Exploring Natural Language to Visualization translation through conversational interaction

Rishab Mitra

A thesis presented for the degree of

Bachelor of Science

College of Computing

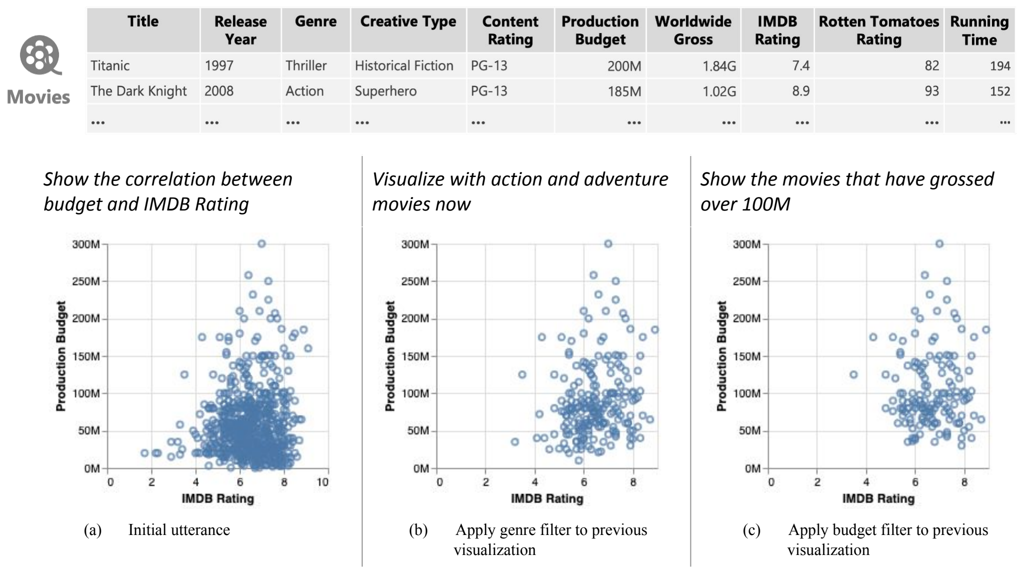
Georgia Institute of Technology

United States of America

December 17th, 2021

Abstract

Natural language interfaces (NLIs) have allowed users to explore and visualize data without much overhead in learning how to operate the interface for visual analysis. However, the state-of-the-art NLIs for visualization only support one-shot queries with minimal capability for follow-up queries, so users are unable to build upon previous visualizations. We introduce the Conversational Interaction for Data Visualization (CI4DV) toolkit for developers, a toolkit for natural language to data visualization that takes in a natural language query and dataset as input and outputs a visualization specification. CI4DV offers the additional capability to follow-up on previous visualizations, allowing developers to build NLIs on top of CI4DV that allow users the opportunity for conversational interaction with the NLI. We also propose two applications that are built upon CI4DV to demonstrate the toolkit's capabilities and usage - an application that features an editable mind map and a Jupyter notebook with nested cells.



Title	Release Year	Genre	Creative Type	Content Rating	Production Budget	Worldwide Gross	IMDB Rating	Rotten Tomatoes Rating	Running Time
Titanic	1997	Thriller	Historical Fiction	PG-13	200M	1.84G	7.4	82	194
The Dark Knight	2008	Action	Superhero	PG-13	185M	1.02G	8.9	93	152
...	...	...	...	...	...	...	...	...	...

Show the correlation between budget and IMDB Rating

Visualize with action and adventure movies now

Show the movies that have grossed over 100M

(a) Initial utterance

(b) Apply genre filter to previous visualization

(c) Apply budget filter to previous visualization

Figure 1: An example of a conversation that a user can have with an interface that translates a natural language query to a visualization. These queries are sequential, so each visualization builds upon its predecessor. Starting with an initial utterance (a), the user then asks to show only movies that have the genre of ”Action and Adventure” (b). Then the user applies another filter by asking to show movies that have a worldwide gross over 100 million dollars (c). These changes are reflected in the visualizations, as the scatter plots become less populated with each successive query.

1 Introduction

Currently, research in natural language interfaces (NLIs) for visualization has garnered increasing attention due to the inherent accessibility associated with natural language (NL). Users are not required to learn a new programming language, but rather, have a basic understanding of English. An NLI for visualization receives an NL statement (e.g., “correlate goals and salary”) as input, parses this statement to extract necessary information (e.g., “goals” and “salary” as attributes with ”correlate” as the analytic action), and recommends a visualization (e.g., a scatter plot with goals on the X axis and salary on the Y axis).

There have been many attempts to create robust NLIs for visualization [11, 16, 13, 4, 3]. While many of these NLIs have custom architectures and taxonomies to parse natural language queries and display visualizations,the Natural Language for Data Visualization toolkit NL4DV [11] offers a general query processing engine that parses natural language queries to produce analytic specification. This allows developers to create NLIs for visualization without having to be familiar with standard natural language processing techniques. The toolkit also supports Vega-Lite [12], the visualization grammar that is used in many industry applications such as Voyager and Polestar. However, there is an issue in that current NLIs for visualization are narrow in scope due to how these systems currently only support one-off queries. Users are unable to build upon their previous queries. Thus, a toolkit such as NL4DV [11] only benefits those who already have an idea of what their dataset looks like. It does not help those who want to do initial explorations of a dataset and follow up on potential leads that they have found through their explorations. Natural language input has the potential to support richer visualizations by allowing for users to build upon visualizations through conversational interaction. Also, for most dialogue systems it is imperative that most of them are able to hold the context of multiple queries at once, as initial explorations of the dataset can have multiple one-off queries that are unrelated to each other. A user may want to follow up on one of these queries in the future, so the system must retain the contexts for these queries and generate visualizations based on the contexts.

Because of these limitations of rule and grammar-based systems, the purpose of this thesis is to implement a toolkit that supports conversational interaction for users. We build upon NL4DV's architecture and present the Conversation Interaction for Data Visualization toolkit (CI4DV), a toolkit that allows users to hold multiple conversations and build visualizations through sequential queries. Furthermore, we propose a set of applications built upon CI4DV that exhibit the follow-up system, namely an editable mind map and nested Jupyter notebook. This, in turn, would show that the toolkit can be used to build conversational NLIs from scratch or augment existing NLIs.

2 Literature Review

We explore existing literature in the visualization community to understand context-based approaches to translate natural language to visualization.

Current approaches detect a NL query's attributes and tasks and use these components to craft a visualization [11, 2]. Attributes are dataset-specific while analytic tasks are dataset-agnostic. Amar et al. [1] have defined a taxonomy of ten analytic tasks encountered by users while working with a dataset. While earlier visualization systems used proprietary tasks that made development somewhat confusing and unportable, more recent work (e.g., NL4DV[11] and QUDA[2]) are based on these low-level analytic tasks. FlowSense [16] uses a rule and grammar-based architecture similar to NL4DV [11] to translate an NL query to a visualization. One difference between the two systems is that Flowsense allows for the manipulation and linkage of multiple visualizations but it does not work with one visualization at a time.

Other examples of NLIs for visualization also serve as influences for CI4DV. DataTone is another example of an NLI for visualization that parses NL queries and creates visualizations

[3]. One issue with user-generated NL queries is that some input queries may have potential ambiguities (e.g. the word *rating* can refer to either *content rating* or *critics' rating* in a movie dataset). DataTone proposes a solution to this problem by introducing GUI assets called *ambiguity widgets* that seek to disambiguate certain words or phrases in an input query. Evizeon is another NLI for visualization that introduces a taxonomy to support conversational interaction with users [4]. Evizeon's methodology for creating follow-up visualizations modifies previous context objects based on its probabilistic grammar-based approach. However, there are some limitations to Evizeon as it is tailored towards map-based visualizations. Snowy is another interface that is aimed at supporting conversational interaction [14]. This system generates and recommends natural language utterances based on language pragmatics and data uniqueness. Thus, the system supports the generation of NL queries rather than parsing user-supplied queries like NL4DV and CI4DV.

All of these NLIs mentioned use rule and grammar-based systems to convert NL queries into visualizations. While these rule and grammar-based systems are effective for certain grammars, an issue with such systems is that they are restricted by the count and coverage of grammar rules that are in place. Furthermore, attributes of a dataset can have aliases (e.g., cost and price) and abbreviations (e.g., GDP stands for "Gross Domestic Product"). NL4DV supports configuring these aliases but there will always be more undiscovered aliases. This problem can potentially be solved with machine learning models given enough data, and there is some research done on converting natural language into visualization using deep learning or machine learning [2, 7]. These models currently do not support follow-up queries, as the datasets for natural language to visualization systems [2, 8, 15] do not include enough follow-up queries for deep learning training.

Natural language toolkits like NLTK [6] and Stanford CoreNLP [9] are general purpose toolkits that can perform dependency parsing, entity recognition, and part of speech tagging. Much like its predecessor NL4DV, CI4DV uses these toolkits to parse users' NL queries to detect attributes and tasks. As CI4DV handles all of the natural language processing in the backend, developers who leverage CI4DV to create NLIs do not have to understand the architecture of these natural language toolkits. Developers instead can focus on UI design, and make high-level calls to the CI4DV functions to handle all natural language to visualization tasks.

3 Methodology

In this section, we detail CI4DV's design and implementation. We will use and augment NL4DV [11] as our main base for implementation.

Given a dataset (as a CSV, JSON, or TSV), and a single-line query, NL4DV infers attributes, tasks and visualizations with a single function call to `analyze_query(query, dialog)`. The parameter `query` represents the user-supplied NL utterance. The optional parameter `dialog` is a boolean value that determines the type of query. If true, then the query is considered as a follow-up. Otherwise the query is treated as a standalone query distinct from any other queries that the user may have inputted previously.

All of the user's previous contexts are stored in a unique data structure very similar to a tree. The context tree also allows for branching conversations, where users can provide multiple unrelated follow-ups to the same query. It is implemented in CI4DV as a dictionary of lists where `conversation_id` is the key for the dictionary and `context_id` is the position of the context object in the list. This makes accessing context objects in a dictionary and appending elements to the end of a list efficient, as both operations can be done in O(1) time. The capability of branching conversations would be simple if the context tree was implemented with nodes and pointers. However, a dictionary that cannot represent branching conversations as well as trees backed by nodes and pointers. Thus, we created an approach to maintain the branching capability. For users that want to follow up on a context object that was in the middle of a conversation, they must supply a follow-up query, `conversation_id`, and `context_id` to `analyze_query`. To represent this branch, CI4DV would, in turn, create a new conversation for the branch with a conversation id of the string `'{conversation_id}.{context_id}.{branch_number}'`. This shows that the conversation is a child of a previous context object. Moreover, it allows for multiple branches to stem off of each context object, as each branch will have its own unique `branch_number`. Figure 2 presents a visual representation of how conversations and follow-up queries are stored in CI4DV.

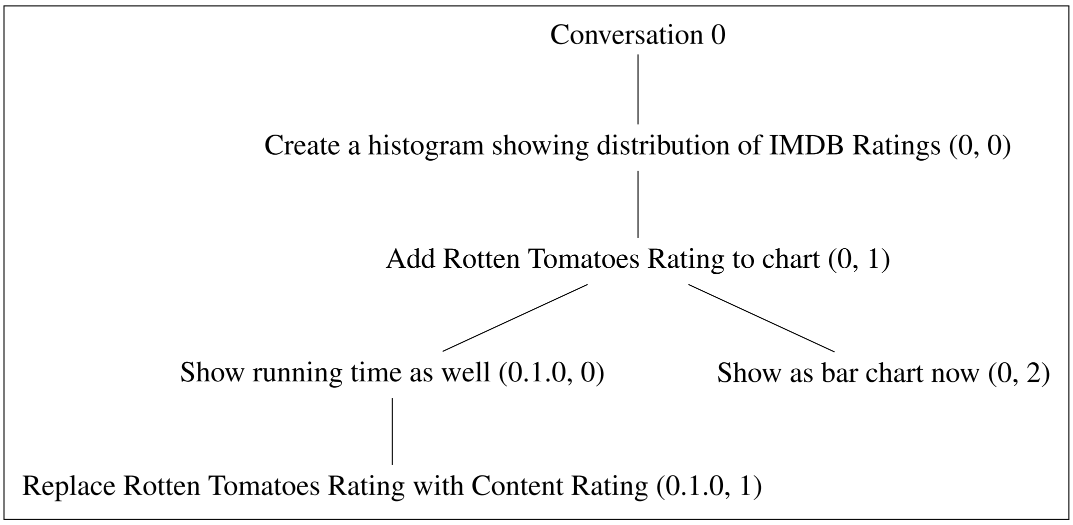

Conversation 0
Create a histogram showing distribution of IMDB Ratings (0, 0)
Add Rotten Tomatoes Rating to chart (0, 1)
Show running time as well (0.1.0, 0)
Show as bar chart now (0, 2)
Replace Rotten Tomatoes Rating with Content Rating (0.1.0, 1)

Figure 2: An example of how NL queries are stored in the context tree. The tuple in the tree is in the format of (`conversation_id`, `context_id`). The conversation IDs are determined by which follow-up query was uttered first.

The most immediate solution to implementing the context tree in CI4DV was to use nodes and pointers. However, an issue with the usage of pointers is that accessing, adding, and modifying the tree can take

Each query is assigned a `conversation_id` and `context_id`. `Conversation_id` is the id given to a user when the user asks a standalone query instead of a follow-up. If a user provides a standalone query, then the system returns a new conversation id with `conversation_id = 0`. In order to receive a correct follow-up, a user must input a `conversation_id`, and optionally a `context_id`. If no `context_id` is given, then the system takes the last query inputted in a given `conversation_id`. Thus, the system has the capability of managing multiple contexts at the same time, so the user can explore datasets and their questions even further.

When parsing a query to generate a visualization, CI4DV tries to detect for tasks and attributes. Attributes and tasks can be referenced *explicitly* (e.g. through direct reference of attribute names) or *implicitly* (e.g. through references to an attributes' values or through synonyms of attribute names). As NL4DV is already effective at detecting tasks and attributes, the pipeline for this detection remains largely intact in our development for CI4DV. In fact, the methodology for task and attribute detection acts as our foundation for follow-up detection. Most follow-up queries that a user will ask using this system can be broken down into the following 3 operations - **add, remove,** and **replace**. Table 1 details how each follow-up operation can affect the succeeding visualization. As Table 1 details, simply adding, removing, or replacing an attribute can change the tasks and visualization types for an analytic specification, entirely revamping the visualization.

CI4DV uses rules to determine what attributes, tasks, or visualizations appear in the follow-up query and which follow-up operation (add, remove, or replace) that the query contains. We use ngram-distance to determine which attributes, tasks, or visualizations are applied to the follow-up operation. Figure 3 shows the overall workflow for how the system works to process follow-up queries. At a high level, given a NL follow-up query, CI4DV treats this query as if it is a standalone query, and uses the NL4DV engine to detect new attributes, tasks, and visualization types. CI4DV also has its own contextual query processor that determines the follow-up operation type (e.g. add, remove, and replace) for each new attribute, task, or visualization type detected. Then CI4DV extracts the context object of the query that the user is following up on and updates this context object given the follow-up operation for each new task, attribute, or visualization type detected, creating new lists for each of them. Then using these lists, we reuse the NL4DV pipeline to generate visualization recommendations and return them to the user.

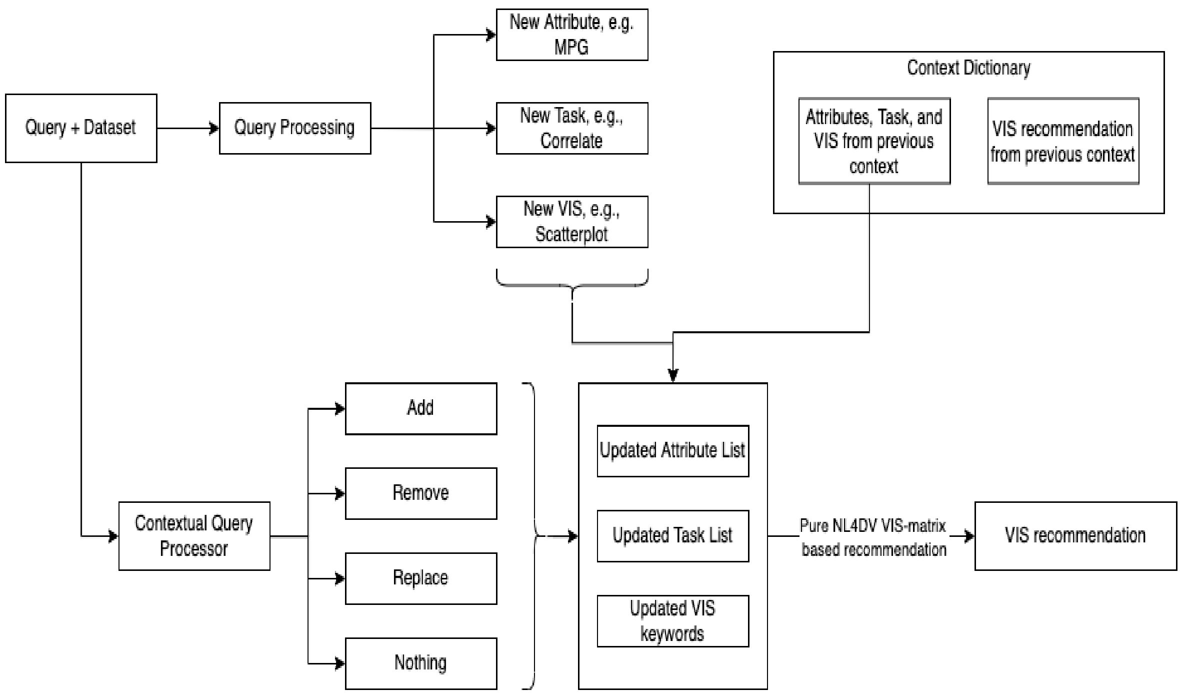
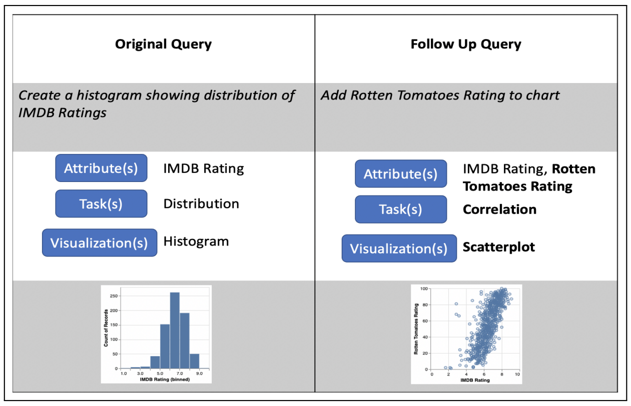
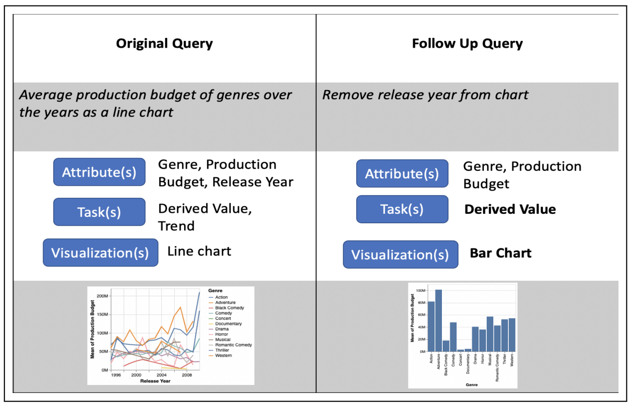

New Attribute, e.g. MPG
Context Dictionary
Query + Dataset
Query Processing
New Task, e.g., Correlate
Attributes, Task, and VIS from previous context
VIS recommendation from previous context
New VIS, e.g., Scatterplot
Add
Updated Attribute List
Remove
Contextual Query Processor
Updated Task List
Pure NL4DV VIS-matrix based recommendation
VIS recommendation
Replace
Updated VIS keywords
Nothing

Figure 3: An overview of CI4DV's architecture. The arrows represent the flow of information between the modules. Ultimately all of the information from the query and dataset is transformed into a visualization recommendation.

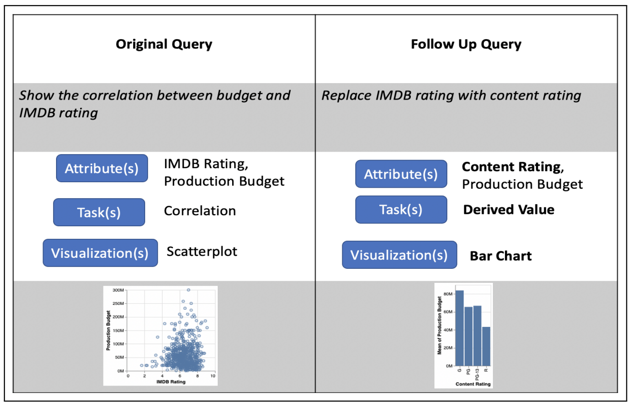
When considering the three follow-up operations, users may be interested in following up on a previous query's tasks, attributes, and visualization types. Thus, we came up with

Add	

Original Query	Follow Up Query
Create a histogram showing distribution of IMDB Ratings	*Add Rotten Tomatoes Rating to chart*
Attribute(s): IMDB Rating	Attribute(s): IMDB Rating, **Rotten Tomatoes Rating**
Task(s): Distribution	Task(s): **Correlation**
Visualization(s): Histogram	Visualization(s): **Scatterplot**

Remove	

Original Query	Follow Up Query
Average production budget of genres over the years as a line chart	*Remove release year from chart*
Attribute(s): Genre, Production Budget, Release Year	Attribute(s): Genre, Production Budget
Task(s): Derived Value, Trend	Task(s): **Derived Value**
Visualization(s): Line chart	Visualization(s): **Bar Chart**

Replace	

Original Query	Follow Up Query
Show the correlation between budget and IMDB rating	*Replace IMDB rating with content rating*
Attribute(s): IMDB Rating, Production Budget	Attribute(s): **Content Rating**, Production Budget
Task(s): Correlation	Task(s): **Derived Value**
Visualization(s): Scatterplot	Visualization(s): **Bar Chart**

Table 1: Instances of how each follow-up operation can affect the succeeding visualizations of the follow-up queries.

potential follow-up queries that could generate new visualizations. After compiling and creating multiple follow-up queries that differed in syntax and semantics, we were able to generate a matrix that essentially assigned each query to a specific cell in the matrix. Table 2 presents a snippet of the matrix below for `movies-w-year.csv`, a dataset that gives information about movies.

	Attributes	**Tasks**	**Vis**
Add	Show **production budget** as well	Show me the **max** gross	Show as a **barchart**
Replace	Replace **budget** with **gross**	Show **max** gross instead of min	Show a **barchart** instead of a **scatter plot**
Remove	Remove **genre** from chart	Remove all **filters**	Show **original** chart

Table 2: The follow-up matrix that classifies each potential follow-up query into one cell of the matrix. One example for each type of follow-up query is provided in each cell.

CI4DV is able to fully support all types of the follow-up queries that are present in the follow-up matrix. Furthermore, queries that follow up on attributes can be separated into two forms - *explicit* and *implicit*. Explicit follow-up queries are queries that make direct reference to one of the follow-up operations - add, replace, and remove (e.g. replace production budget with gross). Implicit follow-up queries do not make direct reference to one of the follow-up operations, but contain phrases or words that denote the follow-up operation. An example of such a query would be *Show only production budget*. In this example, the user's query suggests to remove all other attributes from the previous query except for production budget, and to visualize the distribution of production budget. The keyword in this query is *only*, which tells the system that the follow-up operation should be *remove*.

When implementing the follow-up matrix, while the *explicit* and *implicit* types of queries align very well when a user wants to modify attributes, this methodology does not work as well with tasks. This is because a user will be unaware of the low-level analytic tasks that NL4DV uses for its design [1] (e.g. a user would not say *add find extrema task to query*). Naturally, a user would rather say *Show me the max value*, which would add the *find extremum* task to the original query. Thus, almost all of the queries that follow-up on tasks would be considered as implicit follow-up detection.

Furthermore, this CI4DV only follows up on a select few of the follow-up tasks implemented in NL4DV, namely *sort, find extremum, filter, and derived value*. The other analytic tasks do not lend well to a follow-up task, so they are not supported in this system. For example, a user should not be able to add a correlation task to a follow-up query that does not have two quantitative attributes (e.g. Production Budget and Running Time). Likewise, a user cannot remove a distribution task if the original query only had one attribute. Thus, the system only follows up on the tasks that users can provide natural utterances for all three follow-up operations (add, replace, and remove).

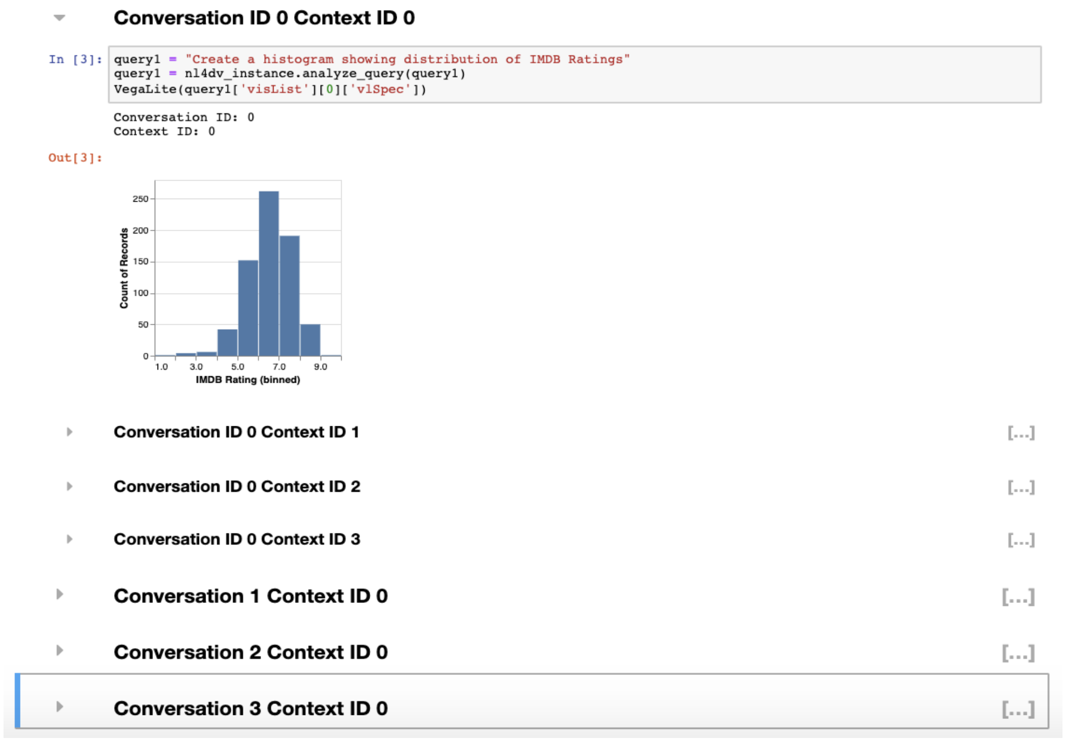
In terms of following up on a visualization type with a natural language query, it is not possible to add a visualization to a previous query in the same manner as adding an attribute to a previous query. The only follow-up operation that can be used for following up on the visualization type is to *replace* the previous visualization. However, we must also ensure that the attributes and tasks of the previous query can be represented by the replacement visualization. If not, CI4DV defaults to the recommendation provided by NL4DV.

4 Example Applications

Using CI4DV, developers can create NLIs that exhibit the conversational nature of the toolkit, allowing for users to build visualizations and explore their datasets. We propose applications that can represent the full capabilities of CI4DV.

4.1 Nested Jupyter Notebook

As CI4DV uses Vega-Lite specifications to render visualizations in Python environments like Jupyter Notebook, this would enable Python developers to create visualizations without learning other visualization libraries such as `seaborn` and `matplotlib`. Typically, Jupyter Notebooks have cells that are executed linearly. Python programmers can call the `render_vis(query, dialog)` function in CI4DV to output a visualization in the Jupyter notebook. However, for a Python programmer working with CI4DV and executing multiple follow-ups in multiple conversations, it can be hard to navigate which conversations are mapped to certain visualizations. Furthermore, cells can be executed nonlinearly, which can lead to CI4DV recommending incorrect visualizations. Thus, the Jupyter notebook must be organized to provide clarity in the structure of the conversations and their follow-ups. Conversations and all of their respective follow-ups must be grouped together. Figure 4 shows an instance of a Jupyter notebook running CI4DV. The notebook is organized by conversation, and a conversation's follow-ups are indented. If users want to follow up on the last query in a conversation (e.g. Conversation 3), they can create another indented cell under the Conversation 3 header and the application will use the last context object of Conversation 3.


Conversation ID 0 Context ID 0

```
In [3]: query1 = "Create a histogram showing distribution of IMDB Ratings"
        query1 = nl4dv_instance.analyze_query(query1)
        VegaLite(query1['visList'][0]['vlSpec'])

        Conversation ID: 0
        Context ID: 0
Out[3]:
```

Conversation ID 0 Context ID 1 [...]

Conversation ID 0 Context ID 2 [...]

Conversation ID 0 Context ID 3 [...]

Conversation 1 Context ID 0 [...]

Conversation 2 Context ID 0 [...]

Conversation 3 Context ID 0 [...]

Figure 4: An instance of the nested Jupyter Notebook. Each respective cell has its auto-generated nested header that displays the conversation id and context id. The headers are also collapsible, giving users the option to focus on one conversation at a time.

4.2 Editable Mind Map

A mind map allows users to organize information and present a hierarchical relationship between nodes of the mind map. Because of its hierarchical nature and capability of displaying different branches of conversations, the mind map can be used to represent the context tree. Thus, we present an editable mind map that provides users a graphical representation of the context tree. Users' input queries and their respective visualizations are displayed in the application. Each conversation has its own mind map in this application. Furthermore, users can extend the mind map further by adding an input query as a node to the end of a path or create a new branch at any point of the conversation. Figure 5 shows an instance of the mind map, displaying a conversation's branches, follow-up queries, and the queries' respective visualizations.

The graphical user interface behaves similar to other mind map creation tools like *coggle.it*. The server-side code involves activating an instance of CI4DV and making a call to `analyze_query(query, dialog)`. If the user adds a new query to the mind map and is part of an existing conversation, then `dialog=True` The application also takes the `context_id` and `conversation_id` from the previous node and extracts the context object from the context tree using these values to create the new visualization. The visualization is rendered and displayed in the mind map.

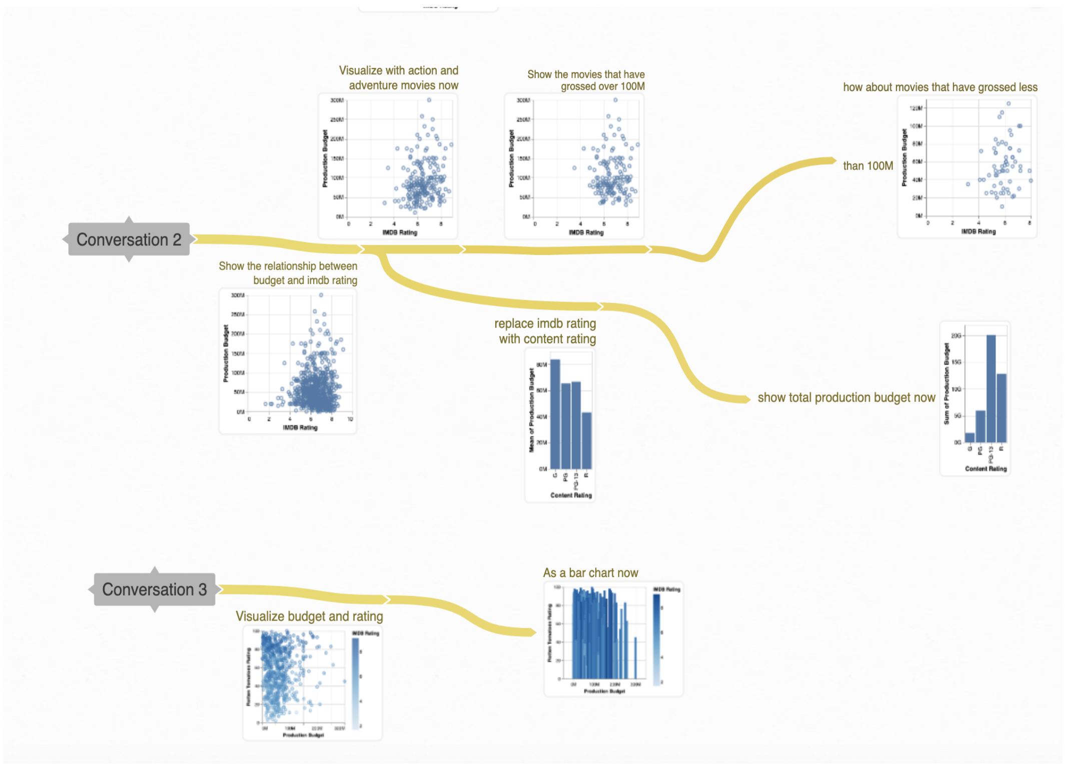

Conversation 2
Visualize with action and adventure movies now
Show the movies that have grossed over 100M
how about movies that have grossed less than 100M
Show the relationship between budget and imdb rating
replace imdb rating with content rating
show total production budget now
Conversation 3
Visualize budget and rating
As a bar chart now

Figure 5: An editable mind map that allows users to append queries to the end of conversations' branches or create new branches in a conversation. New conversations can be added by creating a conversation node.

5 Limitations and Future Work

While CI4DV is an initial effort to introduce dialogue to natural language interface for visualization, there is still future work to be considered for this problem. The immediate next step is to remove the need for the `dialog` parameter. This parameter tells the user if the query is a follow-up or not, but for most applications that a developer creates using CI4DV, they must include a text box for the query as well as a checkbox for the `dialog` parameter. This can end up being incredibly cumbersome for the user of such applications, because they must remember to flag the checkbox. If they do not, then this can lead to incorrect visualizations. We want to reduce overhead for the user as much as possible. Thus, a solution to removing the `dialog` parameter is to implement additional query processing to determine if a query is a follow-up or not. Of course, this processing needs to be very accurate, or else the recommendations will be untrustworthy. However, if the follow-up processing is not as robust, CI4DV can provide two recommendations - one recommendation assuming that the query is a follow-up, and one recommendation assuming that the query is new. Both recommendations can be assigned a score that reflects the confidence level of the visualization recommendation.

Also, there is no existing corpus that contains a large amount of natural language queries and follow-ups with visualizations. A user study can be conducted to generate this corpus, so that there is a benchmark for natural language interfaces for visualization. Similarly, we can also conduct a user study to assess the utility and usability of CI4DV for users.

Furthermore, while we support context and conversation management through the id numbers (e.g. `conversation_id` and `context_id`), there should also be a way for users to switch conversations through natural language. Further research must be done about the ways humans switch conversations naturally so that we can analyze the syntax of these utterances. This would make CI4DV even more accessible to users, as users may be unfamiliar with how to work with conversation ids and context ids in code.

In terms of the visualization aspects of the follow-up system, the follow-up system does not take the original query's visualization in as context. Therefore, the original query's visualization has no bearing on the follow-up query's visualization, so if the original visualization was a bar chart, the follow-up query's visualization can be another type of visualization, such as a scatterplot. We may want to have the original visualization factored into the follow-up's visualization and use it as an anchor point. There is already existing literature and systems in place for anchoring visualizations such as Dziban [5] and Draco [10]. As these systems are open-source, the main challenge would be to integrate these systems within CI4DV and the follow-up system at large to create more sensible visualizations for the user.

6 Conclusion

Through CI4DV, we present a natural language toolkit where users can hold multiple conversations with the interface to generate visualizations that they want to explore. CI4DV has two essential components - the taxonomy for follow-up operations (add, remove, and replace) as well as the capability for a user to hold multiple conversations with the system. With the two applications that exhibit CI4DV's features, we hope to show developers can create other interfaces that allow for conversational interaction, thus increasing the accessibility for users to build visualizations.

References

[1] Robert Amar, James Eagan, and John Stasko. "Low-Level Components of Analytic Activity in Information Visualization". In: INFOVIS '05 (2005), p. 15. URL: https://doi.org/10.1109/INFOVIS.2005.24.

[2] Siwei Fu et al. "Quda: Natural Language Queries for Visual Data Analytics". In: (May 2020).

[3] Tong Gao et al. "DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization". In: UIST '15 (2015), pp. 489–500. DOI: 10.1145/2807442.2807478. URL: https://doi.org/10.1145/2807442.2807478.

[4] Enamul Hoque et al. "Applying Pragmatics Principles for Interaction with Visual Analytics". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 309–318. DOI: 10.1109/TVCG.2017.2744684.

[5] Halden Lin, Dominik Moritz, and Jeffrey Heer. "Dziban: Balancing Agency Automation in Visualization Design via Anchored Recommendations". In: CHI '20. Honolulu, HI, USA: Association for Computing Machinery, 2020, pp. 1–12. ISBN: 9781450367080. DOI: 10.1145/3313831.3376880. URL: https://doi.org/10.1145/3313831.3376880.

[6] Edward Loper and Steven Bird. "NLTK: The Natural Language Toolkit". In: *CoRR* cs.CL/0205028 (2002). URL: http://dblp.uni-trier.de/db/journals/corr/corr0205.html#cs-CL-0205028.

[7] Yuyu Luo et al. "Natural Language to Visualization by Neural Machine Translation". In: *IEEE Transactions on Visualization and Computer Graphics* (2021), pp. 1–1. DOI: 10.1109/TVCG.2021.3114848.

[8] Yuyu Luo et al. "Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks". In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD/PODS '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 1235–1247. ISBN: 9781450383431. DOI: 10.1145/3448016.3457261. URL: https://doi.org/10.1145/3448016.3457261.

[9] Christopher D. Manning et al. "The Stanford CoreNLP Natural Language Processing Toolkit." In: *ACL (System Demonstrations)*. The Association for Computer Linguistics, 2014, pp. 55–60. ISBN: 978-1-941643-00-6. URL: http://dblp.uni-trier.de/db/conf/acl/acl2014-d.html#ManningSBFBM14.

[10] Dominik Moritz et al. “Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 438–448. ISSN: 1077-2626. DOI: 10.1109/TVCG.2018.2865240. URL: https://doi.org/10.1109/TVCG.2018.2865240.

[11] Arpit Narechania, Arjun Srinivasan, and John Stasko. “NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (Feb. 2021), pp. 369–379. ISSN: 2160-9306. DOI: 10.1109/tvcg.2020.3030378. URL: http://dx.doi.org/10.1109/TVCG.2020.3030378.

[12] Arvind Satyanarayan et al. “Vega-Lite: A Grammar of Interactive Graphics”. In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2017). URL: http://idl.cs.washington.edu/papers/vega-lite.

[13] Vidya Setlur et al. “Eviza: A Natural Language Interface for Visual Analysis”. In: UIST ’16 (2016), pp. 365–377. DOI: 10.1145/2984511.2984588. URL: https://doi.org/10.1145/2984511.2984588.

[14] Arjun Srinivasan and Vidya Setlur. “Snowy: Recommending Utterances for Conversational Visual Analysis”. In: Oct. 2021, pp. 864–880. DOI: 10.1145/3472749.3474792.

[15] Arjun Srinivasan et al. “Collecting and Characterizing Natural Language Utterances for Specifying Data Visualizations”. In: (May 2021). URL: https://www.microsoft.com/en-us/research/publication/collecting-and-characterizing-natural-language-utterances-for-specifying-data-visualizations/.

[16] Bowen Yu and Claudio T. Silva. “FlowSense: A Natural Language Interface for Visual Data Exploration within a Dataflow System”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 1–11. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934668. URL: http://dx.doi.org/10.1109/TVCG.2019.2934668.