries about a number of databases introduced in Spider. Each query corresponds to a sequence of manually annotated QDMR operations together with attributes such as the table and column names.

c) **Implementation:** Our decomposition network was implemented based on PyTorch [43] and fine-tuned via 20 epochs with the gradient accumulation step as 16. The batch size was set to 8. The Adam [44] optimizer was used and the learning rate was set to 2e-4. The overall training procedure spent around 2 hours on an Ubuntu server with a V100 GPU.

B. Calibration Network ($\mathcal{C}$)

Although effective, the above model cannot guarantee accurate decomposition results, sometimes producing disordered or incomplete pipelines. Interactive user feedback can help address these issues, but accumulating enough feedback for retraining or fine-tuning the deep learning model can be time-consuming, hindering quick responses. To solve this problem, we introduce an efficient online post-hoc knowledge calibration mechanism, as shown in Fig. 3, that uses feedback to modify the original decomposition network's parameters (θ) with a calibration ($\Delta\theta$) to produce desired results as suggested in user feedback.

A deep knowledge calibration network (Fig. 4(a)), denoted as $\mathcal{C}$, is designed to implement the above idea. It is a collection of auxiliary multi-layer perceptrons (MLPs). We choose to use MLP for building the network due to many of its advantages such as excellent fault tolerance and strong adaptive and self-learning features. Each MLP is responsible for calibrating the parameters of a corresponding layer in the decomposition network toward the direction of making the model output the desired operation sequence s_c given by the users' feedback regarding the data query q_x. We implement the MLP with a single hidden layer and adopt skip connection [45] to improve

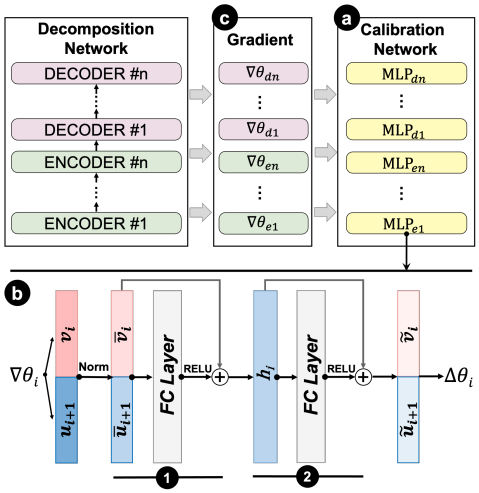

Fig. 4. The architecture of the calibration network.

the performance and convergence of the MLP (Fig. 4(b)). The input of each MLP is the loss gradient ($\nabla_{\theta_i} L(q_x, s_c)$, Fig. 4(c)) of the decomposition network in the i-th layer calculated when fine-tuning the decomposition network based on user's feedback s under the parameter setting of θ_i. The outputs of the MLP are the parameters $\Delta\theta_i$ that calibrate θ_i to make the decomposition network achieve a result that matches s_c. In particular, the loss gradient in the i-th layer is calculated based on the chain-rule during a back-propagation process when fine-tuning the decomposition network based on s_c. As the feedback s_c is processed token by token, the gradient is thus computed token by token as well. We compute an averaged gradient that could be used as the input to the aforementioned MLP:

$$\nabla_{\theta_i} L(q_x, s_c) = \frac{1}{n}\sum_{j=1}^{n} \nabla_{\theta_i} L(q_x, j) = \frac{1}{n}\sum_{j=1}^{n} u_{(i+1)j} v_{ij} \tag{7}$$

where n is the total number of tokens in s; $u_{(i+1)j}$ and v_{ij}, respectively, indicate the loss gradient in the last layer and the hidden vector in the currently layer corresponding to the j-th token in s_c. Equation (7) is derived based on the chain rule of the back-propagation process. Details can be found in the book by Goodfellow *et al.* [46] (Section 6.5).

However, the above gradient is a high-dimensional vector. Therefore, it requires even more parameters to directly train a network that maps such a gradient to a parameter calibration. It could be millions of parameters to tune when calibrating a large decomposition model like T5, making it very difficult to converge. To address this issue, we leverage the gradient decomposition strategy [47] to reduce the number of parameters used in the calibration network. The idea is to directly map each gradient corresponding to a token j in s_c into a parameter calibration $\Delta\theta_{ij}$ independently via an MLP, and then use the average as the overall calibration $\Delta\theta_i$:

$$\Delta\theta_i = \frac{1}{n}\sum_{j=1}^{n} \Delta\theta_{ij} = \frac{1}{n}\sum_{j=1}^{n} \widetilde{u}_{(i+1)j} \widetilde{v}_{ij} \tag{8}$$

where $\widetilde{u}_{(i+1)j}$ and $\widetilde{v}_{ij}$ are the output of an MLP that are derived based on $u_{(i+1)j}$ and v_{ij} as follows:

$$z_{ij} = concat(norm(u_{(i+1)j}), norm(v_{ij})) \tag{9}$$

$$h_{ij} = s_i \odot (z_{ij} + \sigma(U_1 V_1 z_{ij})) + o_i \tag{10}$$

$$\widetilde{z_{ij}} = s'_i \odot (h_{ij} + \sigma(U_2 V_2 h_{ij})) + o'_i \tag{11}$$

$$\widetilde{u}_{(i+1)j},\ \widetilde{v}_{ij} = split(\widetilde{z_{ij}}) \tag{12}$$

where Eq. (10) and Eq. (11), respectively, indicate the com-

putation of the two consecutive blocks in MLP as shown in Fig. 4(b1,b2). In particular, the first block (Eq. (10)) takes a vector z_{ij} that concats the normalized $u_{(i+1)j}$ and v_{ij} (Eq. (9) as the input followed by a fully connected layer whose weight matrix is factorized into U_1 and V_1, which will be gradually transformed into the final output. Here, we use ReLU (denoted as $\sigma(\cdot)$) as the activation function whose output is directly added with the input z_{ij} through a skip connection in purpose of mitigating the degradation issues during the training process. s_i and o_i are the trainable scale and offset used to regularize the output of the first block before passing it to the next step. The second block takes the previous output h_i to perform a similar computation as shown in Fig. 4(b2) and described in Eq. (11). The final result $\widetilde{z_{ij}}$ is split into the desired $\widetilde{u}_{(i+1)j}$ and $\widetilde{v}_{ij}$ (Eq. (12)).

Finally, we update the parameter θ_i in the i-th layer of the decomposition network by θ_i' as follows:

$$\theta_i' = \theta_i + \Delta\theta_i \tag{13}$$

a) **Loss Function:** The above calibration network is trained one feedback (q_k, s_{ck}) a time based on a set of training samples consisting of data queries $Q_x = \{q_1, \cdots, q_m\}$. Their decomposition results $S_d = \{s_{d1}, \cdots, s_{dm}\}$ are generated before calibrating the decomposition network, and the desired results $S_c = \{s_{c1}, \cdots, s_{cm}\}$ are given by users' feedback. In each round of training, the following loss function is minimized:

$$L = \alpha L_r + L_p \tag{14}$$

where L_r is the cross-entropy loss that estimates the similarity between the decomposition result generated after calibrating the decomposition network using θ' and the desired results indicated in a user's feedback $s_c \in S_c$:

$$L_r = -\sum_{i=1}^{n} \log p_{(\theta')}(t_i \mid t_1, \ldots, t_{i-1}, q_k) \tag{15}$$

where $p_{(\theta')}(\cdot)$ estimates the probability of generating a token t_i that belongs to a desired decomposition result in $s_c \in S_c$, given all the previous tokens $t_1, \ldots, t_{i-1}$ and the data query q_k, after calibrating the decomposition network based on θ'.

In Eq. (14), L_p is the Kullback-Leibler divergence that estimates the similarity of the output distributions before and after calibrating the decomposition network $\mathcal{D}$ regarding to the remaining data queries $Q_x' = Q_x - \{q_k\}$:

$$L_p = KL(\mathcal{D}(Q_x'|\theta), \mathcal{D}(Q_x'|\theta')) \tag{16}$$

Intuitively, minimizing L_r ensures that the feedback will be accepted by the decomposition network and minimizing L_p will prevent our calibration from affecting the decomposition results beyond the feedback.

b) **Training Corpus:** We prepared a corpus with 12k data samples to train the calibration network. Each sample is a triplet that consists of a data query q_x, the corresponding problematic decomposition result s_d, and a correction s_c, denoted as (q_x, s_d, s_c). To collect these data samples, we deliberately trained a decomposition model with a low accuracy based on T5 [38] by only using a small subset of training samples randomly selected from the aforementioned corpus. We tested the entire datasets on the model and selected the incorrect outputs s_d together with the corresponding data query q_x and ground truth s_c into our calibration data set. We iteratively performed the above training and testing process by using a different subset of samples to train the model every time. We enriched the data queries by translating the original English queries into Chinese and then translating it back to English via Google translate. As a result, a number of 12,227 unique training samples were collected, which covered two types of decomposition errors: missing operations (3340 samples) and disordered sequences (4693 samples).

Fig. 5. The user interface of the datamation editor.

c) **Implementation:** The calibration network was also implemented in PyTorch. We chose Adam [44] as the optimizer and set the learning rate as 1e-4. The overall training procedure spent around 16 hours on an Ubuntu server with one NVIDIA V100 GPU.

V. Animation Generation and Datamation Editor

In this section, we first introduce how the QDMR operation sequences are visually represented by an action-orientated unit visualization to generate datamations. After that, we describe the design of the datamation editor and the interactions.

A. Action-Oriented Unit Visualization

We developed an action-oriented dynamic unit visualization to visualize and animate an analysis pipeline represented in the form of a sequence of decomposed QDMR operations as a datamation. The proposed unit visualization represents each data item via a mark, known as a unit, whose existence, appearance, and layout are precisely controlled by a collection of low-level actions. In this way, by translating each QDMR operation into a series of these actions, we are able to control the visualization to dynamically represent the detailed analysis steps via the animated transitions between succeeding unit visualization views that represent the intermediate analysis results. Our design was implemented based on **D3.js** [5].

In particular, the following three types of low-level actions are developed, which are respectively responsible for processing the underlying data (*data actions*), manipulating the visual encoding methods (*visualization actions*), or adding annotations on units (*annotation actions*):

a) **Data Actions:** The data actions are designed to update the data items represented in the unit visualization via the following approaches: (1) ***select*** a set of qualified data items; or (2) ***filter*** the current data items based on a condition; or (3) ***union*** or ***intersect*** two subgoups of data items shown in the view; or (4) ***aggregate*** the data items to calculate a statistic measure such as minimum, mean, and sum; or (5) ***sort*** the data items in order. When these actions are performed, the visualization layout will be updated accordingly to add, remove, merge, or rearrange the units shown in the visualization.

b) **Visualization Actions:** In our design, the visual appearance (position, color, and size) of each unit in the visualization is determined by the data mappings through four independent visual channels, i.e., x-axis, y-axis, unit size, and

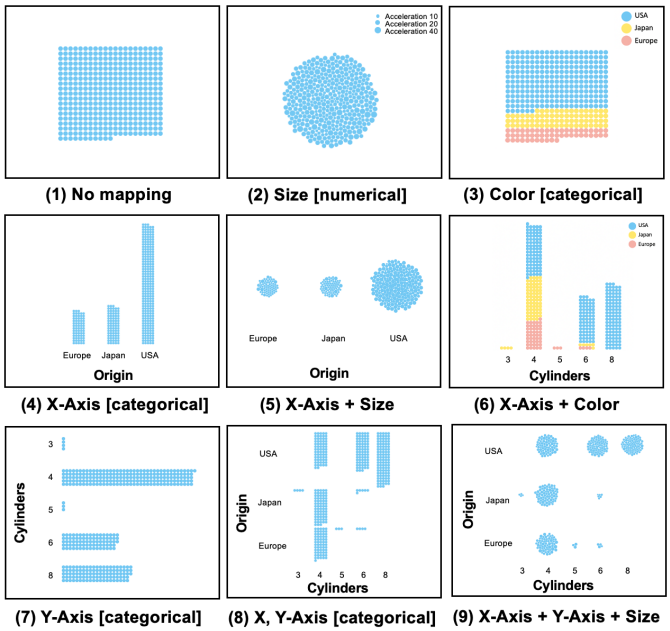

Fig. 6. The appearance of the unit visualization is controlled by a set of visual mapping actions.

unit color designed for mapping different types of data attributes. The visualization actions were designed to control the data mapping of these channels. In particular, ***x*** and ***y*** actions are used for mapping numerical, categorical, or temporal data attributes via {x}-axis and {y}-axis, resulting either in a scatter plot view for representing data with continuous numerical attributes or a group view for the data with discrete categorical attributes (Fig. 6(4-9)). The ***size*** action was designed to encode a numerical data attribute by the size of a unit. By default, the size of each unit is mapped to unit 1 and the units in the visualization are packed in a square form to facilitate counting (Fig. 6(1)). When the size channel is used to map a numerical attribute, the size of each unit is proportional to the corresponding attribute value, and the layout of the units will be changed to circle packing to avoid the overlaps of the units with different sizes (Fig. 6(2)). Finally, the ***color*** action is used to map a categorical data attribute by the filling colors(hue) of the units (Fig. 6(3,6)). Combining these actions will provide flexible data mapping strategies, resulting in a variety of unit visualization forms with different layouts as shown in Fig. 6.

c) **Annotation Actions:** These actions were designed to ***highlight*** a focal unit by changing its filling color, or ***delight*** the non-focal units by making it invisible, i.e., hidden in the view, or ***annotate*** a unit or a group of units by adding a textual tooltip that shows the specific data attributes of the unit or group. The annotation actions will not affect the change of the layout. Their effects will just be shown through a fade-in or fade-out animation.

B. Datamation Generation

With the above actions in mind, we create a smooth and meaningful datamation via the following three steps:

a) **Reordering:** The major changes in an analysis process are related to the data produced by the intermediate analysis steps, thus it is important to first sort the QDMR operations in order to provide a fluent data flow so that the output of the current operation is the input of the next operation. Although in most cases, the decomposition model will produce QDMR operations in the right order, the system will still double-check the decomposition results and reorder the operations to ensure a smooth data flow when necessary. When the automatic reordering fails, errors will be reported for users to fix. Such a failure is usually caused by a problematic decomposition result that contains a wrong type of QDMR operation.

b) **Translation:** In the second step, we translate each operation into a sequence of the aforementioned actions based on the rules described in Table I. These actions are also arranged in order to guarantee a smooth transition. In particular, we first perform the data actions to update the data items shown on the view and then gradually modify their encoding methods and add annotations via the visualization and annotation actions in the follow-up steps.

c) **Captioning:** Finally, to help communicate the operation's meaning and effects, we generate a caption for each QDMR operation based on a pre-defined template by considering its corresponding low-level actions. The caption describes the change in the data and the visual mappings. It is displayed on top of each view and updated gradually during the animation process. For example, the `PROJECT` operation can be described as "use size/color/x-axis to present/encode [attribute]" and the `AGGREGATE` operation is described as "the maximum/minimum/average/total value of [a numerical attribute] of the following [units] is [a numeric value]".

Now, let us take the following operation sequence introduced in Section III-C as an example to illustrate how to generate a datamation based on an input QDMR operation sequence:

$$[\texttt{SELECT}][\texttt{PROJECT}][\texttt{COMPARATIVE}][\texttt{AGGREGATE}]$$

it can be translated into an action sequence as follows:

$$[select, layout][x-axis][filter, highlight]$$
$$[hide][aggregate, annotate]$$

Semantically, it indicates to (1) *select* a collection of data records, i.e., students, from the data source and represent each record as a unit and layout all the records in the unit visualization; (2) *encode* the attribute, i.e., year of birth, by x-axis; (3) *filter* out the records that satisfies the condition and highlight them by filling each qualified unit color and (4) *hide* those unqualified ones; (5) *aggregate* the records, i.e., students born in 2000, by counting the units and annotate on records via a tooltip to illustrate the total number of students.

C. User Interface and Interactions

The *datamation editor* is designed to translate the QDMR operation sequence into a datamation and help users refine it, and eventually acquire a datamation with correct analysis results and coherent animations. In particular, when a user uploads a spreadsheet into the system, the raw data is displayed in the data panel (Fig. 5-1), which allows users to easily access the data during the authoring process. The user can preview the data and communicate an analysis task of interest by typing a data query into the input box (Fig. 5-2). By resolving the query input, the underlying system will generate an operation sequence to create an initial datamation. The datamation is displayed in the center of the interface (Fig. 5-3) with playback buttons at the bottom to control the datamation. Meanwhile, the key-frames of datamation, which correspond to QDMR operations, are arranged and visualized (Fig. 5-5) to illustrate the intermediate analysis result of each operation. Users can drag to rearrange their orders when necessary. Once modified, the order of the operations shown in the configuration panel (Fig. 5-4) will be updated accordingly. Through the configuration panel, users can edit each of the QDMR operations or add new operations in the analysis

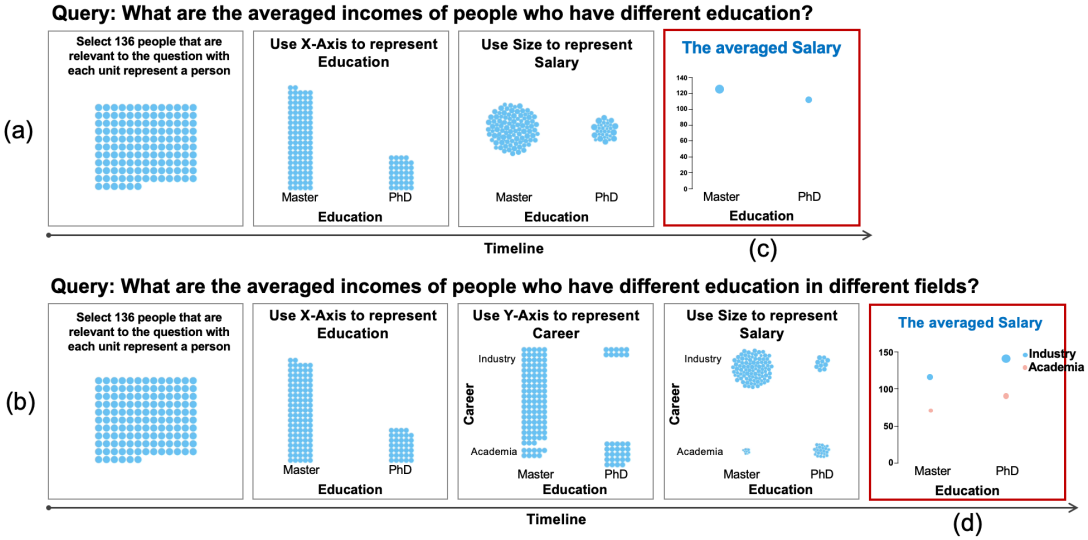

Fig. 7. In the user study, the participants were asked to resolve a paradox shown in this figure: (a) people with a higher education have lower average income; (b) however, when we take the work field into consideration, the contrary is true.

pipeline. In particular, the users can modify an operation's parameter and the corresponding actions used to update the unit visualization to fully control the design of the datamation. Finally, by clicking the update button, users' modifications can be fed back into the calibration network to update the decomposition model.

VI. Evaluation

We first estimated the decomposition and calibration network via quantitative experiments and verified the effectiveness of the generated datamation via a controlled user study. Finally, interviews with two expert users were performed to verify the usability of our system.

A. Quantitative Evaluation

We evaluated the performance of the decomposition network based on the corpus introduced in Section IV-A. In particular, we computed the rate of exactly match between the decomposition results and ground truth, and calculated the averaged SARI scores [48] of the decomposition results. SARI score is commonly used in text simplification tasks, it evaluates the goodness of words that are added, deleted, and kept in the simplified sentences. Two baseline models introduced in [32] were used for comparison. In particular, the first baseline ($\mathcal{B}_1$) was a sequence-to-sequence neural network with a 5-layer LSTM encoder and a cross attention. The second one ($\mathcal{B}_2$) was another sequence-to-sequence model that incorporated a copy mechanism [49] to deal with unseen queries. We trained these models based on the entire corpus and tested their performance via a validation set containing 30% of data that were randomly selected from the corpus with the data queries being replaced by similar but different questions. The evaluation results are summarized in Table II.

TABLE II
Performance evaluation of the query decomposition models.

Models	Exact Match	SARI
$\mathcal{B}_1$	25.64%	0.759
$\mathcal{B}_2$	38.39%	0.812
$\mathcal{D}$	**82.23%**	**0.876**

We measured the effectiveness of the calibration network via *success rate* and the *retain rate*. Here, the *success rate* was defined as the percentage of the incorrect decomposition results S_d that could be successfully amended by the calibration network via the feedback S_c among all the incorrect results. The *retain rate* was defined as the percentage of correct decomposition results that remain correct after updating the model's parameters via calibrations. We performed the experiment based on the above decomposition network $\mathcal{D}$ and calculated the *success rate* and the *retain rate* based on the dataset introduced in Section IV-B. In particular, we used 80% of the data to train the calibration network and used the rest 20% for validation. As a result the success rate was 76.61% and the retain rate was 91.79%.

B. User Study

We re-performed the experiment (Study-I) described in [1] based on our system to verify the effectiveness of the generated datamations. In particular, 40 lab students (18 males and 22 females, mean age 25.1, SD 1.77) from a design college with the background of visual communication design were invited to take part in our study. They were divided into two groups (i.e., G_1, G_2), where each group had 20 members. A between-subject study was performed, in which two groups of people were asked to answer the same question by exploring the data via datamations generated by Urania and the static charts (the last frames of the datamations), respectively.

a) Procedure and Tasks: During the study, the participants were asked to resolve a paradox as shown in Fig. 7: it seems people with higher education (PhD) have a lower average income (Fig. 7(c)), but when we take the work fields into consideration, we obtain the opposite result, i.e., people with higher education have a higher income in both industry and academia (Fig. 7(d)). The truth is that people with lower education tended to work in industry, where a higher salary is usually paid; and these people outnumbered the people with higher education by a large margin. We visualized the two opposite sides of the paradox in a pair of datamations generated by Urania as illustrated in Fig. 7(a,b). We showed these datamations to the participants in G_1 and showed the last frames as static charts to the participants in G_2. These participants were asked to read these visual representations carefully and then select the truth of the paradox from 8 potential choices (7 were distractors) provided by us (refer to [1] for details about the choices). Our hypothesis was G_1 tends to have higher accuracy than that of G_2.

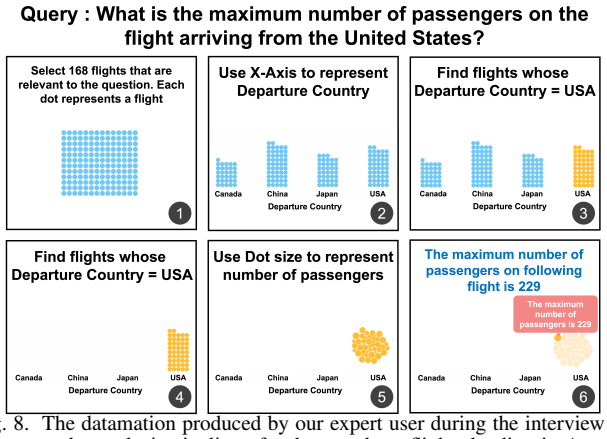

Fig. 8. The datamation produced by our expert user during the interview that illustrates the analysis pipeline of a dataset about flights landing in Australia in May 2013.

b) Results: As a result, only 40% of participants in G_2 correctly found out the truth of the paradox, while 75% of participants in G_1 did so. Chi-squared test showed that G_1 performed significantly better ($\chi^2(1, N = 40)$=5.01, p<.05, one-tailed) than G_2 in the experiment. The gap between G_1 and G_2 was 35%, which was a sizeable difference with Cohen's h-value equals to 0.68. This result supported our hypothesis.

C. Interview with Experts

Two expert users were interviewed separately. The first expert ($E1$) was a researcher in data visualization who had published over 10 TVCG papers. The second expert worked for an IT consulting company, whose job was to analyze customer data using tools like PowerBI and Tableau. During the interview, the experts were asked to create datamations using Urania based on datasets selected from the spider corpus. They tried a number of data queries and the corresponding datamations were created. Their operations, generation results, and comments were recorded in detail. Each interview lasted for about 1.5 hours. Here, we first demonstrate two example datamations generated by the experts and then report their comments in general.

*a) Case I: Flights Analysis (***E1).:** The first example was based on a dataset that consisted of 168 international flights that landed in Australia in May 2013. Each flight had four attributes: date, flight number, departure country, and number of passengers. Fig. 8 showed a valid datamation directly generated by Urania after the expert input the queried *"what is the maximum number of passengers on the flight arriving from the United States?"* The first frame illustrated all the related flights with each flight visualized as a unit. These flights were grouped by the countries that they arrived from in the second frame. The next three frames gradually highlighted the flights from the USA and encoded the size of each unit by the number of passengers on the corresponding flight. Finally, in the last frame, a tooltip was added to point out the flight had the maximum number of passengers.

*b) Case II: Cars Dataset (***E2).:** The second example as shown in Fig. 9 demonstrated a data query with multiple conditions about the famous cars dataset. The query initially resulted in a problematic datamation that contained only the first six frames shown in Fig. 9. The last two frames were added by E_2 and used as a feedback to calibrate the model. Specifically, to answer the data query, a set of 389 relevant cars were selected and visualized as units (Fig. 9(1)), which were latter grouped by their production year (Fig. 9(2)) with the targets highlighted in yellow (Fig. 9(3)). After that, these cars

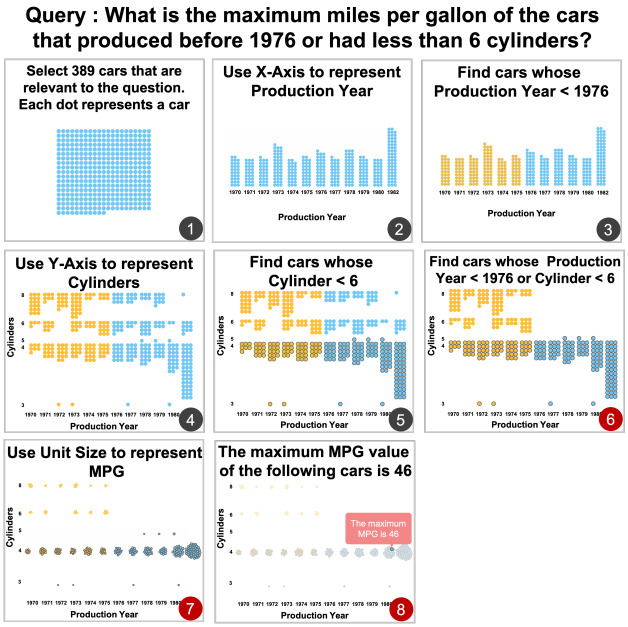

Fig. 9. The datamation produced by our expert user that illustrates the analysis pipeline of a dataset consisting of 389 cars.

were further divided regarding to their number of cylinders (Fig. 9(4)) with the cars with less than 6 cylinders highlighted by a thicker broader (Fig. 9(5, 6)). The last two frames encoded the MPG by size (Fig. 9(7)) and pointed out the final results by a tooltip (Fig. 9(8)).

c) **Interview Feedback:** Due to the page limitation, we restrict our report to summarizing only some major feedback:

Data query decomposition. All the experts felt starting a datamation generation process by inputting a nature language query was a "good idea that greatly reduces the technique barriers." $E1$ mentioned: "resolve a natural language question into data operations is useful for users with little data analysis background." $E2$ said: "although [the system] cannot always provide a comprehensive [decomposition] result, it is always a good start by pointing out the potential directions to solve problem."

The authoring tool. All experts were able to successfully generate datamations based on our system. They were quite excited when seeing an analysis pipeline shown in an animation. "This is my first time to see such kind of tool," said $E2$ and he continued: "this feature is very nice, I can hardly generate this kind of animations using the tools that I have ever used." $E1$ believed that "showing the analysis via an animation is very intuitive" and "it is a good strategy for interpreting an analysis result." At the same time, both $E1$ and $E2$ praised the interactive authoring functionalities supported in our system and even constructively suggested a number of new features such as "recommend proper operations for users to choose when creating an analysis pipeline ($E2$)" and "provide more visualization styles ($E1$)." In addition, $E1$ believed that our feedback mechanism is useful as "it could help the underlying model to reduce errors in the next time." $E2$ also mentioned that "[with the help of feedback] I can make the result better and better."

VII. Conclusion, Limitations, and Future Work

In this paper, we presented Urania , developed for creating datamations. To the best of our knowledge, it is the first tool that supports datamation design and generation. Given a dataset and a question, Urania can automatically decompose

the question into a sequence of data analysis operators and generate a datamation based on unit visualization. Urania also allows the user to modify and edit the generated results. Our user studies showed that Urania is highly rated for generating datamations to explain data analysis processes. Its editing function also showed to be effective in correcting the automatically generated results.

There are a number of limitations of our work that are worth further study in the future. First, both the decomposition and calibration networks are pre-trained models, with therefore limited capability of dealing with unseen data. Second, the current calibration network fixes the decomposition errors, but it is restricted in terms of improving the overall performance of the decomposition model. A knowledge accumulation mechanism that could incrementally capture the knowledge from users' feedback to gradually change the decomposition network is still desired. However, this is a difficult problem known in deep learning that is worth further exploration. Third, the scalability of the visualization design needs to be improved to support larger datasets.

REFERENCES

[1] X. Pu, S. Kross, J. M. Hofman, and D. G. Goldstein, "Datamations: Animated explanations of data analysis pipelines," in *CHI*, 2021, pp. 1–14.

[2] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang, "Canis: A high-level language for data-driven chart animations," in *Computer Graphics Forum*, vol. 39, no. 3. Wiley Online Library, 2020, pp. 607–617.

[3] Y. Kim and J. Heer, "Gemini 2: Generating keyframe-oriented animated transitions between statistical graphics," in *IEEE Visualization*. IEEE, 2021, pp. 201–205.

[4] D. Shi, F. Sun, X. Xu, X. Lan, D. Gotz, and N. Cao, "Autoclips: An automatic approach to video generation from data facts," in *Computer Graphics Forum*, vol. 40, no. 3, 2021, pp. 495–505.

[5] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE TVCG*, vol. 17, no. 12, pp. 2301–2309, 2011.

[6] F. Amini, N. H. Riche, B. Lee, J. Leboe-McGowan, and P. Irani, "Hooked on data videos: assessing the effect of animation and pictographs on viewer engagement," in *AVI*, 2018, pp. 1–9.

[7] T. Ge, B. Lee, and Y. Wang, "Cast: Authoring data-driven chart animations," in *CHI*, 2021, pp. 1–15.

[8] J. R. Thompson, Z. Liu, and J. Stasko, "Data animator: Authoring expressive animated data graphics," in *CHI*, 2021, pp. 1–18.

[9] S. Drucker and R. Fernandez, "A unifying framework for animated and interactive unit visualizations," *Microsoft Research*, 2015.

[10] G. Robertson, R. Fernandez, D. Fisher, B. Lee, and J. Stasko, "Effectiveness of animation in trend visualization," *IEEE TVCG*, vol. 14, no. 6, pp. 1325–1332, 2008.

[11] J. Heer and G. Robertson, "Animated transitions in statistical data graphics," *IEEE TVCG*, vol. 13, no. 6, pp. 1240–1247, 2007.

[12] D. Fisher, "Animation for visualization: Opportunities and drawbacks." *Beautiful visualization*, vol. 19, pp. 329–352, 2010.

[13] C. Ware and R. J. Bobrow, "Motion to support rapid interactive queries on node–link diagrams," *TAP*, vol. 1, no. 1, pp. 3–18, 2004.

[14] J. Hullman, P. Resnick, and E. Adar, "Hypothetical outcome plots outperform error bars and violin plots for inferences about reliability of variable ordering," *PloS one*, vol. 10, no. 11, p. e0142444, 2015.

[15] Y. Kim and J. Heer, "Gemini: A grammar and recommender system for animated transitions in statistical graphics," *IEEE TVCG*, vol. 27, no. 2, pp. 485–494, 2020.

[16] W. Li, Y. Wang, H. Huang, W. Cui, H. Zhang, H. Qu, and D. Zhang, "Anivis: Generating animated transitions between statistical charts with a tree model," *arXiv preprint arXiv:2106.14313*, 2021.

[17] F. Amini, N. H. Riche, B. Lee, A. Monroy-Hernandez, and P. Irani, "Authoring data-driven videos with dataclips," *IEEE TVCG*, vol. 23, no. 1, pp. 501–510, 2016.

[18] J. Lu, W. Chen, H. Ye, J. Wang, H. Mei, Y. Gu, Y. Wu, X. L. Zhang, and K.-L. Ma, "Automatic generation of unit visualization-based scrollytelling for impromptu data facts delivery," in *IEEE PacificVis*. IEEE, 2021, pp. 21–30.

[19] Y. Sun, J. Leigh, A. Johnson, and S. Lee, "Articulate: A semi-automated model for translating natural language queries into meaningful visualizations," in *SG*. Berlin, Heidelberg: Springer-Verlag, 2010, p. 184–195.

[20] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios, "Datatone: Managing ambiguity in natural language interfaces for data visualization," in *UIST*. ACM, 2015, p. 489–500.

[21] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang, "Eviza: A natural language interface for visual analysis," in *UIST*. ACM, 2016, p. 365–377.

[22] J. Aurisano, A. Kumar, A. Gonzalez, J. Leigh, B. DiEugenio, and A. Johnson, "Articulate2: Toward a conversational interface for visual data exploration," in *IEEE Visualization*, 2016.

[23] A. Narechania, A. Srinivasan, and J. Stasko, "Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries," *IEEE TVCG*, vol. 27, no. 2, pp. 369–379, 2021.

[24] E. Hoque, V. Setlur, M. Tory, and I. Dykeman, "Applying pragmatics principles for interaction with visual analytics," *IEEE TVCG*, vol. 24, no. 1, pp. 309–318, 2017.

[25] K. Cox, R. E. Grinter, S. L. Hibino, L. J. Jagadeesan, and D. Mantilla, "A multi-modal natural language interface to an information visualization environment," *International Journal of Speech Technology*, vol. 4, no. 3, pp. 297–314, 2001.

[26] Z. Wen, M. X. Zhou, and V. Aggarwal, "An optimization-based approach to dynamic visual context management," in *IEEE InfoVis*. IEEE, 2005, pp. 187–194.

[27] A. Srinivasan, B. Lee, and J. T. Stasko, "Interweaving multimodal interaction with flexible unit visualizations for data exploration," *IEEE TVCG*, 2020.

[28] "Microsoft Power BI Q&A," https://powerbi.microsoft.com, [Online; accessed 11-January-2022].

[29] "Tableau Ask Data," https://www.tableau.com/products/new-features/ask-data, [Online; accessed 11-December-2021].

[30] B. Yu and C. T. Silva, "Flowsense: A natural language interface for visual data exploration within a dataflow system," *IEEE TVCG*, vol. 26, no. 1, pp. 1–11, 2019.

[31] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin, "Natural language to visualization by neural machine translation," *IEEE TVCG*, vol. 28, no. 1, pp. 217–226, 2022.

[32] T. Wolfson, M. Geva, A. Gupta, M. Gardner, Y. Goldberg, D. Deutch, and J. Berant, "Break it down: A question understanding benchmark," *TACL*, vol. 8, pp. 183–198, 2020.

[33] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *EMNLP*. ACL, 2018, pp. 3911–3921.

[34] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, "HotpotQA: A dataset for diverse, explainable multi-hop question answering," in *EMNLP*. ACL, 2018, pp. 2369–2380.

[35] J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *CVPR*, 2017, pp. 2901–2910.

[36] I. Saparina and A. Osokin, "SPARQLing database queries from intermediate question decompositions," in *EMNLP*. ACL, 2021, pp. 8984–8998.

[37] M. Geva, T. Wolfson, and J. Berant, "Break, perturb, build: Automatic perturbation of reasoning paths through question decomposition," in *TACL*, 2021.

[38] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *EMNLP*. ACL, 2020, pp. 38–45.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NIPS*, vol. 30, 2017.

[40] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar *et al.*, "Tensor2tensor for neural machine translation," *arXiv preprint arXiv:1803.07416*, 2018.

[41] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *ACL*. ACL, 2020, pp. 7567–7578.

[42] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," *arXiv preprint arXiv:1908.08345*, 2019.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NIPS*, vol. 32, 2019.

[44] K. Diederik, B. Jimmy *et al.*, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, pp. 273–297, 2014.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[46] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[47] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning, "Fast model editing at scale," *arXiv preprint arXiv:2110.11309*, 2021.

[48] W. Xu, C. Napoles, E. Pavlick, Q. Chen, and C. Callison-Burch, "Optimizing statistical machine translation for text simplification," *TACL*, vol. 4, pp. 401–415, 2016.

[49] J. Gu, Z. Lu, H. Li, and V. O. Li, "Incorporating copying mechanism in sequence-to-sequence learning," in *ACL*. ACL, 2016, pp. 1631–1640.