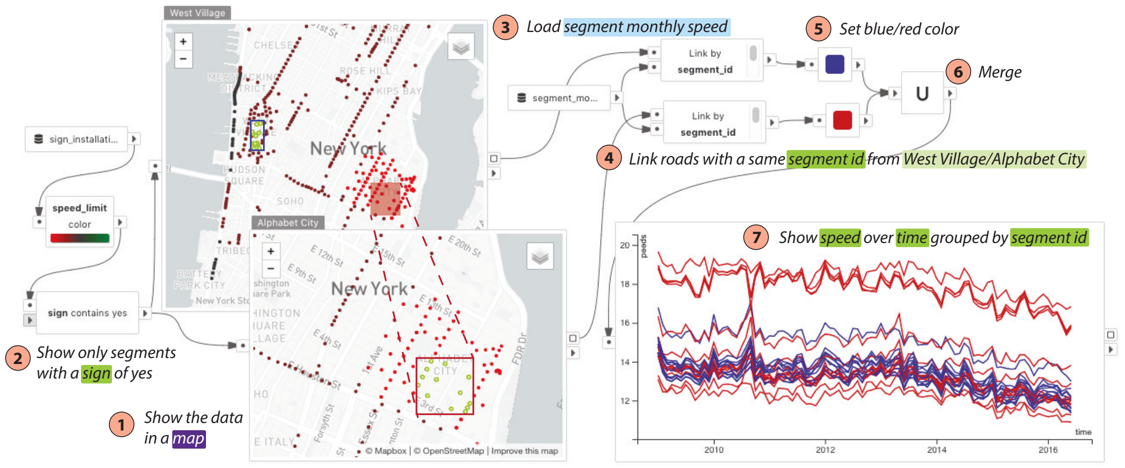
FlowSense: A Natural Language Interface for Visual Data Exploration within a Dataflow System

Bowen Yu and Cláudio T. Silva *Fellow, IEEE*



Load segment monthly speed
Set blue/red color
Merge
Link roads with a same segment id from West Village/Alphabet City
Show speed over time grouped by segment id
Show only segments with a sign of yes
Show the data in a map
West Village
Alphabet City
sign_installati...
speed_limit
sign contains yes
segment_mo...
Link by segment_id
U

Fig. 1. Using FlowSense for a comparative study on the street speed changes between two slow zones: West Village (blue) and Alphabet City (red). The analysts start by drawing locations of speed limit signs, which appear as dots with speed limits encoded by color. Selected speed limit signs are interactively linked with the line chart that shows the changes of average vehicle speed over time on corresponding streets and areas. All diagram elements are created via FlowSense. The NL queries shown are executed in the numbered order. FlowSense processes the rich dataflow context and allows the user to reference dataflow elements at different specificity levels, *e.g.* with node types, node labels, or implicitly.

Abstract— Dataflow visualization systems enable flexible visual data exploration by allowing the user to construct a dataflow diagram that composes query and visualization modules to specify system functionality. However learning dataflow diagram usage presents overhead that often discourages the user. In this work we design FlowSense, a natural language interface for dataflow visualization systems that utilizes state-of-the-art natural language processing techniques to assist dataflow diagram construction. FlowSense employs a semantic parser with special utterance tagging and utterance placeholders to generalize to different datasets and dataflow diagrams. It explicitly presents recognized dataset and diagram utterances to the user for dataflow context awareness. With FlowSense the user can expand and adjust dataflow diagrams more conveniently via plain English. We apply FlowSense to the VisFlow subset-flow visualization system to enhance its usability. We evaluate FlowSense by one case study with domain experts on a real-world data analysis problem, and a formal user study.

Index Terms—Natural language interface, dataflow visualization system, visual data exploration.

1 Introduction

Natural language interfaces (NLI) for data visualizations seek better usability of visualization solutions by introducing natural language (NL) query support. Compared with visualization systems that only support traditional mouse/keyboard interactions, systems with NLI require less prior knowledge on their functionality and usage details to work with. Latest research has progressed in visualization-oriented NLIs [25,29,44]. Most of these interfaces present a single visualization answer that can be interacted with (possibly with a few auxiliary views and widgets). The user does not have the opportunity to specify the relationships between multiple visualizations. However, practical data analysis tools often have multi-view linked visualizations, for which the design of an NLI becomes more challenging.

• Bowen Yu is with New York University. E-mail: bowen.yu@nyu.edu.
• Cláudio T. Silva is with New York University. E-mail: csilva@nyu.edu.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

Dataflow visualization systems (DFVS) have been proposed to achieve larger analytical flexibility [15,39,51]. These general-purpose visualization toolkits allow the user to draw a dataflow diagram that composes system modules to process and visualize data. It has been shown that DFVS can help build data analysis environments with multi-view linked visualizations that adapt to different domains [30]. Despite the flexibility, a DFVS often has higher learning overhead, due to its dataflow complexity, than a bespoke visualization application in which system components have pre-defined connections. The user must be proficient with the underlying DFVS modules to effectively use it.

In this work we propose FlowSense, a novel NLI that seeks to benefit both from the usability of NL and the analytical flexibility of DFVS. FlowSense uses semantic parsing to support NL queries that manipulate multi-view visualizations produced by a dataflow diagram. The NL capability may help reduce the overhead of learning dataflow and simplify the interactions of dataflow diagram construction. The FlowSense input box utilizes special utterances tagged by the underlying parsing algorithm to provide the user with real-time feedback of what the system sees and understands. To demonstrate the application of FlowSense, we build it on top of the recent dataflow system VisFlow proposed by Yu et al. [57]. We choose VisFlow because it focuses on creating linked visualizations that have good interactivity and support brushing and linking, which are two essential aspects of visual data exploration. With the integration of FlowSense, dataflow diagram editing becomes more intuitive in VisFlow, and consequently the user can use the DFVS more efficiently. The contributions of this work are summarized as follows:

1) We propose FlowSense, a novel NLI for visual data exploration within a DFVS. FlowSense uses NL to reduce the dataflow learning overhead, while taking advantage of the flexibility of a DFVS.
2) We exemplify a generalizable approach of applying state-of-the-art semantic parsing techniques to create a grammar that is tailored for a DFVS. In particular, FlowSense employs special utterance tagging and utterance placeholders to be aware of the dataflow context, and make its grammar independent of datasets and tasks. The identified special utterances are presented interactively as the user types the query. Such design echoes the underlying parsing state to the user. It not only helps the user understand the query semantics behind the scene, but also is useful for identifying errors and resolving ambiguity.
3) We demonstrate that FlowSense is able to support NL queries for the majority of diagram editing operations in VisFlow. We showcase the application of FlowSense by a case study with domain experts on studying the traffic speed reduction based on NYC taxi trip data. We further conduct a formal user study to evaluate the proposed NLI. We measure the task completion time, collect user feedback, and analyze the query logs to identify the strengths and weaknesses of FlowSense.

Details on the FlowSense grammar and its implementation can be found in the appendix and the FlowSense GitHub repository[1].

2 Related Work

2.1 Dataflow Visualization System (DFVS)

Dataflow systems enable the user to configure system functionality by drawing a dataflow diagram that defines how the system modules interact with each other. While dataflow systems are effective in fields other than data visualization such as computational workflow design [2, 4, 53], we focus on dataflow systems for visualization purposes in this section. Previous DFVS have demonstrated the effectiveness of using dataflow to render scientific data [28, 39, 51] and manage volume rendering pipelines [15, 37]. Dataflow systems that pass only data subsets (versus program method arguments) yield simpler dataflow diagrams and lower learning overhead [42, 43]. ExPlates [30] and VisFlow [57] present embedded visualizations in their dataflow, and focus on interactive information visualization. Most dataflow systems support diagram editing in a drag-and-drop manner. However, it is observed that even with drag-and-drop interfaces, users may often have difficulty in translating their intention to system operations [27]. In this work we design FlowSense to further simplify dataflow diagram construction, so that the user can intuitively use dataflow and make the most of a DFVS's analytical capability. In particular, we build FlowSense for VisFlow, as its subset flow model supports many of the low level visual data analysis tasks [12, 45], such as characterizing distribution, finding extremum, etc.

2.2 NLI for Data Visualization

Extensive research has been devoted to NLIs for decades. These interfaces address NL queries that otherwise have to be translated to formal query languages, *e.g.* SQL. A few examples are the interfaces for querying XML [33], entity-relational database [13, 55], and sequence translator to SQL [58]. NLIs for data visualizations answer the queries by presenting visual data representations. Compared with other interfaces that simply return a numerical answer or a set of database entries, visualization NLIs present results that are more human-readable. Cox et al. [19] design the Sisl service within the InfoStill data analysis framework. The service asks a series of NL questions to complete an unambiguous query. The Articulate system [49] uses a Graph Reasoner to select proper visualizations to answer a query. DataTone [25] addresses query ambiguity by showing ambiguity widgets along with the main visualization so that the user is able to switch to desired alternative views. Eviza [44] and Evizeon [29] further improve the user experience by allowing for conversation-like follow-up questions. Fast et al. [22] propose a conversational user interface called Iris that may perform analytical tasks and plot data upon requests in dialogues. Kumar et al. [31] also propose a dialogue system for visualization. Orko [47] is an NLI designed for visual exploration of network data. Dhamdhere et al. [21] design Analyza that provides database-based NL query and visualizations. Srinivasan et al. [48] provide a summary and comparison of the majority of these NLIs. Several commercial tools integrate NLIs. IBM Watson Analytics [3] and Microsoft Power BI [5] provide a list of relevant data and visualizations to an NL question, from which the user may choose to continue an analysis. Wolfram Alpha [10] supports knowledge-based Q&A and is able to plot the results. ThoughtSpot [8] enables interactive search from a relational database, and provides multiple types of visualizations for the database. The design of NLIs for data visualization has two challenges: First, modern natural language processing (NLP) techniques cannot yet understand well arbitrary NL input due to the complex nature of NL. User queries are apt to be free-form and ambiguous; Second, choosing a proper visualization to answer an analytical question is non-trivial as there can be multiple possible visual representations [35].

2.3 Comparison with Other NLIs

FlowSense makes a distinction from the other interfaces as it is to our best knowledge the first NLI to address a dataflow context. We set the scope of FlowSense to focus on assisting dataflow diagram construction, rather than to directly answer free-form analytical questions or seek a best visualization for a given query. We believe such an approach is beneficial in several aspects:
Capability: The analytical capability of FlowSense is rooted in the design of the DFVS. The outcome of FlowSense is a complete, interactive, and iterative visual data exploration process supported by the DFVS, rather than a single visualization that only answers one particular query as in other interfaces. Dataflow also naturally preserves analysis provenance [24], allowing the user to frequently revisit and reassess the current workflow. The diagram created by FlowSense explicitly keeps the user's preference and intention from previous queries, which must otherwise be maintained by a model behind the scene [25, 44].
Usability: FlowSense integrates real-time presentation of tagged special utterances in the interface that reflect the state of the underlying semantic parser, and help the user understand the context of the dataset and dataflow loaded in the system. This is a novel design that facilitates the user's understanding of the behavior of the NLI, as in most other NLIs the parsing feedback is only given after the query is submitted. The auto-completion suggestions also present special utterance tags so that the user may better understand the expected query components. Consequently, FlowSense may reduce the number of interactions required to construct a dataflow diagram. Our case study and user study (Sect. 5) show that FlowSense improves the DFVS usability. Its convenience is desirable by both novice and experienced VisFlow users. Besides, the DFVS is able to recover from errors more easily as the user always has full control over the system. However in other interfaces the user has to mostly rely on the behavior of the NLI and can hardly make corrections in case of misinterpretation.
Feasibility: The scope of assisting dataflow diagram construction is well defined and practicable. Even state-of-the-art NLP techniques have limited success in understanding an arbitrary query. Because each query is expected to update dataflow diagram and the user decides what the system should do and what visual representation to apply, FlowSense can produce more expected results and give better user experience under a well-defined scope. The mixed-initiative design mitigates the ambiguity problem. The DFVS users in our case study and user study are all able to understand the scope of FlowSense and use FlowSense effectively.

2.4 Semantic Parsing

FlowSense uses semantic parsing to process NL input and map user queries to VisFlow functions (Sect. 3.1). It depends on a pre-defined grammar that captures NL input patterns. A semantic parser recursively

[1]https://github.com/yubowenok/flowsense

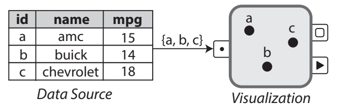
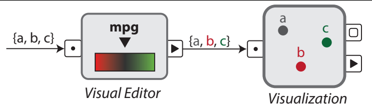
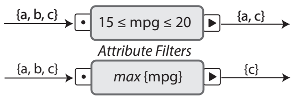
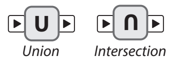
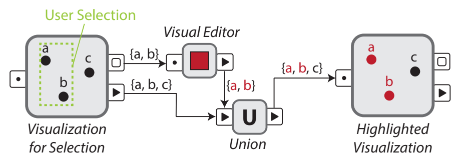
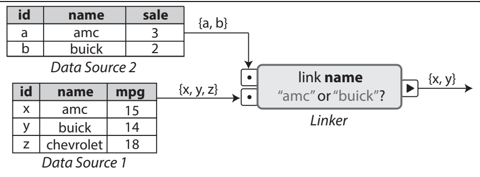
#	Function	Sample Queries	Description	Sample Sub-Diagram
A	Visualizing	*Show a scatterplot of mpg and horsepower*	Present the data in a visualization	
B	Visual Encoding	*Encode mpg by red green color scale*	Map data attributes to visual channels	
C	Filtering and Finding Extremum	*Find all cars with mpg between 15 and 20; List five cars with maximum mpg*	Filter data items and locate extremums and outliers	
D	Subset Manipulation	*Merge the cars with those from the scatterplot*	Refine and identify interesting subsets	
E	Highlighting	*Highlight the selected cars in a parallel coordinates plot*	View the characteristics of one subset among its superset or another subset	
F	Linking	*Link the cars with a same name from the sales table*	Extract primary keys from one table and find their correspondence from another (heterogeneous) table	

Table 1. Six major categories of VisFlow functions. These sub-diagrams are frequently used to compose more sophisticated diagrams that address analytical tasks. FlowSense aims at mapping NL input to one of these functions. The illustration only shows one possible sub-diagram from each category and does not exhaustively list all the possible sub-diagram variations of the function options. In practice the user can specify the function options via NL, *e.g.* visualization type, filter type, etc. Combinations of functions may apply.

expands the variables in the grammar to match the input query and can interpret the input based on the rules applied and the order of their application [16]. At a high level, the mapping performed by FlowSense can also be considered a classification task and addressed by classification algorithms [11]. However we prefer semantic parsing because most classification approaches are supervised algorithms that require a large corpus of labeled examples. Such training data are not available for DFVS. Besides, compared with deep learning methods [20, 26], semantic parsing does not require heavy computational resources.

The FlowSense semantic parser is implemented within the Stanford SEMPRE framework [40] and CoreNLP toolkit [36]. The CoreNLP toolkit integrates a comprehensive set of NLP tools including the Part-of-Speech (POS) tagger, Name-Entity-Recognizer (NER), etc. A POS tagger identifies roles of words in a sentence, e.g. verb, preposition, adverb. The SEMPRE framework employs a modular design in which different types of parsers and logical forms can be easily plugged-in. The framework can quickly be adapted for domain-specific parser design [52]. We apply SEMPRE together with CoreNLP to the DFVS domain. In particular, the FlowSense parser utilizes the POS tags produced by CoreNLP for processing special utterances and grammar matching. The FlowSense grammar expects words with certain POS tags to appear in query parts.

3 Semantic Parser

In this section, we define the building blocks of the semantic parser: the VisFlow functions that can be specified by NL, the definition of the parsing grammar, and the general query pattern the parsing algorithm expects. For concept illustration we use the Auto MPG dataset[2] throughout the paper, which has information about cars in 9 columns, including mpg, horsepower, origin, etc.

[2]http://archive.ics.uci.edu/ml/datasets/Auto+MPG

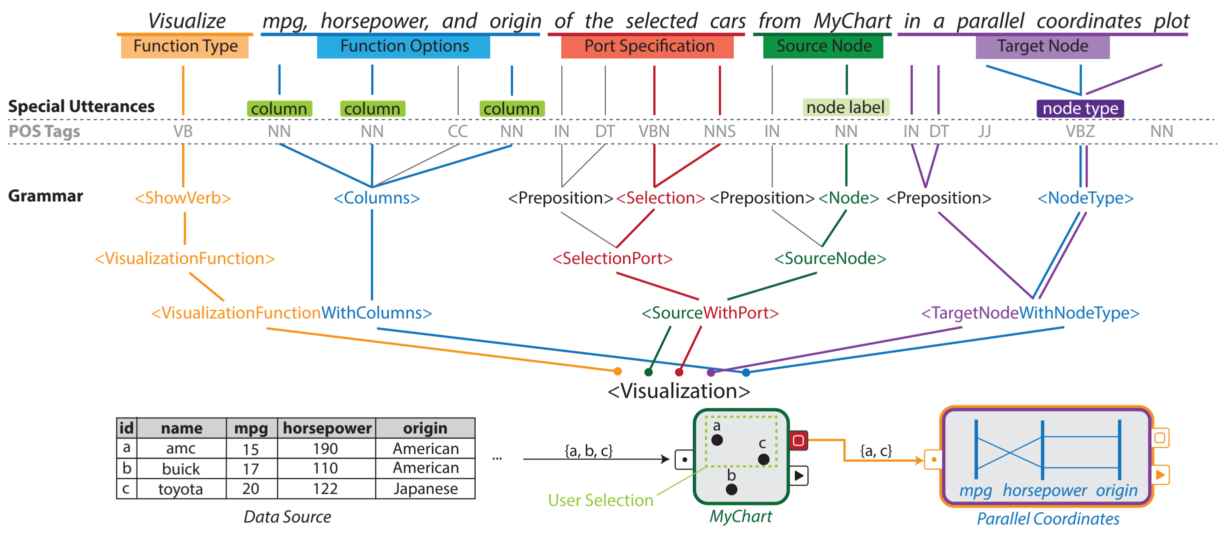
3.1 VisFlow Functions

To create an NLI for VisFlow, we first studied a sample diagram set that includes 60 dataflow diagrams created by 16 VisFlow users from their recorded VisFlow sessions. These diagrams cover a wide range of VisFlow usage scenarios and deal with various types of datasets. We identify a set of frequently appearing sub-diagrams and categorize them into six major categories as listed in Table 1. The construction of these sub-diagrams are defined as the *VisFlow functions*. By implementing the VisFlow functions, FlowSense essentially supports the building blocks of visual data exploration in VisFlow so that analyses rendered by VisFlow native interactions can be carried out with FlowSense. These functions also reflect the fundamental analytical activity defined in information visualization task taxonomies [12, 45]. Table 1 explains the usage of each VisFlow function and shows several sample queries.

In addition to the six major categories, FlowSense also supports many utility functions such as adding/removing dataflow nodes/edges, undo/redo, loading datasets, etc. Though these functions also enhance the usability of the system, we omit them here as they are indirectly related to visual data analysis.

3.2 Dataflow Context and Special Utterances

It is important to make the semantic parser aware of the dataflow context, such as the dataset loaded and the nodes in the dataflow diagram. The FlowSense grammar consists of a special group of tokens called the *special utterances*. Special utterances are words that refer to entities in the dataset or the dataflow diagram. They are the arguments and operands of VisFlow functions. FlowSense recognizes node types,



id	name	mpg	horsepower	origin
a	amc	15	190	American
b	buick	17	110	American
c	toyota	20	122	Japanese

Fig. 2. An example FlowSense query and its execution over the Auto MPG dataset. The derivation of the query is shown as a parse tree in the middle. The sub-diagram expanded by the query is illustrated at the bottom. The five major components of a query pattern are underscored. Each component and its relevant parts in the parse tree and the dataflow diagram are highlighted by a unique color. The result of executing this query is to create a parallel coordinates plot of columns mpg, horsepower, and origin, with its input coming from the selection port of the node labelled MyChart.

table column names, diagram node labels, and dataset names as special utterances. For the query shown in Fig. 2, FlowSense tags "mpg", "horsepower", and "origin" as columns, "MyChart" as a node label, and "parallel coordinates" as a node type.

The special utterances identified by FlowSense are shown in colored tags in the FlowSense input box (Fig. 3). Each different color represents a one special utterance type: green for data column, light green for node label, purple for node type, and light blue for dataset name. The colors are applied consistently throughout the user interface. In the rare case that the same word can be a special utterance of different types, or not a special utterance (type "none"), the user is able to disambiguate using a dropdown (Fig. 3(b)).

3.3 Grammar

FlowSense applies a semantic parser to map an NL query to one of the VisFlow functions based on an elaborate grammar designed for these functions. The grammar is context-free [46] and formally defined as a 4-tuple $G = (V, \Sigma, R, S)$. V is a finite set of variables. Σ is a finite set of terminals. A terminal represents an English word or phrase. R is the rule set that defines how a single variable matches an ordered list of terminals and variables (possibly itself in a recursive rule). Below is an example rule:

$$\langle \textit{Visualization} \rangle \rightarrow \langle \textit{ShowVerb} \rangle \ \langle \textit{Columns} \rangle \ \textit{in} \ \langle \textit{VisualizationType} \rangle$$

In this rule, $\langle \textit{Visualization} \rangle$ is a high-level variable that matches a query that requests a visualization. $\langle \textit{ShowVerb} \rangle$ matches a verb that has a meaning similar to "show". $\langle \textit{Columns} \rangle$ matches one or more columns from the data. $\langle \textit{VisualizationType} \rangle$ stands for a phrase that describes a visualization metaphor such as scatterplot or parallel coordinates. The token "in" is a terminal symbol that comes from the NL input directly. The example rule above is simplified for the convenience of explanation. In practice, a rule often matches against generic variables rather than a specific word. S is the start variable that expands to other variables to match the whole query.

The grammar of the FlowSense semantic parser attempts to derive an input query by recursively searching for all possible matches (up to a preset limit) of the grammar rules. This procedure is called *derivation* [16]. FlowSense uses the semantic parsing implementation from SEMPRE. It also uses the Stanford CoreNLP [36] toolkit that is built into SEMPRE for special utterance tagging. The variables and rules (*i.e.* SEMPRE formulas) are defined in SEMPRE grammar files.

3.3.1 Special Utterance Placeholders

The FlowSense grammar consists of static grammar rules and the special utterance placeholders. The special utterance placeholders are at runtime dynamically replaced by their corresponding dataflow elements. Therefore, the FlowSense semantic parsing is independent of the dataset, the dataflow diagram, and the analytical tasks. The rules are generalizable across domains: No new rules need to be created when the system switches to new datasets or tasks.

For example, FlowSense uses the generic variable $\langle \textit{column} \rangle$ in its grammar as a special utterance placeholder. At runtime, a real column name (*e.g.* "mpg") is automatically extracted from the dataset. FlowSense identifies column names on the fly as the user types the query. "mpg" would show up as a tagged column, and then matched with $\langle \textit{column} \rangle$ by the parser. A reverse mapping is performed from the placeholder to the particular column after query parsing so that the system may operate on that column.

Using special utterances in the grammar has several benefits. First, special utterances enable VisFlow functions to operate on elements that are important for dataflow diagram editing and visual data exploration. Second, it makes the grammar set small as rules may be written with generic variables rather than specific dataset or diagram content. Last but not least, the real-time tagging of special utterances provides important feedback to the user about what operations are available in the system and how the NLI interprets the query.

3.3.2 Derivation Ambiguity

It is possible to have ambiguity when multiple possible query derivations exist, which can be defined as syntactic ambiguity [25]. For example, FlowSense uses wildcard variables to match general *table row* references. Over the Auto MPG dataset, the token "cars" from "*Show a plot of cars*" describes the user's understanding of data entities but should be only treated as table rows from the NLI perspective. Meanwhile, the token "horsepower" from "*Show a plot of horsepower*" is a special utterance and should be treated as a column to visualize. Therefore a wildcard rule that matches "cars" as table rows may also match "horsepower", resulting in the second query getting improperly executed. We could handle this case by creating a wildcard variable that rejects a special utterance token. Nevertheless, such design could lead to a larger number of variables and rules in the grammar, which are harder to maintain and develop. Therefore we choose to resolve certain syntactic ambiguity in the parsing phase with supervised learning on a weight vector $\mathbf{w} \in \mathbb{R}^d$ that gives the probability of derivations based on input utterances. Stochastic gradient descent (SGD) is employed to optimize the multiclass hinge loss objective [50], as introduced by Liang et al [34] in the SEMPRE framework. The objective is given by:

$$\min_{\mathbf{w}} \sum_{(x,y)} \max_{y'} \{\mathbf{w} \cdot feature(x,y') + penalty(y,y')\} - \mathbf{w} \cdot feature(x,y)$$

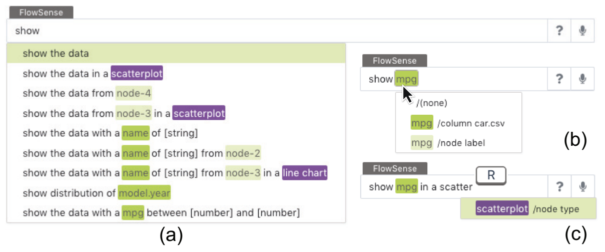
In the above, x is the input query, y is the preferred derivation, and y' is the chosen derivation by the parser. The pair (x,y) is chosen over all training data. The feature of a derivation, $feature(x,y)$, maps the pair (x,y) to a d-dimensional space and is determined by the applied rules in the derivation. $penalty(y,y')$ is 0 if $y = y'$ and 1 otherwise. The objective function has a penalty for possible choices of incorrect predictions that are within a margin of one from the correct predictions. The parser fits the training examples by giving intended derivations higher probability so that they are preferred in case of ambiguity. In particular, the rule that expands to a column special utterance will be preferred over a rule that expands to a wildcard. Note that we only apply this training to facilitate the simplicity of the FlowSense grammar and reduce the number of required rules. The training cannot address the ambiguity in natural language itself at large. We were able to use a small training set of fewer than twenty examples to guide the preferred derivation in case of syntactic ambiguity for a rule set of around 500 rules. This is feasible because the FlowSense rules are independent of data and dataflow diagrams. The training set only needs to guide the semantic parser to focus on certain important grammatical features, such as emphasizing special utterances or word proximity.

3.4 Query Pattern

The main goal of FlowSense is to support progressive construction of dataflow diagrams. We studied the creation process of the VisFlow diagrams in our sample diagram set and empirically identified a common pattern with five key *query components* that all VisFlow functions may contain: *function type*, *function options*, *source node(s)*, *target node(s)*, and *port specification*. This pattern is illustrated in Fig. 2 with a sample query "*Visualize mpg, horsepower, and origin of the selected cars from MyChart in a parallel coordinates plot*". In this query, the verb "visualize" implies applying a visualization function. The three columns "mpg, horsepower, and origin" indicate the options (*i.e.* what to visualize) for the visualization function. The phrase "from MyChart" tells the system the location of the data to be plotted and provides source node information. The phrase "in a parallel coordinates plot" indicates a new visualization node with the given visualization type is to be created as the target node. As VisFlow explicitly exports interactive data selection from visualization nodes, the phrase "selected cars" is a port specification that further describes that the user wants to visualize the selection from MyChart and the new visualization node should be connected to the selection output port of MyChart.

The grammar of FlowSense includes a variable hierarchy that matches the five key components of an NL query. Fig. 2 illustrates the parse tree that derives the sample query. The variables involved in the derivation are shown in the parse tree, in which rule expansions are bottom-up. A variable may carry information for multiple query components. We design a broad set of variables and rules that are able to not only accept queries with a particular component order, but also their different arrangements. For instance, "Show mpg and horsepower in a scatterplot" is equivalent to "Show a scatterplot of mpg and horsepower". They both can be accepted by FlowSense. FlowSense is also able to derive multiple functions from one single query and execute their combination, *e.g.* "*Show the cars with mpg greater than 15 in a scatterplot*" infers both visualization and filtering functions.

A query may not necessarily contain all the five components explicitly. For example, the user may simply say "*Show mpg and horsepower*" without mentioning any source node or target visualization type. FlowSense may automatically locate source and target nodes in its query pattern completion phase (Sect. 4.3). An NL query may also contain implicit information, *e.g.* "*Find cars with maximum mpg*" intends to perform data filtering to search for cars with the largest mpg value. The use of a filter is identified by function classification in the query execution phase (Sect. 4.2).


FlowSense
show
show the data
show the data in a scatterplot
show the data from node-4
show the data from node-3 in a scatterplot
show the data with a name of [string]
show the data with a name of [string] from node-2
show the data with a name of [string] from node-3 in a line chart
show distribution of model.year
show the data with a mpg between [number] and [number]

FlowSense
show mpg
/(none)
mpg /column car.csv
mpg /node label

FlowSense
show mpg in a scatter
scatterplot /node type

Fig. 3. The FlowSense input box and its query and token auto-completion. Special utterances are identified by unique colors: table columns (green), node types (purple), node labels (light green). (a) Suggested queries from query auto-completion; (b) Dropdown for handling tagging ambiguity: "mpg" are both column name and node label; (c) Special utterance token completion: "scatterplot" is presented after the letter "R" is entered.

3.5 Auto-Completion

The usability of an NLI is closely related to its discoverability. It is desirable that when the query is partially completed, the system is able to provide hints or suggestions to the user about valid queries that include the partial input. This has been a requested feature in prior NLI user studies [25]. We therefore develop an auto-completion algorithm in FlowSense to enhance its usability and discoverability. When the user types a partial query and pauses, the system triggers query auto-completion automatically. The auto-completion may also be invoked manually with a button press. Fig. 3(a) shows the auto-completion suggestions in the FlowSense input box.

Auto-completion has been implemented in other visualization NLI, such as Eviza [44]. Eviza applies a template-based auto-completion, in which the system attempts to align user input to available templates. Here we take a similar approach by creating a set of query templates with around one hundred queries. Upon an auto-completion request, the algorithm searches through all possible textual matches between the user's partial query and a prefix of some template. All matched queries are then sent to the FlowSense parser for evaluation. If a query is accepted, it becomes an auto-completion candidate. Some of the queries contain value placeholders and the user is expected to fill in those values ([string], [number] in Fig. 3(a)).

We also design a token auto-completion algorithm that matches the partially typed word against all available special utterances. This helps speed up query typing with respect to the dataflow context. The user may use the tab and arrow keys to select token completion candidates as in a programming IDE. For example, when the user types "scatter" it can be completed to the available visualization type "scatterplot" (Fig. 3(c)). Token auto-completion reduces the typing workload and helps remind the user of the DFVS capability and the current dataflow diagram elements.

4 Query Execution

FlowSense is built as an extension to VisFlow. The user may activate the NLI at any time while working with the DFVS. The user may either type the query in the input box or use the speech mode that is implemented with HTML5 web speech API. In this section we introduce the query execution workflow as depicted by Fig. 4.

4.1 Special Utterance and POS Tagging

Special utterances have remarkable roles in executing a VisFlow function. Their tagging is performed on the fly when the user types the query. For typo and naming tolerance, FlowSense employs approximate matching and checks each k-gram in the query (where k may range from 1 to the maximum special utterance word length) against all special utterances using case-insensitive Levenshtein distance [32, 38]. We divide the distance over the string length and use the ratio to mitigate the fact that longer strings are more prone to typos. We find a k value of 2 or 3 and a ratio threshold of 0.2 work well in practice.

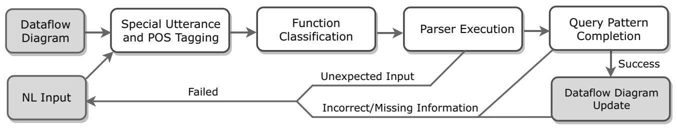


Fig. 4. Steps of FlowSense query execution. In case the grammar rejects the input, or there is no valid way to complete query components, a failure is returned to the user.

In addition to recognizing special utterances, FlowSense also performs POS tagging on the query with CoreNLP. Each token receives a POS tag as shown in Fig. 2. POS tags are used to generalize the FlowSense grammar. For example, many prepositions can be used interchangeably, *e.g.* "selection *of* the plot" is equivalent to "selection *from* the plot". Instead of having one rule for every preposition, the grammar uses a generic variable that matches any preposition. POS tagging helps analyze the basic semantic structure of a query.

4.2 Function Classification

FlowSense uses keyword classification to identify the semantic meaning of words in the NL query and uses this information to decide a proper VisFlow function to execute. For instance, the verb "show" is a synonym of "visualize", "draw", etc. These words indicate the intention to create a visualization. Meanwhile, "find" may implicitly specify a data filtering requirement and is similar to "filter". We compute the Wu-Palmer similarity scores [54] between words and use the measured scores to classify words in the NL query that have close meaning to a set of pre-determined VisFlow function indicators. The implementation of the similarity scores is based on WordNet [23] and NLTK [6].

4.3 Query Pattern Completion

After the parser identifies the existing key components of a query, FlowSense attempts to fill in the blanks where information is missing using default values or the diagram editing focus.

4.3.1 Finding Default Values

Query components may be completed using default values. Function options may have defaults. For instance, FlowSense automatically chooses two numerical columns to visualize in a scatterplot triggered by a simple query "*Show a scatterplot*". Note that within a DFVS decisions like this can easily be changed by the user. So FlowSense does not necessarily need to make a best guess. Similar decisions include completing port specification. By default FlowSense applies the newly created filter to all the data a visualization node receives, rather than the data subset interactively selected in the visualization. Sometimes the default values could even be empty. A query like "*Filter by mpg*" results in FlowSense creating a range filter on the mpg column with no filtering range given (*i.e.* a no-op filter placeholder). The user can then follow up and fill in the filtering range via the DFVS interface.

4.3.2 Finding Diagram Editing Focus

Whenever the user expands the dataflow diagram there always exists an editing focus, though sometimes the focus is implicit. For example, when the query contains a phrase like "from MyChart", the focus (*i.e.* the source node of the query) is explicitly given. However, users tend to neglect the source or target nodes in their queries, especially when there is a sequence of commands that together complete a task. When a query does not have explicit focus, FlowSense derives the user's implicit focus based on user interaction heuristics. We compute a focus score for every node X by:

$$score(X) = activeness(X,t) + \alpha\left(1 - \frac{1}{1+e^{-(distanceToMouse(X)/\gamma-\beta)}}\right).$$

The activeness of X is re-iterated upon every user click in the system:

$$activeness(X,t) = activeness(X,t-1)/2 + click(X,t),$$

where $click(X,t) = 1$ if the t-th click is on X and 0 otherwise. This definition measures how actively a user focuses on a node by how many times she recently clicks on it, as well as how close it is to the mouse cursor. The activeness derived from user clicks decreases exponentially over time, while the closeness to mouse dominates under a small distance with a shifted sigmoid function[3]. We find the parameters $\alpha = 2, \beta = 5, \gamma = 500$ achieve good result. FlowSense chooses the node with the highest focus score to be the diagram editing focus. If multiple source nodes are required (*e.g.* in a merge query), FlowSense looks at the nodes in the order of their decreasing focus scores.

The focus may also be required by node type references. For instance, the user may input "*show the data from the scatterplot*", in which case "scatterplot" is a reference by node type that describes a scatterplot node existing in the dataflow diagram. In case of a tie during the node type search, *e.g.* there are multiple scatterplots in the diagram, the nodes with higher focus scores are chosen.

4.3.3 Query Completion Ambiguity

There may be multiple syntactically correct ways to execute a same query. Consider the query "*Show the cars with mpg greater than 15*" applied on a visualization node. From the grammar perspective the parsed outcome has no ambiguity: Apply an attribute filter and visualize the result. However, there are two ways of execution: One is to create a filter and then visualize the filtered cars in a new visualization; Alternatively we may apply the filter on the input of the current visualization so that the existing visualization shows only the filtered cars. Both can be desired under some circumstances. FlowSense has the default behavior that prefers filtering the input when the source node is a visualization, which we find empirically more intuitive. Such ambiguity can often be resolved with a slightly refined query, *e.g.* "*Show the cars with mpg greater than 15 from the plot*", which would explicitly indicate that the filter should be created from the output of the existing visualization.

4.4 Diagram Update

Once a query is successfully completed, FlowSense performs the VisFlow function(s) with the given function options. This typically results in the creation of one or more nodes, *e.g.* the visualization function creates one plot while the highlighting function creates three nodes (Table 1). FlowSense may also update existing nodes without creating any new nodes, *e.g.* when the user only changes rendering colors. Additionally, a query may operate on multiple existing nodes at once, *e.g.* linking and merging two tables create edges between two nodes. Operating on multiple nodes together helps simplify dataflow interaction, as these operations previously require multiple drag-and-drop interactions.

After new nodes and edges are created, the diagram may become more cluttered. FlowSense locally adjusts the diagram layout after each diagram update. We use a modified force-directed layout from the D3 library [1] that works on the vicinity of the current diagram editing focus. We extend the force to take rectangular node sizes into account so that larger nodes such as embedded visualizations have stronger repulsive force for avoiding node overlap. User-adjusted node positions are remembered by the system, and the layout algorithm avoids moving nodes that have been positioned by the user. Currently FlowSense does not look for an optimal dataflow layout. We leave layout improvement [14] for future work.

4.5 Error Recovery

There are several types of potential errors in executing a query:
(1) The query cannot be accepted by the grammar. For example, out-of-context input ("*What time is it now*") and unsupported functionality ("*Split the data into two halves*") would receive grammar rejection;
(2) The query is grammatically correct but invalid based on the dataflow context, possibly due to incorrect references of dataset and diagram elements. For example, the user may attempt to show data from a non-existing node, *e.g.* asking to "*Highlight the selected cars from the scatterplot*" when there is no scatterplot in the dataflow. Such errors are captured at the diagram update step.

[3]See the appendix for more explanation on the characteristics of the diagram editing focus heuristics.

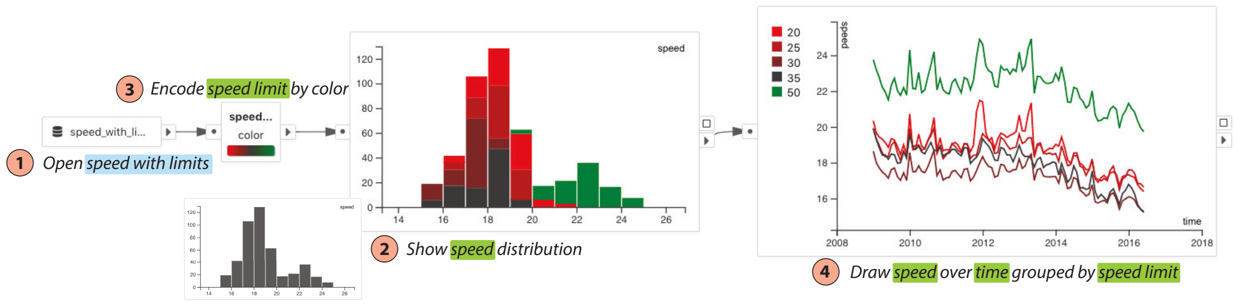

3 Encode speed limit by color
1 Open speed with limits
speed_with_li...
speed...
color
speed
2 Show speed distribution
20
25
30
35
50
speed
time
4 Draw speed over time grouped by speed limit

Fig. 5. Using FlowSense to study the aggregated monthly average vehicle speed on NYC streets with different speed limits. The queries are applied in the numbered order. The result shows a histogram for speed distribution and a line chart for speed changes over time. Both charts use color encoding based on the speed limit of the roads. The smaller histogram snapshot shows the speed histogram without color encoding before step 3.

(3) The query is executed fully but does not meet the user's expectation. For example, "*Show the data*" by default creates a scatterplot but the user instead wants a heatmap, or "*Merge these two nodes*" merges an unexpected pair of nodes when "these" appears to be a vague reference (the system chooses two nodes with the highest focus scores).

Upon the first two types of errors the system displays a message and asks for a query correction. For the last type of error it is up to the user to adjust the dataflow diagram. Since the user is simultaneously using the underlying VisFlow DFVS while using FlowSense, she always has the flexibility to undo the FlowSense action or to make partial adjustments when the NLI does not yield exactly the desired outcome.

5 Evaluation

To evaluate the effectiveness of FlowSense, we describe one case study and conduct a formal user study.

5.1 Speed Reduction Study

We invite several users to try out the FlowSense prototype in different data analysis domains and analyze their usage of our NLI. In this paper we introduce one case study in which we work with two domain experts in person to address a practical research task using a comprehensive set of NL queries. The analysts are researching the city regulation issued on November 7, 2014 that reduces the default speed limit on all New York City streets from 30 MPH to 25 MPH. The data contain the estimated average hourly speed [41] for each road segment in Manhattan from January 2009 to June 2016. The speed estimation was performed over the TLC yellow taxi records [9] that only have pickup and dropoff information. The analysts are familiar with the data, and the visualizations to be created are similar to the visualizations they previously generated for the project using Tableau [7]. However they have no prior experience with either VisFlow or FlowSense. We met the analysts in person and first introduced VisFlow and FlowSense in a 30-minute session. Then we guided the analysts through how FlowSense can be used to create visualizations to study the speed reduction. We observed in this study that almost all the analysts' visualization requests (excluding those that exceed the scope of the VisFlow subset flow) can be effectively supported by FlowSense. Here we summarize the NL queries that are applied for the speed reduction study.

Initially, the analysts would like to look at the speed reduction impact at a larger scale. They first load a pre-computed speed table (Fig. 5(1)) with the FlowSense data loading utility function (the analysts know the dataset name). The table contains the monthly average speed aggregated by the speed limits of the streets. The analysts ask the system to present a histogram of speed by "*Show speed distribution*" (Fig. 5(2)). The first histogram has no color encoding but the analysts are able to immediately add a color scale by "*Encode speed limit by color*". FlowSense inserts a color mapping node with red-green scale at the input of the histogram (Fig. 5(3)). The histogram shows the street groups with higher speed limit in green, and lower speed limit in red. To view the speed changes over time, the analysts use the query "*Draw speed over time grouped by speed limit*" (Fig. 5(4)). The query result is a line chart showing average speed changes for different speed limit groups. The analysts observe that overall there is a speed reduction for each speed limit group that started around middle 2013.

Seeing the overall trend, the analysts move on to a comparative analysis between individual streets from two slow zones. They load and visualize a speed sign installation table in a map (Fig. 1(1)) by "*Show the data in a map*". This dataset has for each road segment in Manhattan its speed limit, geographical location, and whether the street has speed signs installed (signs are shown as dots in the map). As the slow zones mostly have speed signs installed, the analysts narrow down the data in the map by placing a filter on the "sign" column (Fig. 1(2)). The filtered map reveals two slow zone neighborhoods with densely located signs: Alphabet City and West Village. The analysts apply one map visualization for each zone for a comparison between the two zones. They label the two maps by the slow zone names and select a few streets from each zone (marked in the maps of Fig. 1). To study the speed changes of these selected streets, another table (named "segment monthly speed", also known to the analysts) that includes monthly average speed for each road segment is added to the diagram (Fig. 1(3)). The analysts then use the link queries to create a sequence of nodes that extract segment IDs from the selected streets and find their monthly average speed from the segment monthly speed table (Fig. 1(4)). Blue and red colors are assigned to the streets in West Village and Alphabet City respectively to visually differentiate them (Fig. 1(5)). The two groups of streets are then merged by a subset manipulation function (Fig. 1(6)). Note that the query "*Merge*" only has a single word. It works because the query completion of FlowSense automatically locates the recently focused color editors as the source nodes for this query. Finally, the two groups are rendered together in a speed series visualization (Fig. 1(7)), which compares the speed changes between the two groups of streets. As the visualizations produced by FlowSense are linked, the analysts can easily change the street selection in the maps to compare different groups of streets.

This case study demonstrates that FlowSense can be applied to a practical, comprehensive analytical task. The generated visualizations may guide the analysts towards further data analysis. The analysts participating in this study think FlowSense is helpful, especially since it exemplifies how to build VisFlow diagrams and facilitates their learning of the DFVS.

5.2 User Study

We conduct a formal user study to evaluate the effective of FlowSense together with the VisFlow framework. Through the user study we validate whether a user is able to smoothly apply FlowSense for dataflow diagram construction, and how well FlowSense responses meet the user's expectation. We design an experiment that introduces FlowSense and VisFlow to the participant and assigns analytical tasks to be solved within the system.

5.2.1 Experiment Overview

The user study is carried out in a fully automated manner using an online system with step-by-step instructions. The participants join the study using a web browser on their own machines. Participants may ask the experiment assistant for help and clarification via web chat or phone call during the experiment session.

We recruited 17 participants (11 male, 6 female, all with an age between 20 and 30) who work or study in the field of computer science.

12 participants have a data visualization background. 9 are graduate students, and the other 8 are professionals (software engineer, researcher, faculty). 3 participants have prior experience with VisFlow. No participants have prior knowledge about FlowSense. The participants are chosen to have a variety of specialities so as to represent potential DFVS users. The participant group includes visualization designers, data scientists, and software engineers who share data analysis interest but have different skill sets. The study is structured into two phases:

Tutorial Phase. The participant completes a tutorial of the VisFlow dataflow framework, and then a tutorial of the FlowSense NLI. After each tutorial, the participant is asked to complete the tutorial diagram to demonstrate familiarity with the introduced tool. Each tutorial is expected to take 10 to 20 minutes. After the tutorials there is an on-demand practice session with a flexible duration.

Task Phase. The participant explores an *SDE Test* dataset and constructs dataflow diagrams using FlowSense and VisFlow to answer questions about the data. The participant is encouraged to use FlowSense as much as possible. The usage of the NLI is not enforced because the goal of the NLI design is to improve the user experience of the DFVS, rather than to completely replace the traditional DFVS interactions (which is likely infeasible). The entire task phase is expected to take 30 to 60 minutes.

At the end of the study, the participant takes a survey to give comments and quantitative feedback about FlowSense and VisFlow.

5.2.2 Dataset and Tasks

The *SDE Test* dataset includes the test results of software development engineer (SDE) candidates stored in two tables. The first table describes the test results for each candidate. A test consists of answering several multi-choice questions selected by the system from a large question pool. Each question has a unique ID, a pre-determined difficulty, its supported programming language(s), and possibly a time limit. For each question, the candidate receives a result (correct, wrong, skipped, unanswered)[4]. The dataset also has a "TimeTaken" column that stores how much time a candidate took to answer a question. The second table includes background information about each candidate, such as the candidate's highest degree level, field of study, and institution. We give three analytical tasks about this dataset. The tasks are designed to reflect common tasks performed in visual data exploration:

(T1) Overview Task. The participant is asked to visualize the overview distribution of the question answering results, and figure out the total number of questions that were skipped, and the percentage of a question being answered correctly.

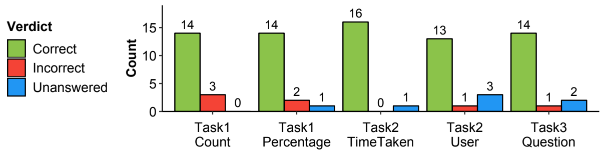
(T2) Outlier Task. The participant is first asked to find a candidate with an outlier background information value (who incorrectly entered the current year "2018" in place of his own information). Then the participant is asked to investigate a data recording discrepancy regarding the "TimeTaken" column: Some of the "TimeTaken" values are erroneously large numbers when a question is unanswered.

(T3) Comprehensive Task. The participant is asked to identify one question that Masters candidates answer significantly better than Bachelors candidates. This task requires comprehensive usage of the dataflow features: attribute filtering, brushing, and heterogeneous table linking.

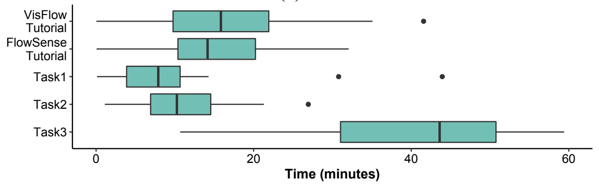
All the three tasks have definitive correct answers to ensure that participants explore the data and draw conclusions reasonably. Each user study session is logged with anonymous full diagram editing history. We analyze the study results based on task answers and completion time, comments and quantitative feedback, and NL query logs.

5.2.3 Task Completion Quality

Fig. 6(a) shows the verdict distribution of the participants' answers. It can be seen that the majority of the participants were able to come up with the correct answers to the tasks. Fig. 6(b) shows the completion time distribution for each step of the user study. It can be observed that the time taken for the tutorials and tasks are mostly as expected. Yet the time required for a task increases when the task involves heterogeneous tables and interactive data filtering to find solutions (T3). After reading



(a)



(b)

Fig. 6. (a) Verdict distribution of participant answers to each of the user study tasks. (b) Box plot of completion time for each user study step[4].

the user comments in the feedback, we believe this may be due to the fact that many participants are first-time VisFlow users and need to digest the concept of the VisFlow subset flow model. In particular, linking heterogeneous tables can be challenging to understand at first. However, most users were able to get the idea and formulate a solution. This is reflected by one of the feedback comments: "*The linker functions are confusing at first. But after experimenting with the tool for a while and getting to know how they work, things become easier.*" We believe such a learning curve is natural for DFVS.

5.2.4 Quantitative Feedback

We ask for feedback on six aspects regarding FlowSense (and also VisFlow[4]) in our survey. Each aspect is presented with a statement with a 1–5 Likert scale for the participant to express agreement (5) or disagreement (1). Table 2 lists the feedback for the FlowSense NLI. The quantitative feedback shows that most users were able to understand the scope of FlowSense, and apply it for dataflow diagram construction. The users were also asked to compare the NLI-assisted dataflow usage against their earlier experience in the tutorial phase with the standalone VisFlow framework. Twelve users agree (with a feedback score of at least 4) that FlowSense simplifies the diagram construction, and ten users agree that FlowSense speeds up the data exploration.

The feedback also reveals space for improving the NLI. In particular, it is unclear to most users how to update a rejected query to make it accepted. It may be helpful to design an algorithm that provides suggested corrections or changes to a failed query. However, this is technically challenging as changing minimally a query to fit it into the parse tree is algorithmically non-trivial. We would like to leave query correction suggestions for future work.

5.2.5 Query Log Analysis

To closely study where FlowSense does not accept a query, we conduct a manual walk-through of the rejected queries and categorize each rejected query by its reason of rejection. Overall, we analyzed 649 queries, out of which 421 were accepted by FlowSense. Excluding the 34 invalid and mistyped queries, the raw acceptance rate was 68.455%. We found some of the rejection issues straightforward to resolve: the requested functionality was not implemented, bugs in the query execution code, etc. We were able to fix those issues in a short iteration of the NLI implementation, resolving 34 "not implemented" queries and 18 software bugs. The improved acceptance rate would be 76.911%. In general, it requires systematic engineering efforts to thoroughly increase query coverage for the "not implemented" category, which is beyond the scope of this paper. The remaining non-resolved failures are summarized in Fig. 7 with their counts[5].

Some of those failures are more challenging to resolve. Specifically, FlowSense does not make logical inferences and deals only with the raw values in the data. If the user rephrases the query by natural language variation or implication (26 occurrences in Fig. 7), the query would be difficult to parse. The query "*Show only segments with signs*" is

[4]See the appendix for additional remarks and results of the user study.

[5]See the appendix for the detailed definition and examples for each category.

Aspect	Feedback
I understand what queries FlowSense may accept and execute.	1 3 9 4
The responses of FlowSense meet my expectations.	4 11 2
FlowSense simplifies dataflow diagram construction.	3 2 6 6
FlowSense speeds up my data exploration.	4 3 3 7
FlowSense helps me learn VisFlow features that I was not aware of.	1 1 4 4 7
When my query got rejected, I can figure out how to update it to let it be accepted.	1 3 9 3 1

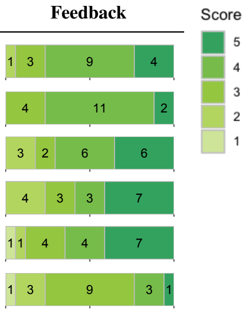


Table 2. FlowSense Survey Result. The feedback column shows the score distribution for each assessed aspect of the NLI. The numbers on the colored bars show the counts of the scores received. Darker green represents higher score.

more natural than that in Fig. 1(2). Yet FlowSense does not infer that a segment with a "sign" value of "yes" implies that it is a segment "with sign". In T3 the dataset has "HighestLevelOfEducation" as column name, but if the user mentions "degree", FlowSense does not know that it is equivalent. There needs to be additional knowledge base added to the system so that the NLI can detect concept equivalence, which is generally difficult to achieve. In a "composite" query, the user intends to perform several VisFlow functions in one query (*e.g.* creating nodes, applying filter, and assigning color together). It is difficult to write concise grammar rules to accept composite queries. In practice, by informing the users of these limitations, in most cases the issues can be circumvented via rephrasing the queries, *e.g.* composite queries can be split into smaller steps that are easier to parse and execute.

When an operation requested is not supported by the DFVS, a "not supported" failure arises, *e.g.* VisFlow without its data mutation extension[6] cannot aggregate and mutate data. When the special utterance tagging over-aggressively tags a non-special word, its placeholder fails to resolve, leading to a "tagging error". The user may use the token dropdown in the FlowSense input box to correct tagging mistakes.

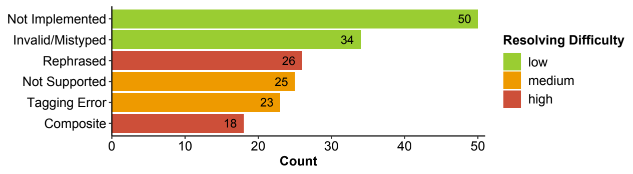


Fig. 7. Number of failed queries categorized by the reasons of the failures. The colors of the bars indicate the relative difficulty of resolving a failure.

6 Discussion and Limitations

6.1 Scalability and Generalizability

Technically there are many ways to create a set of rules that implement the same dataflow function. Following the active development and enhancement of the system functionality, from time to time grammar rules can be combined and rewritten to make the grammar more concise. We keep iterating and refining the FlowSense grammar to expand its functionality. FlowSense currently includes about 200 variables and a rule set of around 500 rules in its grammar. Our grammar development practice employs continuous integration and maintains a test set (of 131 test queries by the time of this writeup) to ensure that all categories of VisFlow functions may execute properly during iterations and extensions of the grammar. Approximately 10 to 20 rules need to be added to support a new dataflow function category.

Though the grammar rules of FlowSense are coupled with the underlying VisFlow functionality, its approach of utilizing special utterance placeholders is generalizable to other dataflow systems that employ similar modular component design. Once the data- and diagram-independent dataflow elements are identified, these elements can be represented by special utterances in the grammar and dataflow implementation can subsequently be extended to process them. For example, we may extend the grammar to support more data processing power obtainable from a computational dataflow system like KNIME [4].

[6]https://visflow.org/extension. The extension is not supported by the NLI.

6.2 User Behavior and Engagement

The effectiveness of a grammar-based semantic parser couples with the grammar design. One negligence in the grammar may result in unexpected rejections of seemingly acceptable queries. Despite careful grammar design, the user is likely to come up with questions that exceed the scope of the grammar. However, we find that users are willing and able to refine rejected queries with a small number of trial-and-error attempts. Besides users may become more proficient with the NLI after reading query examples so as to understand the NLI capability. Yet showing too many examples may limit the user's thoughts and forfeit the benefit of using an NLI. We would like to further study user behavior regarding NLI usage in DFVS in the future to better identify when and where query examples need to be provided.

We also observe that users tend to perform composite queries and ask for batch operations using the NLI. With traditional mouse/keyboard interaction, the results of such queries have to be achieved by a sequence of interactions. FlowSense increases the data exploration efficiency by naturally enabling batch operations. In fact, we notice some users were able to repeat successful short queries that achieved the most batched result. The convenience of using NL to carry out multiple operations may improve the user's engagement [18], provide interaction "shortcuts", and make dataflow features more accessible by simplifying the creation of rather complicated sub-diagrams, *e.g.* "highlighting".

6.3 Technique and Scope

We prefer semantic parsing to deep learning mainly because the latter has a bottleneck of requiring a large volume of training examples. Though there are benchmark datasets for general NLP, there has not yet been a training set catered for visualization-oriented NLI or DFVS. In the future with more users working with the NLI, we may collect more user queries that constitute a rich training set in order to support methods like neural networks for text classification [56].

Currently FlowSense only works with dataflow diagram editing. But it may be desirable for the NLI to answer analytical questions like "*Does the vehicle speed decrease over years in NYC?*" by creating a visualization like Fig. 5(4). To that end we need further research on the dataflow functions and their application to answering analytical questions. One possible direction is to study how DFVS diagrams can be constructed for knowledge-based Q&A [17].

7 Conclusions

In this work we design FlowSense, a novel NLI for visual data exploration within a DFVS. We build FlowSense for the VisFlow framework and show that it improves the DFVS usability and simplifies diagram construction. FlowSense applies semantic parsing to map NL input to VisFlow functions. Its emphasis on special utterances and usage of special utterance placeholders make the semantic parsing independent of datasets and diagrams, but at the same time aware of the dataflow context. The real-time feedback of tagged special utterances, as well as query and token auto-completion features, largely helps the user understand the underlying parsing state. Our case study and user study results demonstrate the effectiveness of the proposed NLI, and help identify future research directions for its improvement.

Acknowledgments

We would like to thank BlindData.com for providing the user study dataset. This work was supported in part by: the Moore-Sloan Data Science Environment at NYU; NASA; NSF awards CNS-1229185, CCF-1533564, CNS-1544753, CNS-1730396, CNS-1828576. B. Yu and C. T. Silva are partially supported by the DARPA MEMEX and D3M programs. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] D3: Data-Driven Documents. `https://d3js.org`

[2] IBM SPSS Modeler. `https://www.ibm.com/products/spss-modeler/`

[3] IBM Watson Analytics. `https://www.ibm.com/watson-analytics`

[4] KNIME data analysis platform. `http://www.knime.org/`

[5] Microsoft Power BI. `https://powerbi.microsoft.com/`

[6] NLTK. `http://www.nltk.org/`

[7] Tableau Software. `http://www.tableausoftware.com/`

[8] Thoughtspot. `http://www.thoughtspot.com/`

[9] TLC trip records. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`

[10] Wolfram Alpha. `http://www.wolframalpha.com/`

[11] M. Allahyari, S. A. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut. A brief survey of text mining: Classification, clustering and extraction techniques. In *Proc. KDD Bigdas*, 2017.

[12] R. Amar, J. Eagan, and J. Stasko. Low-level components of analytic activity in information visualization. In *IEEE Symposium on Information Visualization (InfoVis'05)*, pages 111–117, 2005.

[13] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[14] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Software Engineering*, 12(4):538–546, 1986.

[15] L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. Crossno, C. T. Silva, and J. Freire. VisTrails: Enabling interactive multiple-view visualizations. In *Proc. IEEE Visualization Conference*, pages 135–142, 2005.

[16] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on Freebase from question-answer pairs. In *Proc. Empirical Methods in Natural Language Processing (EMNLP'13)*, pages 1533–1544, 2013.

[17] P. Clark, J. Thompson, and B. Porter. A knowledge-based approach to question-answering. In *Proc. AAAI Fall Symposium on Question-Answering Systems*, pages 43–51, 1999.

[18] P. R. Cohen. The role of natural language in a multimodal interface. In *Proc. 5th Annual ACM Symposium on User Interface Software and Technology (UIST'92)*, pages 143–149, 1992.

[19] K. Cox, R. E. Grinter, S. L. Hibino, Lalita, J. Jagadeesan, and D. Mantilla. A multi-modal natural language interface to an information visualisation environment. *International Journal of Speech Technology*, 4:297–314, 2001.

[20] L. Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Trans. Signal and Information Processing*, 3, 2014.

[21] K. Dhamdhere, K. McCurley, M. Sundararajan, Q. Yan, and R. Nahmias. Analyza: Exploring data with conversation. In *Proc. 22nd International Conference on Intelligent User Interfaces*, pages 493–504, 2017.

[22] E. Fast, B. Chen, J. Mendelsohn, J. Bassen, and M. S. Bernstein. Iris: A conversational agent for complex tasks. In *Proc. CHI Conference on Human Factors in Computing Systems (CHI'18)*, 2018.

[23] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[24] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *Proc. Provenance and Annotation of Data: International Provenance and Annotation Workshop*, pages 10–18, 2006.

[25] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. DataTone: managing ambiguity in natural language interfaces for data visualization. In *Proc. 28th Annual Symposium on User Interface Software and Technology (UIST'15)*, pages 489–500, 2015.

[26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

[27] L. Grammel, M. Tory, and M. A. Storey. How information visualization novices construct visualizations. *IEEE Trans. Visualization and Computer Graphics*, 16(6):943–952, 2010.

[28] P. E. Haeberli. ConMan: A visual programming language for interactive graphics. *ACM SigGraph Computer Graphics*, 22(4):103–111, 1988.

[29] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE Trans. Visualization and Computer Graphics*, 24(1):309–318, 2018.

[30] W. Javed and N. Elmqvist. ExPlates: Spatializing interactive analysis to scaffold visual exploration. *Computer Graphics Forum*, 32(2):441–450, 2013.

[31] A. Kumar, J. Aurisano, B. D. Eugenio, A. Johnson, A. Gonzalez, and J. Leigh. Towards a dialogue system that supports rich visualizations of data. In *Proc. 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2016.

[32] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

[33] Y. Li, H. Yang, and H. V. Jagadish. NaLIX: A generic natural language search environment for XML data. *ACM Trans. Database Systems*, 32(4), 2007.

[34] P. Liang and C. Potts. Bringing machine learning and compositional semantics together. *Annual Review of Linguistics*, 1:355–376, 2014.

[35] J. Mackinlay, P. Hanrahan, and C. Stolte. Show Me: Automatic presentation for visual analysis. *IEEE Trans. Visualization and Computer Graphics*, 13(6):1137–1144, 2007.

[36] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, P. Inc, S. J. Bethard, and D. Mcclosky. The Stanford CoreNLP natural language processing toolkit. In *Proc. 52nd Annual Meeting of the Association for Computational Linguistics (ACL'14): System Demonstrations*, pages 55–60, 2014.

[37] J. Meyer-Spradow, T. Ropinski, J. Mensmann, and K. Hinrichs. Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications*, 29(6):6–13, 2009.

[38] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[39] S. G. Parker and C. R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Proc. ACM/IEEE Conference on Supercomputing*, 1995.

[40] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL'15)*, 2015.

[41] J. Poco, H. Doraiswamy, H. T. Vo, J. L. D. Comba, J. Freire, and C. T. Silva. Exploring traffic dynamics in urban environments using vector-valued functions. *Computer Graphics Forum*, 34(3):161–170, 2015.

[42] J. C. Roberts. Waltz - an exploratory visualization tool for volume data, using multiform abstract displays. In *Proc. SPIE Visual Data Exploration and Analysis V*, volume 3298, pages 112–122, 1998.

[43] J. C. Roberts. On encouraging coupled views for visualization exploration. In *Proc. SPIE Visual Data Exploration and Analysis VI*, volume 3643, pages 14–24, 1999.

[44] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *Proc. 29th Annual Symposium on User Interface Software and Technology (UIST'16)*, pages 365–377, 2016.

[45] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[46] M. Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012.

[47] A. Srinivasan and J. Stasko. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE Trans. Visualization and Computer Graphics*, 24(1):511–521, 2018.

[48] A. Srinivasan and J. T. Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *Eurographics Conference on Visualization (EuroVis'17 short paper)*, 2017.

[49] Y. Sun, J. Leigh, A. Johnson, and S. Lee. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In *Proc. 10th International Conference on Smart Graphics*, pages 184–195, 2010.

[50] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. MIT Press, 2003.

[51] C. Upson, J. Faulhaber, T.A., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The application visualization system: a computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4):30–42, 1989.

[52] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *Proc. Annual Meeting of the Association for Computational Linguistics (ACL'15)*, 2015.

[53] K. Wolstencroft, R. Haines, D. Fellows, A. R. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, and C. A. Goble. The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):557–561, 2013.

[54] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proc. 32nd Annual Meeting on Association for Computational Linguistics (ACL'94)*, pages 133–138, 1994.

[55] P. Yin, Z. Lu, H. Li, and B. Kao. Neural Enquirer: Learning to query tables with natural language. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI'16)*, 2016.

[56] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[57] B. Yu and C. T. Silva. VisFlow – Web-based visualization framework for tabular data with a subset flow model. *IEEE Trans. Visualization and Computer Graphics*, 23(1):251–260, 2017.

[58] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

A FlowSense Grammar Design

We provide an open source repository that contains the details of the FlowSense implementation: `https://github.com/yubowenok/flowsense`. This repository contains the grammar rules, backend API (implemented in TypeScript and Python), and integration tests. The structure of this repository and its installation and setup guide can be found within its README file.

In particular, the grammar rules are located in the `*.grammar` files. The entry point is `main.grammar`. The grammar rules are written in the SEMPRE grammar format. More details can be found at the SEMPRE GitHub repository.

B Characteristics of the Diagram Editing Focus Heuristics

Intuitively, the focus score keeps track of the diagram element that is last interacted with. It has two components: the activeness resulted from mouse clicks, and the distance-to-mouse bonus. The activeness score exponentially decreases when there is no interaction on the node, while the distance-to-mouse bonus prioritizes the elements around the last interaction.

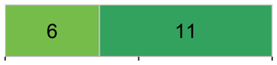
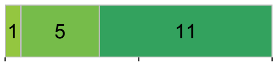
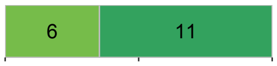
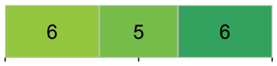
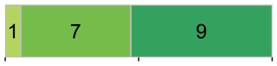
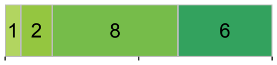
When the mouse hits a node x, node x receives a high activeness score of one from the *Click(X, t)* part, which almost certainly ensures that the focus score of x is higher than any other node y that is not interacted with. Though y (when it is in the proximity of x) may receive a distance-to-mouse bonus that remedies for its exponential loss on the activeness score, note that x receives a distance-to-mouse bonus too, and the bonus can only be higher than the bonus received by y because x is clicked on. Therefore, the outcome is that x becomes the first prioritized node, and y becomes the second prioritized. In other words, if a VisFlow function requires two operand nodes, then x is chosen first, and then y is chosen.

If the user clicks on the background, all nodes have exponentially decreasing activeness score, and their distance-to-mouse bonus will likely dominate the focus score. Consequently, the nodes that are closer to the last click become the chosen query targets. As there can be multiple nodes around the background click, occasionally a node not actually focused by the user may happen to be close to an unintentional background click (*e.g.* accidentally performed during canvas panning). The next NL query may then be incorrectly performed on this node. This error can be fixed by clicking on a specific node to focus on it and redoing the NL query.

C Additional User Study Remarks

* In the SDE test, answering a question wrong results in negative score penalty. Therefore skipping a question can be worthy. Skipping requires an explicit button click. The "unanswered" result is given when the user has no action within the allocated time limit of a question.

* In Fig. 6(b), four outliers due to interruptions on the participant's end are not shown: 2550 minutes on Task1, and 109, 119, 212 minutes on Task3 were measured as the task completion time, which include the interruptions.

Aspect	Feedback
I understand the majority of VisFlow features.	
I understand the subset flow in VisFlow.	
I can follow the VisFlow dataflow diagram and understand their functionality.	
VisFlow is relatively simple to learn and use.	
VisFlow is an effective system for visual data exploration.	
I would like to use VisFlow for my future data exploration tasks.	

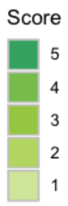


Table 1: VisFlow Survey Result. The feedback column shows the score distribution for each assessed aspect of VisFlow. The numbers on the colored bars show the counts of the scores received. Darker green represents higher score. Overall the users were able to understand well the DFVS functionality and use it effectively for visual data exploration.

D VisFlow Survey Results

The VisFlow user study is part of the FlowSense user study. Before FlowSense was introduced to the users, a tutorial on the details of the VisFlow dataflow framework was given. The participant group of the VisFlow user study is thus exactly same as the group described in the paper: In total, 17 users (11 male, 6 female, all with an age between 20 and 30) who work or study in the field of computer science participated in this study. 12 participants have a data visualization background. 9 are graduate students, and the other 8 are professionals (software engineer, researcher, faculty). 3 participants have prior experience with VisFlow. The participant group is chosen to represent potential DFVS users who share data analysis interest but have The participants are chosen to have a variety of specialities so as to represent potential DFVS users. The participant group includes visualization designers, data scientists, and software engineers who share data analysis interest but have different skill sets.

The users were given a form to assess the effectiveness of VisFlow qualitatively using a Likert scale of 1 to 5 (5 is “strongly agree” and 1 is “strongly disagree”). Table 1 shows the quantitative survey feedback for the VisFlow DFVS. It can be observed that the users were able to understand the subset flow model of VisFlow. The majority of the users agree (with a score of at least 4) that VisFlow presents an effective approach to visual data exploration, and can successfully utilize VisFlow features for their data exploration tasks.

E Query Analysis – Failure Category Description

This following list provides the descriptions for each query failure category we identified in the user study results:

- **Not Implemented**. FlowSense grammar may technically support parsing this query. Yet we have not implemented the corresponding grammar and its web client handler (query execution code for diagram update). Example queries include "*change x column to mpg*". The current system implementation does not support node option changes triggered by the NLI (except for visual editors). Queries of this category can be accepted by extending the grammar and adding more rules.

- **Invalid/Mistyped**. The query is an invalid sentence and cannot be understood by a human; Or the query has mistyped words and fails to describe the intended data entity or dataflow element.

- **Rephrased**. The user rephrases the query using grammatical structures not expected by the grammar, or the user uses words that do not appear in the dataset table to describe a table entity or value. For example, in Task 3 if the user mentions "degree", FlowSense does not know that "degree" is equivalent to the "HighestLevelOfEducation" column in the data. Though one can easily inform a system of such equivalence case-by-case or find synonyms from WordNet, it is non-trivial to generally detect such equivalence. Consider the equivalence between "*Show only segments with signs*" and "*Show only segments with a sign of yes*": "yes" is not an immediate synonym of "with" and their common implication of "existence" is subtle. There needs to be additional knowledge base added to the system to support the detection of concept equivalence.

- **Not Supported**. The functionality indicated by the query is not supported by the VisFlow dataflow framework. This is not an issue of the NLI but a limitation of the underlying DFVS. A query like "*How many questions were skipped*" asks directly an analytical question about the dataset and exceeds the scope of VisFlow. It cannot be accepted by simply extending the grammar because there needs to be a reasonable way to construct dataflow sub-diagrams to answer the analytical questions, which can be complex and challenging to identify.

- **Tagging Error**. A special utterance should have (have not) been tagged, but it was not (was) tagged. For example, the query "*Select iris with id between 3 and 5*" has the word "iris" that is both a word to describe the data entity and a dataset name. When FlowSense automatically tags "iris" as a dataset name special utterance, the parser may fail to accept the query. In this case the user may manually override the tagging to avoid the error resulted from parsing ambiguity. In future work we may also explore techniques that can be integrated into the parser to structurally reduce such errors.

- **Composite**. The user inputs a query that attempts to execute multiple VisFlow functions that exceed the limit expected by the grammar or the web client handler. An example is

"*Highlight bachelors in red and masters in green in node-15*". This is achievable using attribute filters to find candidates with Bachelors and Masters degrees, followed by visual editors to give them colors, and finally a set operator to merge the two groups. In this case multiple VisFlow functions have to be performed altogether. The parser and execution handler did not expect queries of this composite level. The grammatical structure between these multiple functions poses parsing difficulty. It is recommended that composite queries are refactored into multiple smaller steps so as not to overload the NLI with complicated grammatical structure that exceeds its parsing capability.

- **Bug**. The system should have the capability of handling that query. But due to an implementation bug that was unidentified at the time of the user study, the query parsing or execution went wrong and did not arrive at the expected outcome.