Facilitating Conversational Interaction in Natural Language Interfaces for Visualization

Rishab Mitra[π*] Arpit Narechania[π†] Alex Endert[‡] John Stasko[§]

Georgia Institute of Technology

Abstract

Natural language (NL) toolkits enable visualization developers, who may not have a background in natural language processing (NLP), to create natural language interfaces (NLIs) for end-users to flexibly specify and interact with visualizations. However, these toolkits currently only support one-off utterances, with minimal capability to facilitate a multi-turn dialog between the user and the system. Developing NLIs with such conversational interaction capabilities remains a challenging task, requiring implementations of low-level NLP techniques to process a new query as an intent to follow-up on an older query. We extend an existing Python-based toolkit, NL4DV, that processes an NL query about a tabular dataset and returns an analytic specification containing data attributes, analytic tasks, and relevant visualizations, modeled as a JSON object. Specifically, NL4DV now enables developers to facilitate multiple simultaneous conversations about a dataset and resolve associated ambiguities, augmenting new conversational information into the output JSON object. We demonstrate these capabilities through three examples: (1) an NLI to learn aspects of the Vega-Lite grammar, (2) a mind mapping application to create free-flowing conversations, and (3) a chatbot to answer questions and resolve ambiguities.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools—Visualization toolkits; Human-centered computing—Human computer interaction (HCI)—Interaction techniques—Text input

1 Introduction and Background

Natural language interfaces (NLIs) for databases [5, 13, 14, 21, 28, 31, 44, 49] and visualizations [4, 10, 16, 18–20, 27, 29, 35–38, 41–43, 47] have shown great promise, democratizing access to data through the querying power and expressivity of natural language (NL). Given a dataset (e.g., movies), an NLI for visualization receives an NL query (e.g., *"Show the distribution of budget"*) as input, extracts data attributes (*Production Budget*) and analytic tasks (*Distribution*), and recommends one or more relevant visualizations (*Histogram*). Many of these NLIs also help resolve ambiguities that may occur during query interpretation. For example, DataTone [10] presents ambiguities through interactive GUI-based widgets (e.g., dropdowns) for disambiguation. Implementing such NLIs, however, requires experience with natural language processing (NLP) techniques and toolkits (e.g., NLTK [23], spaCy [15]) as well as GUI and visualization design tools (e.g., D3.js [6], Vega-Lite [33]), making it challenging for developers without the necessary skillset.

Recently, NL toolkits [9, 22, 25, 29] have enabled visualization developers, who may not have a background in NLP, to create new visualization NLIs or incorporate NL input within their existing systems. For example, given a tabular dataset and an NL query about the dataset, NL4DV generates an analytic specification comprising data attributes, analytic tasks (based on [2]), and visualizations (as Vega-Lite specifications [33]) modeled as a JSON object. However, these toolkits currently support one-off utterances (singleton queries) only, with minimal capability to facilitate a multi-turn dialog between the end-user and the system, e.g., by following-up on a previous query. Because of this, end-users would have to specify longer NL queries (e.g., *"Show the relationship between budget and rating for Action and Adventure movies that grossed over 100M")* to accomplish more complex tasks. These types of queries may also have a greater chance of failing (e.g., attribute detection can fail; filter operators may be incorrect), eventually warranting several paraphrasing attempts. We believe specifying multiple short queries in a natural sequence can enable end-users to incrementally accomplish a complex task, fix minor errors, and also make debugging easier, as in [3, 11, 16, 35, 38, 40]. This is called *conversational interaction* – "face-to-face or technology-mediated forms of interaction that use language, encompassing a wide range of different types of talk" [12].

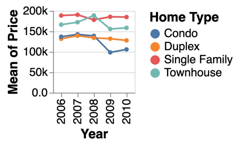
Developing NLIs with such conversational interaction capabilities remains a challenging task, however, requiring implementations of low-level NLP techniques to process a new query as an intent to follow-up on an older query, e.g., replacing an existing attribute with a new one. To the best of our knowledge, no NL toolkit facilitates conversational interaction, yet. Hence, in this work, we extend a Python-based toolkit, NL4DV [29], in order to enable visualization developers to facilitate multiple simultaneous conversations (through manual specification as well as automatic detection of intents to follow-up) and resolve associated ambiguities through an easy-to-use application programming interface (API). As a result, NL4DV also augments additional conversational information into the output JSON. We demonstrate these capabilities through three examples: (1) an NLI to learn aspects of Vega-Lite – an implementation of a grammar for interactive graphics [33], (2) a mind mapping application to create free-flowing conversations about a dataset, and (3) a chatbot to answer questions and resolve ambiguities in collaboration with the enduser. To support development of future systems, we open-source NL4DV and the described applications at **`https://nl4dv.github.io/nl4dv/`**.

2 Conversational Interaction with NL4DV

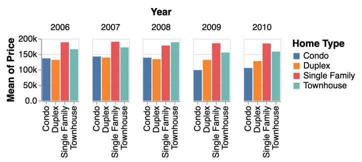
Listing 1 illustrates the basic Python code for developers to enable conversational interaction in their own applications using NL4DV. Given a tabular dataset on Houses (adapted from [7]; accessible at [17]) and a query string specified by the end-user, *"Show average prices for different home types over the years"*, with a single function call **`analyze_query(query)`** (lines 1-3), NL4DV first determines it as a standalone query (as it is the very first query), extracts data attributes and analytic tasks, recommends visualizations, and then assigns new objects that identify that conversation (**`dialogId`**="0") and the corresponding query (**`queryId`**="0") as part of the output JSON (lines 4-5). After observing the output visualization, if the end-user wants a bar chart instead of a line chart, they may ask, *"As a bar chart"* with a new parameter, **dialog**="auto". NL4DV automatically

[*]e-mail: rmitra34@gatech.edu
[†]e-mail: arpitnarechania4@gatech.edu
[‡]e-mail: endert@gatech.edu
[§]e-mail: stasko@cc.gatech.edu
[π]authors contributed equally

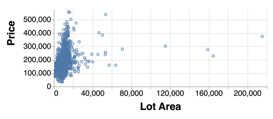
```
1  from nl4dv import NL4DV
2  nl4dv_instance = NL4DV(data_url="housing.csv")
3  resp_1 = nl4dv_instance.analyze_query("Show average prices
     ↪ for different home types over the years.")
4  print(resp_1)
5  # a new dialogId and a queryId get created.
   {
       "dialogId": "0",
       "queryId": "0",
       ...
   }
```



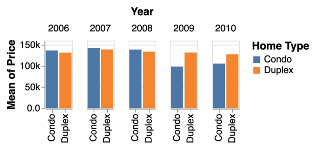
```
6  # this query is automatically inferred as a follow-up.
7  resp_2 = nl4dv_instance.analyze_query("As a bar chart.",
     ↪ dialog="auto")
8  print(resp_2)
   {
       "dialogId": "0",
       "queryId": "1",
       "followUpConfidence":
         "high", ...
   }
```



```
9  # this query is a new, standalone query.
10 resp_3 = nl4dv_instance.analyze_query("Correlate Price and
     ↪ Lot Area.", dialog=False)
11 print(resp_3)
   {
       "dialogId": "1",
       "queryId": "0",
       ...
   }
```



```
12 # this query follows up a specific, older query.
13 resp_4 = nl4dv_instance.analyze_query("Just show condos and
     ↪ duplexes.", dialog=True, dialog_id="0", query_id="1")
14 print(resp_4)
   {
       "dialogId": "0",
       "queryId": "2",
       ...
   }
```



Listing 1: Python code illustrating how developers can enable conversational interaction in their applications using NL4DV.

determines this as a follow-up to the previous query (with a heuristically determined **followUpConfidence**="high") and directly modifies its analytic specification, retaining the **dialogId**="0" but generating a new, now incremented **queryId**="1" as the second query in the conversation (lines 6-8). If the end-user is suddenly curious about how house prices compare with area, they may ask, *"Correlate price and area"*, explicitly specifying the query as standalone (**dialog**=**False**). This time, NL4DV increments **dialogId**="1" and resets **queryId**="0" since this is now the first query of a new, second conversation (lines 9-11). If the end-user wants to resume their original conversation and only focus on certain home types, they may ask, *"Just show condos and duplexes"*, this time explicitly specifying the query as a follow-up (**dialog**=**True**) with additional parameters: **dialog_id**="0", **query_id**="1", that correspond to the first conversation (lines 12-14). As expected, the resultant **dialogId**="0" and **queryId**="2", along with the filtered bar chart.

To achieve this kind of conversational interaction, we extended NL4DV [29]; Figure 1 illustrates the modified technical architecture. The existing **Query Processor** module parses the input NL query using NLP techniques such as tokenizing and parts of speech tagging (*Query Parser*), extracts data attributes through semantic and syntactic similarity matching (*Attribute Identifier*) and analytic tasks through dependency parsing (*Tasks Identifier*), and recommends relevant visualizations based on heuristics used in prior systems [26, 45, 46] (*Visualization Specification Generator*), that are

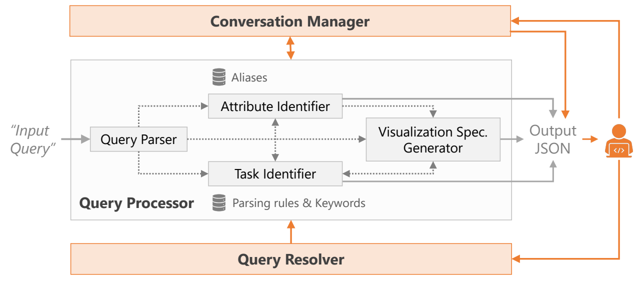


Figure 1: Original NL4DV architecture [29] (in gray) extended to support conversational interaction (in orange). The arrows indicate the flow of information between the modules.

```
all_dialogs = {
 "0": [{"query":"show the distribution of salaries as a
   ↪ boxplot",...}, {"query":"How about goals instead?",..}],
 "1": [{"query": "Show average goals per country",...},
   ↪ {"query": "now group by foot",...}],
 "2": [{"query": "correlate age and salary",...}, {"query":
   ↪ "now show only defenders",...}],
 "2.0.0": [{"query": "correlate age and salary",...},
   ↪ {"query": "what about only goalkeepers?",...}] #
   ↪ follow-up to query with dialog_id="2" and index=0
}
```

Listing 2: Data structure to store multiple conversations, including branches (multiple follow-ups to the same query).

combined into an *Output JSON*. The new **Conversation Manager** module enables developers to automatically determine or manually specify a query as a follow-up (or not). This module also determines the type of follow-up (e.g., *add* or *remove* attributes), managing all operations on the internal data structures. Also new, the **Query Resolver** module facilitates resolving NL ambiguities (e.g., by "medals" did the end-user mean "{Total | Gold | Silver | Bronze} Medals"?).

2.1 Facilitating Multiple Simultaneous Conversations

Following the dialog shown in Listing 1, whenever the end-user asks a new, standalone query, the **dialog_id** is also incremented by "1" and the **query_id** is re-initialized to "0" (identifiers are stringified after incrementing for efficient handling of data), creating a new dialog instance that is uniquely identifiable by **dialogId** and **queryId**. Subsequently, developers can explicitly follow up on specific queries by passing the follow-up query string along with additional input parameters: **dialog** (a boolean flag expressing an explicit intent to follow-up), **dialog_id**, and **query_id** to **analyze_query(query)**.

This design also enables end-users to ask multiple unrelated follow-ups to the same query. To create such conversational branches, developers can provide the same **dialog_id** and **query_id** in repeated calls to **analyze_query(query)**. Internally, NL4DV creates the desired branch point and outputs a new, unique **dialogId** with the format: "{dialog_id}.{query_id}.{branch_id}" (similar to the semantic versioning format [34]), where {branch_id} is the index of the branch stemming from the input parameters: {dialog_id} and {query_id}. This naming convention effectively represents the hierarchy of all entities involved in the conversation. Listing 2 shows how NL4DV stores these conversations in a Python dictionary of lists with **dialog_id**s as the keys and **query_id**s as the indexes of the corresponding list of queries. This data structure enables efficient retrieve, append, modify, and delete operations. Note that calling **analyze_query(query, dialog=True)**, without **dialog_id** or **query_id**, will make NL4DV follow up on the *most recent* **dialogId** and **queryId**; if these too do not exist (e.g., it is the very first conversation), then an error is thrown.

2.2 Detecting, Classifying and Processing Follow-ups

To supply **dialog**, **dialog_id** and **query_id** parameters to **analyze_query(query)**, developers have to provide GUI affordances for end-users, e.g., a checkbox to specify if **dialog**=True or not (and which conversation to follow-up on), which can be an unnatural end-user experience. To alleviate this, NL4DV offers a **dialog**="auto" setting (overloading the otherwise boolean input data type) that automatically determines if the query is a follow-up or not and outputs a **followUpConfidence** rating: {"high", "low", "none"} reflecting NL4DV's confidence in making the inference. This rating is heuristically determined based on the previous query, an **explicit_followup_keywords** map – keywords that convey natural conversational intents to follow-up (e.g., "add", "replace"), and an **implicit_followup_keywords** map – keywords that implicitly convey an intent to follow-up (e.g., "instead of", "only"). The **implicit_followup_keywords** are further classified as *non-ambiguous* – keywords that always convey an intent to follow-up (e.g. "instead of", "rather than") and *ambiguous* – keywords that can occur in a follow-up as well as standalone context (e.g., "only"). NL4DV assigns queries containing **explicit** keywords or **implicit** *non-ambiguous* keywords with a **high** **followUpConfidence** rating and **implicit** *ambiguous* keywords with a **low** **followUpConfidence** rating. For queries with no matching keywords, NL4DV compares the attributes, tasks, and visualizations of the current and the previous query and based on a heuristics and rule-based decision tree, assigns either a **low** or **none** **followUpConfidence** rating, the latter corresponding to a new, standalone query. For example, a query *"Show the average now."* is a compatible follow-up to its predecessor, *"Show maximum price across different home types."*; the desired change from "maximum" to "average" in the absence of any other follow-up keywords or attributes makes them compatible. Developers can override these default maps by supplying custom **explicit_followup_keywords** and **implicit_followup_keywords** objects through the **NL4DV()** constructor during initialization.

Next, the **explicit_followup_keywords** map classifies the follow-up query as one of three types: *add*, *remove*, or *replace* (inspired by Evizeon's *continue*, *retain*, *shift* transitional states [16]) and maps it to one or more components of an analytic specification: *data attributes*, *analytic tasks*, and *visualizations*. Note that the resultant combinations (e.g., *replace* + *data attribute*) are not always mutually exclusive, e.g., replacing an attribute can sometimes also modify the task(s) and/or the visualization(s). Lastly, NL4DV references the parent query's (the query being followed upon) analytic specification and makes necessary associations (e.g., creating new conversational branches) and modifications (e.g., dropping an existing attribute), eventually generating a new specification as a JSON object.

By configuring the keyword maps and supplying methods with appropriate parameters, end-users can *add*, *remove*, or *replace* data attributes, either **explicit**ly, e.g., *"Replace budget with gross"*–which makes a direct reference to the data attributes and the follow-up task; or **implicit**ly, e.g., *"Now show only budget"*–which indirectly suggests to remove all other attributes except "Production Budget". Unlike attributes, following up on tasks is different because end-users are unaware of the associated technical jargon, e.g. *"Add Find Extremum to Worldwide Gross"* is not a natural query an end-user would ask; they would rather say, *"Show me the highest grossing movie"*, which would then infer the *Find Extremum* task [2] (through 'highest'). Thus, most queries that follow-up on tasks are **implicit** in nature. NL4DV currently supports *sort* (e.g., *"Sort by budget in an ascending order"*), *find extremum* (e.g., *"Which of these genres has the smallest budget?"*), *filter* (e.g., *"Now show only action movies"*), and *derived value* (e.g., *"Replace average with sum"*) tasks [2]. A follow-up to *add* (or *remove*) a visualization is meaningless as there will (or must) always be some recommended chart. *Replace* is the only meaningful task and it can be **explicit** (e.g., *"Replace this line chart with a bar chart"*) or **implicit** (e.g., *"As a bar chart instead"*).

```
1 from nl4dv import NL4DV
2 nl4dv_instance = NL4DV(data_url="olympic_medals.csv")
3 init_response = nl4dv_instance.analyze_query("Show medals in
  ↪ hockey and skating by country.")
4 # Multiple ambiguities are detected from the query.
5 print(init_response)
  { "dialogId": "0", "queryId": "0",
    "ambiguities": {
      "attribute": {
        "medals": { "options": ["Bronze Medal","Gold
        ↪ Medal","Silver Medal", "Total Medal"],
                    "selected": null}},
      "value": {
        "skating": { "options": ["Figure Skating", "Short Speed
        ↪ Skating", "Speed Skating"],
                     "selected": null},
        "hockey": { "options": ["Hockey", "Ice Hockey"],
                    "selected": null}}
    }, ... }
6 resolved_response = nl4dv_instance.update_query({"attribute":
  ↪ { "medals": "Total Medal" }, "value": {"skating": "Speed
  ↪ Skating", "hockey": "Ice Hockey"}})
7 print(resolved_response)
8 # The "selected" property is updated in the response.
  {"dialogId":"0","queryId":"0","ambiguities":{...}, ...}
```

Listing 3: Python code illustrating how NL4DV helps resolve ambiguities via **update_query(obj)** (line 6).

2.3 Resolving Ambiguities during Query Interpretation

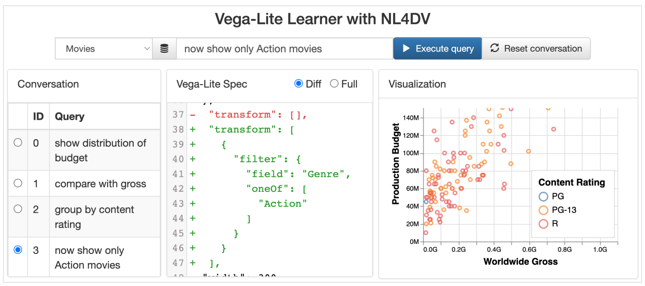
Natural language (NL) is often ambiguous and underspecified, e.g., consider the query, *"Show medals in hockey and skating by country"* regarding a dataset on Olympic Medals (adapted from [32]; accessible at [30]). Here, "medals" (**attribute**) could be mapped to either of ["Total Medals", "Gold Medals", "Silver Medals", "Bronze Medals"], "hockey" (**value**) could be mapped to either of ["Ice Hockey", "Hockey"], and "skating" could be mapped to either of ["Figure Skating", "Speed Skating", "Short Speed Skating"].

These ambiguities can cause problems while processing ambiguous follow-up queries (e.g., *"Sort by medals"*–Which type of "medals"?), and hence must be resolved a priori. NL4DV detects these **attribute**-level and **value**-level ambiguities and makes them accessible in the output JSON under a new key, **ambiguities**. In addition, NL4DV now provides a new function **update_query(obj)**, to help developers design experiences that resolve ambiguities directly through the toolkit, also enabling accurate processing of subsequent follow-up queries. Listing 3 illustrates how **update_query(obj)** (line 6) takes a Python dictionary as input, that includes the types of ambiguities ("attribute" and "value"), the corresponding keywords in the query ("medals", "hockey", and "skating"), and the corresponding entities selected by the end-user for resolution. NL4DV then updates the **selected** entities under **ambiguities** as well as the **attributeMap** and the **taskMap**, recommending a new **visList** (visualizations). Note that developers may not always provide end-users with affordances to resolve such ambiguities. In these cases, NL4DV automatically resolves ambiguities by itself, selecting the entities that have the highest string-based similarity score with the corresponding query keyword, and calling **update_query(obj)**. In case of ties, entities that were detected first are selected.

3 Creating Visualization Systems with NL4DV

3.1 NL-Driven Vega-Lite Learner

The NL-Driven Vega-Lite Editor in NL4DV [29] demonstrated how NL can be used to create, edit, and hence learn the Vega-Lite [33] grammar. However, end-users of this system need to be proficient with the Vega-Lite syntax (e.g., properties and operators) to be able to successfully edit the specifications output by NL4DV. Figure 2 illustrates the user interface of a similar NL-Driven Vega-Lite Learner that demonstrates how conversational interaction can help users


Vega-Lite Learner with NL4DV

Movies | now show only Action movies | Execute query | Reset conversation

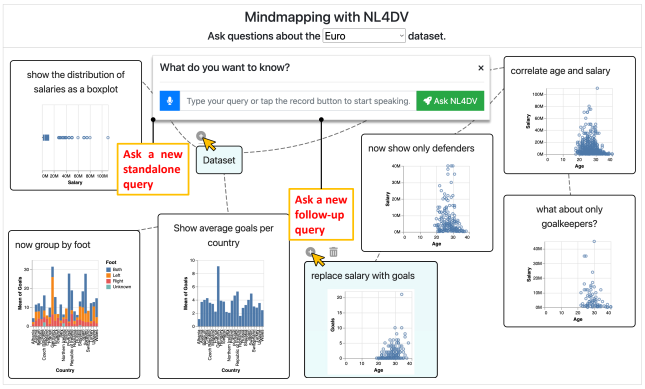
Conversation

	ID	Query
○	0	show distribution of budget
○	1	compare with gross
○	2	group by content rating
●	3	now show only Action movies

Vega-Lite Spec [✓ Diff] Full

```
-  "transform": [],
+  "transform": [
+    {
+      "filter": {
+        "field": "Genre",
+        "oneOf": [
+          "Action"
+        ]
+      }
+    }
+  ],
```

Figure 2: An NLI that helps users learn Vega-Lite syntax (e.g., the *transform* property to apply filters), through NL queries.


Mindmapping with NL4DV

Ask questions about the Euro dataset.

What do you want to know?

Type your query or tap the record button to start speaking. | Ask NL4DV

show the distribution of salaries as a boxplot

Ask a new standalone query

Dataset

Ask a new follow-up query

correlate age and salary

now show only defenders

what about only goalkeepers?

Show average goals per country

now group by foot

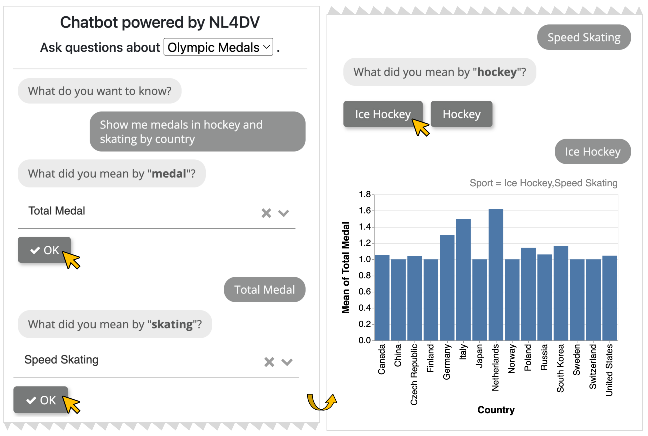
replace salary with goals

Figure 3: A mind mapping app that enables users to have free-flowing conversations about a dataset. Three dialogs with follow-ups are connected to the "Dataset" via dashed lines.

learn this grammar *better* by sequentially processing short, specific NL intents and incrementally building the Vega-Lite specification, helping users learn the syntax changes required to achieve the corresponding intents. Users ask a series of short NL queries and observe the resultant Vega-Lite specifications. These queries are chained together, forming a conversation. Users can see the *diff* (i.e., added and removed entities) between the Vega-Lite specifications of the selected query and its predecessor through code highlights (green implies addition; red implies deletion). For example, a follow-up query to apply a filter, *"Now show only Action movies"* generates a Vega-Lite specification that differs from the previous query's specification in terms of the *transform* property, helping the user learn how Vega-Lite filters are specified. To develop this interface, developers can sequentially call **`analyze_query(query, dialog=True)`** and then focus on computing the *diffs* between the Vega-Lite specifications of the query and its predecessor and programming the layout, styling, and interactivity aspects using HTML, CSS, JavaScript.

3.2 Mind Mapping Conversations about a Dataset

In this second use-case, we demonstrate how the input parameters: **dialog**, **dialog_id** and **query_id** in **`analyze_query(query)`** can help end-users engage in multiple simultaneous conversations, unlike the NL-Driven Vega-Lite Learner, that supports only one conversation at-a-time. Figure 3 illustrates the user interface of a mind mapping application that helps users engage in free-flowing conversations regarding a European soccer players dataset (adapted from [1]; accessible at [8]). Listing 2 shows the corresponding data structure maintained by NL4DV. Through speech or text input, users can ask standalone queries (e.g., *"Correlate age and salary"*) as well as follow-up queries (e.g., *"Now show only defenders"*) by clicking the plus icon, enabled by hovering on the corresponding mind map node (the rectangular block). Users can also follow-up on already followed-up queries, forming new conversational branches (e.g., *"What about only goalkeepers?"*). To develop this interface, developers can call **`analyze_query(query, dialog, dialog_id, query_id)`**, supplying the dialog and query identifiers based on the corresponding query that is to be followed-upon. Then, based on the newly generated **`dialogId`** and **`queryId`**, a new node is created and appended to the corresponding predecessor query node.


Chatbot powered by NL4DV

Ask questions about Olympic Medals.

What do you want to know?

Show me medals in hockey and skating by country

What did you mean by "**medal**"?

Total Medal

✓ OK

Total Medal

What did you mean by "**skating**"?

Speed Skating

✓ OK

Speed Skating

What did you mean by "**hockey**"?

Ice Hockey | Hockey

Ice Hockey

Figure 4: A chatbot where the system collaborates with the user to resolve ambiguities during query interpretation.

3.3 Collaboratively Resolve Ambiguities in a ChatBot

In this third use-case, we demonstrate how NL4DV's **Query Resolver** can help resolve ambiguities that often occur in natural language. Figure 4 illustrates a standard chatbot user interface that presents DataTone-like [10] "ambiguity widgets"–dropdowns and buttons. End-users can disambiguate by interacting with the widgets, notifying NL4DV through a function call to **`update_query(obj)`**. After all ambiguities are resolved, the system renders the now-unambiguous visualization. To develop this interface, developers can loop through the **`ambiguities`** object in the output JSON and present corresponding **`options`** to the end-user, e.g., as options in a select dropdown. As the end-user makes their choices, a function call to **`update_query(obj)`** will resolve the ambiguity, updating the **`selected`** properties in the output JSON. Listing 3 illustrates this data exchange between the user interface and NL4DV.

4 Conclusion, Limitations, and Future Work

In this work, we extend an existing natural language (NL) for data visualization toolkit, NL4DV, to enable developers to integrate conversational interaction capabilities within natural language interfaces. We demonstrate NL4DV's capabilities through three examples and open-source the toolkit at **`https://nl4dv.github.io/nl4dv/`**.

While testing, we noted certain conversational ambiguities, e.g., if a query, *"Show only Action movies"* is followed-up with *"What about R-rated movies?"* does the user mean to augment the previous filter or replace it with the new one? Consider another query, *"Visualize budget distribution as a histogram instead of a boxplot"*; here, the user means to ask a standalone query, but the presence of "instead of" (an implicit follow-up keyword) will make NL4DV wrongly treat it as a follow-up. We will address these ambiguities and translation errors in future releases. We also plan a formal performance evaluation of the toolkit. However, unlike conversational text-to-SQL dataset benchmarks (e.g., CoSQL [48]), there are currently no such benchmarks for visualization tasks. An area of future work, thus, for current text-to-visualization datasets [9, 24, 39], that focus on singleton utterances, is to include multi-turn utterances.

Acknowledgments

This work was supported in part by an NSF Grant IIS-1717111. We thank Arjun Srinivasan and the Georgia Tech Visualization Lab.

References

[1] G. Aisch. The Clubs That Connect The World Cup. https://www.nytimes.com/interactive/2014/06/20/sports/worldcup/how-world-cup-players-are-connected.html, 2014.

[2] R. Amar, J. Eagan, and J. Stasko. Low-level Components of Analytic Activity in Information Visualization. In *IEEE Symposium on Information Visualization*, 2005. doi: 10.1109/INFVIS.2005.1532136

[3] Amazon Alexa. https://www.amazon.com/smart-home-devices.

[4] Amazon Quicksight. https://aws.amazon.com/quicksight/.

[5] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger. SODA: Generating SQL for Business Users. *Proceedings of the VLDB Endowment*, 2012. doi: 10.14778/2336664.2336667

[6] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-Driven Documents. *IEEE TVCG*, 2011. doi: 10.1109/TVCG.2011.185

[7] D. De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*, 2011. doi: 10.1080/10691898.2011.11889627

[8] euro.csv. https://github.com/nl4dv/nl4dv/blob/master/examples/assets/euro.csv.

[9] S. Fu, K. Xiong, X. Ge, S. Tang, W. Chen, and Y. Wu. Quda: Natural Language Queries for Visual Data Analytics. *arXiv preprint arXiv:2005.03257*, 2020.

[10] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *ACM UIST*, 2015. doi: 10.1145/2807442.2807478

[11] Google Home. https://developers.google.com/home.

[12] M. Haugh. Conversational Interaction. *The Cambridge handbook of pragmatics*, 2012. doi: 10.1017/CBO9781139022453.014

[13] P. He, Y. Mao, K. Chakrabarti, and W. Chen. X-SQL: Reinforce Schema Representation with Context. *arXiv preprint arXiv.1908.08113*, 2019.

[14] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. TAPAS: Weakly Supervised Table Parsing via Pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

[15] M. Honnibal and I. Montani. spacy 2: Natural Language Understanding with Bloom Embeddings. *Convolutional Neural Networks and Incremental Parsing*, 2017.

[16] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying Pragmatics Principles for Interaction with Visual Analytics. *IEEE TVCG*, 2018. doi: 10.1109/TVCG.2017.2744684

[17] housing.csv. https://github.com/nl4dv/nl4dv/blob/master/examples/assets/data/housing.csv.

[18] J.-F. Kassel and M. Rohs. Valletto: A Multimodal Interface for Ubiquitous Visual Analytics. In *ACM CHI Extended Abstracts*, 2018. doi: 10.1145/3170427.3188445

[19] D. H. Kim, E. Hoque, and M. Agrawala. Answering Questions about Charts and Generating Visual Explanations. In *ACM CHI*, 2020. doi: 10.1145/3313831.3376467

[20] A. Kumar, J. Aurisano, B. Di Eugenio, A. Johnson, A. Gonzalez, and J. Leigh. Towards a Dialog System that Supports Rich Visualizations of Data. In *SIGDIAL*, 2016. doi: 10.18653/v1/W16-3639

[21] F. Li and H. V. Jagadish. NaLIR: an interactive natural language interface for querying relational databases. In *ACM SIGMOD*, 2014. doi: 10.1145/2588555.2594519

[22] C. Liu, Y. Han, R. Jiang, and X. Yuan. ADVISor: Automatic Visualization Answer for Natural-Language Question on Tabular Data. In *IEEE PacificVis*, 2021. doi: 10.1109/PacificVis52677.2021.00010

[23] E. Loper and S. Bird. NLTK: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[24] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *ACM SIGMOD*, 2021. doi: 10.1145/3448016.3457261

[25] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural Language to Visualization by Neural Machine Translation. *IEEE TVCG*, 2021. doi: 10.1109/TVCG.2021.3114848

[26] J. Mackinlay, P. Hanrahan, and C. Stolte. Show Me: Automatic Presentation for Visual Analysis. *IEEE TVCG*, 2007. doi: 10.1109/TVCG.2007.70594

[27] Microsoft Power BI. https://powerbi.microsoft.com/en-us.

[28] A. Narechania, A. Fourney, B. Lee, and G. Ramos. DIY: Assessing the Correctness of Natural Language to SQL Systems. In *ACM IUI*, 2021. doi: 10.1145/3397481.3450667

[29] A. Narechania, A. Srinivasan, and J. Stasko. NL4DV: A Toolkit for Generating Analytic Specifications for Data Visualization from Natural Language Queries. *IEEE TVCG*, 2021. doi: 10.1109/TVCG.2020.3030378

[30] olympics_medals.csv. https://github.com/nl4dv/nl4dv/blob/master/examples/assets/data/olympics_medals.csv.

[31] P. Pasupat and P. Liang. Compositional Semantic Parsing on Semi-Structured Tables. In *ACL IJCNLP*, 2015. doi: 10.3115/v1/P15-1142

[32] rgriffin (Kaggle username). 120 years of Olympic history: athletes and results. https://www.kaggle.com/datasets/heesoo37/120-years-of-olympic-history-athletes-and-results.

[33] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-Lite: A Grammar of Interactive Graphics. *IEEE TVCG*, 2016. doi: 10.1109/TVCG.2016.2599030

[34] Semantic versioning. https://semver.org/.

[35] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A Natural Language Interface for Visual Analysis. In *ACM UIST*, 2016. doi: 10.1145/2984511.2984588

[36] V. Setlur, M. Tory, and A. Djalali. Inferencing Underspecified Natural Language Utterances in Visual Analysis. In *ACM IUI*, 2019. doi: 10.1145/3301275.3302270

[37] A. Srinivasan, B. Lee, N. H. Riche, S. M. Drucker, and K. Hinckley. InChorus: Designing Consistent Multimodal Interactions for Data Visualization on Tablet Devices. In *ACM CHI*, 2020. doi: 10.1145/3313831.3376782

[38] A. Srinivasan, B. Lee, and J. T. Stasko. Interweaving Multimodal Interaction with Flexible Unit Visualizations for Data Exploration. *IEEE TVCG*, 2020. doi: 10.1109/TVCG.2020.2978050

[39] A. Srinivasan, N. Nyapathy, B. Lee, S. M. Drucker, and J. Stasko. Collecting and Characterizing Natural Language Utterances for Specifying Data Visualizations. In *ACM CHI*, 2021. doi: 10.1145/3411764.3445400

[40] A. Srinivasan and V. Setlur. Snowy: Recommending Utterances for Conversational Visual Analysis. In *ACM UIST*, 2021. doi: 10.1145/3472749.3474792

[41] A. Srinivasan and J. Stasko. Orko: Facilitating Multimodal Interaction for Visual Exploration and Analysis of Networks. *IEEE TVCG*, 2018. doi: 10.1109/TVCG.2017.2745219

[42] Y. Sun, J. Leigh, A. Johnson, and S. Lee. Articulate: A Semi-automated Model for Translating Natural Language Queries into Meaningful Visualizations. In *Proceedings of the International Symposium on Smart Graphics*, 2010. doi: 10.1007/978-3-642-13544-6_18

[43] Tableau Ask Data. https://www.tableau.com/about/blog/2018/10/announcing-20191-beta-96449.

[44] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh. Robust Text-to-SQL Generation with Execution-Guided Decoding. *arXiv preprint arXiv:1807.03100*, 2018.

[45] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE TVCG*, 2015. doi: 10.1109/TVCG.2015.2467191

[46] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *ACM CHI*, 2017. doi: 10.1145/3025453.3025768

[47] B. Yu and C. T. Silva. FlowSense: A Natural Language Interface for Visual Data Exploration within a Dataflow System. *IEEE TVCG*, 2020. doi: 10.1109/TVCG.2019.2934668

[48] T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, et al. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. *ACL EMNLP-IJCNLP*, 2019. doi: 10.18653/v1/D19-1204

[49] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103*, 2017.