Abstract

This paper presents the problem of conversational plotting agents that carry out plotting actions from natural language instructions. To facilitate the development of such agents, we introduce CHARTDIALOGS, a new multi-turn dialog dataset, covering a popular plotting library, matplotlib. The dataset contains over 15,000 dialog turns from 3,200 dialogs covering the majority of matplotlib plot types. Extensive experiments show the best-performing method achieving 61% plotting accuracy, demonstrating that the dataset presents a non-trivial challenge for future research on this task.

1 Introduction

Advances in machine language understanding (Hirschberg and Manning, 2015) have sparked interest in using artificial intelligence to address difficult problems involving language. In this work, we are interested in the problem of plotting via natural language instructions. Plotting is a method for visualizing data and mathematical functions. Plotting libraries such as matplotlib support functionality on a range of levels, from general, “change the X-axis from linear to log scale”, to specific, “color this screen pixel red”. Yet, using such libraries can be difficult for novice users and time consuming even for experts. This obstacle, coupled with the increasing popularity of the scientific method of gleaning information from data (Hey et al., 2009; Dhar, 2013), motivates our objective of designing natural language interfaces (NLIs) for plotting.

NLIs for plotting can be organized into three categories based on what the user is expected to describe: the data, the function, or the plot.

Describing the Data or the Function. In the first category of plotting NLIs, users are expected to describe the data they would like to visualize, by posing queries such as: “Show me medals for hockey and skating by country.” Queries may involve simple data analysis: “Is there a seasonal trend for bike usage?” The system retrieves the relevant data, performs simple data analysis, and produces a visualization. This category of NLIs has been studied in Human Computer Interaction and related areas (Gao et al., 2015; Setlur et al., 2016; Srinivasan and Stasko, 2017; Yu and Silva, 2019; Sun et al., 2010).

In the second category of plotting NLIs, users specify the function they would like to visualize. In this category, commercial products such as wolframalpha.com yield results for queries such as “plot the tangent to $x^2$ at $x = 0.5$”. The system processes such queries by leveraging knowledge of functions and mathematical principles.

Describing the Plot. In the two categories we have discussed, users only describe what data or function they would like to visualize without describing how to visualize it. The system is in charge of all plotting details, which are not accessible to users. We can think of a third, less explored, category of plotting NLIs, in which the user instructs the system on how they would like to manipulate a plot. As an example, consider the following questions from a community question answering forum for matplotlib 1:

(Q1): “How does one change the font size for all elements (ticks, labels, title) on a matplotlib plot?”

(Q2): “I have a scatter plot graph . . . I would like the Y-Axis to start at the max value and go up to 0.”

(Q3): “Given a signal plot with time index ranging from 0 to 2.6(s), I want to draw vertical red lines indicating corresponding time index for the list.”

1https://stackoverflow.com/questions/tagged/matplotlib
Figure 1: Illustration of two of CHARTDIALOGS plot types. (a) Line Chart has slots such as Line Style. (b) A 3D Surface has slots such as Surface Color.

For Q1, the user’s intent is to change the font size of the elements of a plot; for Q2, to invert the Y-axis on the plot; and for Q3, to add vertical lines to a plot. All three questions seek to perform an action directly on the plot. The large number of such questions online indicates that direct plot manipulation is a common technical need. Crucially, expressing these intents in natural language is often faster than perusing the documentation of plotting library. Therefore, there is an opportunity to automatically process such intents by mapping natural language to API calls. This problem is the focus of our work.

Contributions. The contributions of this work are as follows: 1) We identify and define the problem of conversational plotting agents. 2) To facilitate work on this problem, we present a large dataset, CHARTDIALOGS, that consists of written multi-turn plotting dialogs. An in-depth analysis of the data shows that it is linguistically diverse and compares favorably to existing datasets. 3) We conducted extensive experiments in the framework of goal-oriented dialog using various methods. We also collected data on human performance, finding that there is a substantial gap between model and human performance, and therefore room for future work.\(^2\)

2 Problem Definition

Our goal is to develop a conversational plotting agent that takes natural language instructions and updates the plot accordingly. The agent is conversational because plots can be complex, making it difficult to describe everything at once. Users may want to fine-tune the appearance of their plot through multiple turns.

Goal-Oriented Dialog Problem. We treat the conversational plotting agent problem as an instance of slot-based goal-oriented dialog. The applicable slots are plot type specific. Figure 1 illustrates example slots for some of the plot types. Different plot types have different slots. However, some slots are shared across plot types. For example, the slot “X-axis scale” is relevant to the x-axis, thus it is applicable in any plot type with an x-axis, including line chart, bar plot, contour plot, etc. This slot can take a value such as “X-axis scale = log”, as a result of a request such as “change the x-axis scale from linear to log”.\(^3\)

Illustrations of all CHARTDIALOGS plot types and their slots are provided in Appendix A.

3 Related Work

Goal-oriented dialog datasets largely focus on service domains such as airlines (Hemphill et al., 1990; Seneff and Polifroni, 2000; Bennett and Rudnicky, 2002; Asri et al., 2017; Budzianowski et al., 2018; Wei et al., 2018), restaurant (Henderson et al., 2014; Bordes et al., 2017), bus (Raux et al., 2005; Williams et al., 2013), technical support (Lowe et al., 2015), and car agents (Eric et al., 2017). Recently, a multi-domain goal-oriented dataset covering restaurant, attraction, hospital, police, hotel, taxi and train domains was introduced in (Budzianowski et al., 2018). Our dataset is focused

\(^2\)We have released our dataset and code for experiments: https://github.com/sythello/ChartDialog

\(^3\)We wrote a simple script to take as input the plot type (as a special slot) and other slot-value pairs, to generate the actual plot image using matplotlib. This script is included in the released dataset.
on a new domain, which is data plots.

Natural language interfaces to structured languages such as SQL have been explored in Databases (DB) (Li and Jagadish, 2014), Programming Languages (PL) (Yaghmazadeh et al., 2017), and NLP (Zelle and Mooney, 1996). While the problem of language to SQL is different from language to plots, both problems need to deal with the difficulty of automatically interpreting natural language and mapping it to an unambiguous structured representation.

Closer to our work is the task of conversational image editing (Manuvnikurike et al., 2018b,a), whose aim is to enable queries like “Can you please fix the glare on my dog’s eyes”. Although both focus on image manipulation, the images and manipulations are different in the two domains. Additionally, we provide structured representations from which the plot images are generated. Our experiments show that such representations provide useful information for model training. In contrast, the structured representation is not available in conversational image editing. Furthermore, our dataset contains over 3,200 dialogs in comparison to the 129 dialogs for image edits.

Lastly, our task is different from full-fledged program synthesis, which takes natural language as input and produces computer programs in a language such as Python (Church, 1957; Solar-Lezama and Bodik, 2008). Our task is simpler and more structured.

4 Data Collection

To facilitate data collection, we make use of structured representations which we call text plot specifications.

Definition 1 (Text Plot Specification, TPSpec). Let $S^t$ be the set of all relevant slots for a given plot type, $t$, where $t$ takes on plot type values such as histogram, scatter, etc. For each slot $s_i \in S^t$, let the set of values it can take be $V_i^t$. A TPSpec of plot type $t$ is given by:

$$TP^t = \{(s_1 : v_1, s_2 : v_2, \ldots) : s_i \in S^t; v_i \in V_i^t\}$$

Thus a TPSpec is a sequence of tokens and can be considered as a structured text representation of a plot. This representation is invertible, i.e. a TPSpec can be mapped back to its corresponding slot-value pairs in a deterministic way. The design of TPSpecs is similar to how structured representations are used for dialog state tracking (Kan et al., 2018). We leverage TPSpecs in our data collection pipeline, which consists of two steps.

Step 1: Plot Generation. The first step consists of generating a set of matplotlib plots. Since there is a one-to-one mapping between Text Plot Specifications (TPSpecs) and plot images, we only need to generate TPSpecs. Specifically, for each plot type $t$ and all relevant slots $s_i \in S^t$, we design a value pool $P_i^t \subseteq V_i^t$, from which we randomly sample slot values to generate TPSpec samples.

Step 2: Dialog Collection. The second step involves collecting dialogs about the plots we generate in Step 1. A widely-used dialog collection scheme is the Wizard-of-Oz (WOZ) (Kelley, 1984), in which one worker plays the user and another worker plays the computer. Successful dialog datasets have been collected using Wizard-of-Oz approach, including the Air Travel Information System (ATIS) corpus (Hemphill et al., 1990), and others (Budzianowski et al., 2018; Rojas-Barahona et al., 2017; Asri et al., 2017).

We designed Wizard-of-Oz\(^4\) Mechanical Turk (MTurk) tasks to have a Describer worker, who plays the role of the user; and an Operator worker, who plays the role of the plotting agent\(^5\). The Describer has access to a target plot which is the goal plot for the Operator to achieve, but it is not directly visible to the Operator; the Operator has access to an operation panel which consists of a changeable field for each slot. The Operator can use this panel to execute a plot function on a server. Both workers have access to the working plot which is the plot that the Operator has generated based on the Describer’s requests. It is initialized to a placeholder empty plot.

The Describer begins the conversation by writing a message in natural language, describing to the Operator a request that would take them closer to their goal of matching the working plot with the target plot. The Describer could say “invert the Y-axis”. The Operator can respond in natural language to ask clarification questions, or fill out slots in the operation panel and show the resulting plot to the Describer. For example, the operator might select the slot corresponding to “invert Y-axis= True” and the working plot is updated for both workers to see. The describer would continue

\(^4\) Our setting is slightly different from the usual Wizard-of-Oz in that users were informed that they were conversing with fellow humans.

\(^5\) Multi-worker MTurk tasks are implemented using ParlAI (Miller et al., 2017)
The collected dataset, CHARTDIALOGS, consists of 3,284 dialogs, 15,754 dialog turns and 141,876 tokens in total. **Comparison to other Datasets.** Table 1 compares our dataset to other goal-oriented datasets that are about a **single** domain, such as travel, restaurant, car, etc., on several key metrics. In particular, we compare to: DSTC2 (Henderson et al., 2014), SFX (Gašić et al., 2014), WOZ (Wen et al., 2017), FRAMES (Asri et al., 2017), KVRET (Eric and Manning, 2017), M2M (Shah et al., 2018) and ImageEdits (Manuvakurike et al., 2018b,a). Table 1 shows that our corpus compares favorably to other datasets and is strong on two metrics: number of dialogs, and number of slots. This is a positive indication, given the narrowness of our domain in comparison to other domains.

**Naturalness of Utterances.** We took a pre-trained language model, the Generative Pre-trained Transformer (GPT-2) of OpenAI (Radford et al., 2019), to evaluate the naturalness of utterances in our dataset. This language model is trained on Web text, which while different from our domain, can be a good measure of language naturalness, at least for generic texts. Figure 2a shows GPT-2 perplexity distribution for half of the utterances, 7,876, in CHARTDIALOGS. This half consists of the utterances with the lowest perplexity. The second half with higher perplexity forms a long-tail distribution and is omitted for plot readability.

As shown in Figure 2a, the dataset contains utterances of varying degrees of naturalness, from pure natural language (“please invert the Y-axis”), to a

### Table 1: Comparison of CHARTDIALOGS to other single domain goal-oriented dialog data sets.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># Dialogues</strong></td>
<td>1,612</td>
<td>1,006</td>
<td>600</td>
<td>1,369</td>
<td>2,425</td>
<td>1,500</td>
<td>129</td>
<td>3,284</td>
</tr>
<tr>
<td><strong>Total # turns</strong></td>
<td>23,354</td>
<td>12,396</td>
<td>4,472</td>
<td>19,986</td>
<td>12,732</td>
<td>14,796</td>
<td>8,890</td>
<td>15,754</td>
</tr>
<tr>
<td><strong>Total # tokens</strong></td>
<td>199,431</td>
<td>108,975</td>
<td>50,264</td>
<td>251,867</td>
<td>102,077</td>
<td>121,977</td>
<td>59,653</td>
<td>141,876</td>
</tr>
<tr>
<td><strong>Avg. turns per dia.</strong></td>
<td>14.49</td>
<td>12.32</td>
<td>7.45</td>
<td><strong>14.60</strong></td>
<td>5.25</td>
<td>9.86</td>
<td>unk</td>
<td>14.60</td>
</tr>
<tr>
<td><strong>Avg. tokens per turn</strong></td>
<td>8.54</td>
<td>8.79</td>
<td>11.24</td>
<td><strong>12.60</strong></td>
<td>8.02</td>
<td>8.24</td>
<td>unk</td>
<td>9.01</td>
</tr>
<tr>
<td><strong>Total unique tokens</strong></td>
<td>986</td>
<td>1,473</td>
<td>2,142</td>
<td><strong>12,043</strong></td>
<td>2,842</td>
<td>1,008</td>
<td>2,299</td>
<td>2,652</td>
</tr>
<tr>
<td><strong># Slots</strong></td>
<td>8</td>
<td>14</td>
<td>4</td>
<td><strong>61</strong></td>
<td>13</td>
<td>14</td>
<td>unk</td>
<td>53</td>
</tr>
<tr>
<td><strong># Values</strong></td>
<td>212</td>
<td>1847</td>
<td>99</td>
<td>3871</td>
<td>1363</td>
<td>138</td>
<td>unk</td>
<td>328</td>
</tr>
</tbody>
</table>

*M2M is largely on the restaurant domain but also includes movies.*

### Figure 2: (a) Distribution of perplexity of the utterances. (b) and (c) average per word surprise of a growing sentence as new words are added to the sentence. High perplexity is a result of plot-specific terms like ‘outline’, and ‘cap’ arising in unexpected contexts.
structured code-style language (“Y-axis= inverted”). This is inline with our goal to have a conversational plotting agent that deals with requests with different levels of naturalness. The average perplexity even on the first half is high at 399.77. The second half, not shown, has median perplexity of 3,776.0, and mean perplexity of 77188.58.

Figures 2b and 2c show the perplexity behavior for two utterances. The figures show the average per-word surprise of a growing sentence as new words are added to the sentence. For example, in Figure 2b, the perplexity for “add a” is low, increases for “add a black”, increases even more for “add a black outline”, and decreases for “add a black outline to”. It is clear that high perplexity of the dataset is a result of plot-specific terms like ‘outline’ in Figure 2b and ‘cap’ in Figure 2c, arising in unexpected contexts in Web text.

**Turns Per Plot Type.** Figure 3a shows the fraction of dialog turns per plot type. Some plot types have more dialogs and more turns than others, which is a design choice we made in collecting the dataset. Although not the subject of the current paper, we would like the plotting agent to generalize to plot types with few data points, and potentially, to plot types that were never seen before, as a challenge for few-shot or zero-shot learning methods.

**Utterance Length.** Figures 3b shows that our dataset has utterances of varying lengths in terms of tokens. The average number of tokens per utterance is 9.01, which is comparable to the average among all the datasets reported in Table 1, which is 9.57.

**Utterance Syntactic Depth.** Figures 3c shows the distribution of constituency parse tree depths from the Stanford Parser. The average tree depth is 4.5. Figure 4 shows two parse trees of different depths. The parse tree in Figure 4a for the utterance “Add a black outline to the chart” has a tree depth of 4, and reflects the nature of the average utterance. On the other hand, the parse tree in Figure 4b for “Increase the cap size of the error bar but don’t touch the thickness” shows a more complex utterance with a tree depth of 8. We also show the most common top-level constituent combinations in Figure 5 in the Appendix.

### 6 Methods

To study the feasibility of developing conversational plotting agent using CHARTDIALOGS, we assess the performance of various methods.

The main methods we evaluate build on the sequence-to-sequence (seq2seq) framework.
Seq2seq models employ two components: an encoder and a decoder. The encoder produces hidden states of the input. Attention is used to produce a weighted sum of the encoder hidden states, known as the context vector $c_t^*$. The decoder defines the joint probability of an output sequence $y = (y_1, \cdots, y_n)$ as:

$$p(y) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c_t^*)$$

\textbf{Input.} We treat each plot update as a separate datapoint. For each datapoint, the input comes from three available sources: i) current state as represented by the text plot specification (TPSpec), ii) current state as represented by the plot image, and iii) the dialog history. In principle, the entire dialog history can be considered. In our experiments, we consider all utterances from the last plot update to the current one from both interlocutors. In other words, starting from the last plot update, the Descriptor’s instruction and all the clarification questions and responses are concatenated and provided as the dialog history.

\textbf{Output.} We formulate the model output as the update needed from the current TPSpec to the next TPSpec. We denote such an update as $\Delta$TPSpec. For example, if the current TPSpec is \{('line_width': 'thin'), ('line_color': 'black')\} and the next TPSpec is \{('line_width': 'thin'), ('line_color': 'red')\}, the corresponding $\Delta$TPSpec is \{('line_color': 'red')\}. As discussed below, the output module can be a sequence decoder, in which the $\Delta$TPSpec is predicted as a sequence; or a set of classifiers, each of which predicts the new value of a different slot.

\textbf{M1} S2S-Plot+TXT. The first method is a seq2seq method whose input consists of the current state as represented by both TPSpec and plot image, and the dialog history. The TPSpec and dialog history are concatenated and fed to a seq2seq model. For all methods involving a seq2seq model, we use a 2-layer Bi-LSTM for the text encoder and another 2-layer Bi-LSTM for the decoder. To encode the plot image, we used a CNN followed by a row-wise LSTM. The final representation of an image is a sequence of vectors and are concatenated with the text representations on the temporal dimension before they are fed to the decoder. More details are provided in Appendix B.

\textbf{M2} S2S-TXT. The second method is another seq2seq model, but we omit the plot image from the input. We consider this version in order to assess the role of the vision modality in the task.

\textbf{M3} S2S-NoState. This is a seq2seq model whose input consists only of the dialog history. The state in the form of current TPSpec or plot image is completely omitted. The goal is to assess if the state is actually taken into account by the model.

\textbf{M4} S2S-NoUtterance. This is a seq2seq model whose input consists only of the current state as represented by TPSpec. The dialog history is completely omitted. The goal is to assess if the dialog history is actually taken into account by the model.

\textbf{M5} MaxEnt. We trained a logistic regression classifier to take as input the TPSpec and dialog history. They are represented jointly as bag-of-words. Classification predictions are made for each slot separately. For each slot, the candidate label space is all possible labels that appeared in our dataset, along with a special label \{unchanged\} indicating not to change the value of this slot, i.e. using the value from current state. Notice that bag-of-words features have a critical problem of ignoring word ordering. For example, it cannot distinguish between “red line with blue markers” and “blue line with red markers”.

\textbf{M6} RNN + MLP. This model is similar to MaxEnt except that features are extracted by an LSTM encoder, which considers word ordering. It differs from the seq2seq models in that the prediction is made with MLP classifier heads for each slot separately, instead of an LSTM decoder for the whole output. This exempts the model from the burden of generating a structured sequence; on the other hand the model is no longer equipped to learn the dependencies between different slots. We use a 2-layer bidirectional LSTM encoder for the input representation. Each MLP consists of 2 fully-connected layers.

\textbf{M7} Transformer + MLP. We consider another alternative where instead of an RNN, we use a transformer encoder, in particular, BERT (Devlin et al., 2019). The final layer output of the special BERT token “[CLS]” is used as the input representation and fed to MLP classifier heads. The structure of MLP classifier heads is the same as in RNN+MLP.

\textbf{7 Experiments}

We conducted experiments for the following purposes: (P1) to evaluate the performance of the above-mentioned methods; (P2) to establish the
Table 2: Exact match plotting performance.

<table>
<thead>
<tr>
<th>Methods</th>
<th>SPLIT</th>
<th>SINGLE</th>
<th>PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2S-PLOT+TXT</td>
<td>0.585</td>
<td>0.613</td>
<td>0.594</td>
</tr>
<tr>
<td>S2S-TXT</td>
<td>0.601</td>
<td>0.613</td>
<td>0.591</td>
</tr>
<tr>
<td>S2S-NoState</td>
<td>0.525</td>
<td>0.549</td>
<td>0.535</td>
</tr>
<tr>
<td>S2S-NoUtterance</td>
<td>0.060</td>
<td>0.047</td>
<td>0.046</td>
</tr>
<tr>
<td>MaxEnt</td>
<td>0.196</td>
<td>0.265</td>
<td>0.422</td>
</tr>
<tr>
<td>RNN+MLP</td>
<td>0.328</td>
<td>0.324</td>
<td>0.325</td>
</tr>
<tr>
<td>Transformer+MLP</td>
<td>0.311</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 3: Slot change F1 plotting performance.

<table>
<thead>
<tr>
<th>Methods</th>
<th>SPLIT</th>
<th>SINGLE</th>
<th>PAIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2S-PLOT+TXT</td>
<td>0.871</td>
<td>0.890</td>
<td>0.888</td>
</tr>
<tr>
<td>S2S-TXT</td>
<td>0.874</td>
<td>0.893</td>
<td>0.885</td>
</tr>
<tr>
<td>S2S-NoState</td>
<td>0.847</td>
<td>0.866</td>
<td>0.863</td>
</tr>
<tr>
<td>S2S-NoUtterance</td>
<td>0.316</td>
<td>0.306</td>
<td>0.155</td>
</tr>
<tr>
<td>MaxEnt</td>
<td>0.677</td>
<td>0.734</td>
<td>0.806</td>
</tr>
<tr>
<td>RNN+MLP</td>
<td>0.714</td>
<td>0.712</td>
<td>0.724</td>
</tr>
<tr>
<td>Transformer+MLP</td>
<td>0.723</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

7.2 Evaluation Metrics

We evaluate performance using two metrics: Exact Match (EM) and Slot change F1. Exact Match measures how accurate the models are at updating the plots exactly as expected. It is defined as the percentage of datapoints whose current TPSpec, when updated with the model predicted ∆TPSpec, can exactly match the gold target TPSpec. Slot change F1 measures accuracy on individual slots. Let $S_y$ be the set of slot-value pairs in the predicted ∆TPSpec and $S_g$ be the set of slot-value pairs in the gold ∆TPSpec, precision, $P = \frac{|S_p \cap S_g|}{|S_p|}$, recall $R = \frac{|S_p \cap S_g|}{|S_g|}$ and $F1 = \frac{2PR}{P+R}$.

7.3 Performance (P1)

We report Exact Match performance in Table 2, and Slot change F1 in Table 3. From the tables, it is clear that seq2seq based models generally perform better than classification methods. A possible reason is that, by modeling ∆TPSpec as a whole in the decoder, the models implicitly learned dependencies between different slots and thus improved the overall performance. Also, neural classification methods including RNN+MLP and Transformer+MLP displayed poor performance, not even beating MaxEnt with bag-of-words. Further, as an ablation study, the S2S-NoState and S2S-NoUtterance performed significantly worse than S2S-TXT, confirming that both the current state and the user utterance are necessary to seq2seq methods in performing this task.

Both S2S-TXT and S2S-PLOT+TXT perform the best at the SINGLE token granularity. On this granularity, there is no significant difference between their performance on exact match. For slot F1, S2S-TXT even performs significantly better than S2S-PLOT+TXT, with $p = 0.033$ in an unequal variance T-test, which implies that for seq2seq methods adding the image modality does not add much on top of the text modality in this task.

Table 4 shows performance of the best performing methods, S2S-PLOT+TXT and S2S-TXT, per plot type. We ran 5 experiments and reported the means and standard deviations in order to gain a better comparison between their performances. We can see that, as expected for our above results, for most plot types, performance of the two methods is similar.
### 7.4 Agreement Among Workers (P2)

In order to further inspect the quality and difficulty of our dataset, we sampled a subset of 444 partial dialogs. Each partial dialog consists of the first several turns of a dialog, and ends with a Describer utterance. The corresponding Operator response is omitted. Thus, the human has to predict what the Operator (the plotting agent) will plot, given this partial dialog. We created a new MTurk task, where we presented each partial dialog to 3 workers and collected their responses. We calculated the agreements between the newly collected responses and the original Operator response, results shown in Table 5.

The cases in which the majority of the workers (3/3 or 2/3) exactly match the original Operator, corresponding to the first two rows, happen 72.6% of the time. The cases when at least 3 out of all 4 humans (including the original Operator) agree, corresponding to row 1, 2 and 5, happen 80.6% of the time. This setting is also worth considering because the original Operator is another MTurk worker, who can also make mistakes. Both of these numbers show that a large fraction of the utterances in our dataset are intelligible implying an overall good quality dataset.

Fleiss’ Kappa among all 4 humans is 0.849; Cohen’s Kappa between the original Operator and the majority among 3 new workers is 0.889. These numbers indicate a strong agreement as well.

### 7.5 Models vs. Gold Human Performance (P3)

The gold human performance was obtained by having one of the authors perform the same task as described in the previous subsection, on a subset of 180 samples. The result is a 76.8% exact match. That is, our best model is 15.5 percentage points behind gold human performance, showing there is room for models to improve on this dataset.

### 7.6 Comparison to Performance on Image Editing

The best accuracy reported on the aforementioned conversational image editing dataset was 74% on intent classification, ignoring actual attribute values (Manuvinakurike et al., 2018a). This result is not directly comparable to the best accuracy 61.3% on our dataset due to the difference in accuracy definition. To our knowledge, no comparable results has been reported on the image editing dataset, and the dataset is not publicly available.

### 8 Error Analysis

We inspected the output of our best-performing models in order to identify the most common...
causes of errors. Here we used S2S-TXT with SINGLE granularity as a representative; the error categories are similar for S2S-PLOT+TXT or other granularity.

8.1 Ambiguity
Sometimes the Describer utterance is ambiguous and makes different actions all reasonable. We spotted two kinds of ambiguities: the unspecified new slot and the value, exemplified in Table 6a and 6b respectively. 1) Unspecified new slot. The Describer added a new component to the plot (the grid lines), which activated new slots (“grid_line_type”) whose values are unspecified. Therefore, any value for these slots should be correct. 2) Ambiguous value. The Describer asked to change the size of a component (the font), but did not specify the value. As in the example, the font size was “large”; to make it “smaller”, both “medium” and “small” are correct.

8.2 Human Errors
We report some of the errors that are due to mistakes made by MTurk workers. Operators can overlook part of the Describer’s instruction. These erroneous actions are recorded and in turn be counted as errors of models in our automatic evaluation process.

8.3 Model Errors
In addition to human errors, many cases were also due to the model itself. We show examples of model errors in Table 7. 1) Multi-turn dialog history. In most samples, the dialog history consists of only one utterance, the Describer’s instruction. As a result, when confronted with multiple utterances concatenated, the model may get confused. 2) Complex slot value. Some slot values are relatively hard to describe in natural language, such as “colormap” in example 7b. They can cause the models to make mistakes. 3) Infrequent expressions. When the user expresses their request in an unusual way (in example 7c, “log style” for log scale), the model may not understand since it is rarely seen in the training data.

9 Conclusions
In this paper, we defined the problem of conversational plotting agents, which is of great practical importance considering the large volume of questions online about plotting library usage. We also presented a dataset, CHARTDIALOGS, to facilitate the development of such agents. Our experiments have demonstrated the feasibility of seq2seq-based methods to produce working models for dataset; however, there is still a large gap between our best performing methods and human performance.

Future work includes methods that get closer to human performance on the dataset. A practical line of future work is embedding our plotting agent in interactive environments such as Jupyter Lab.

Acknowledgments
This work was partially supported by a Hellman Fellowship. We also appreciate constructive feedback from our anonymous reviewers.
References


Min-Yen Kan, Xiangnan Li, Weijiang Lei, Xisen Jin, Zhaochun Ren, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In ACL.


Appendix

A Plot Types and Slots

We show all plot types and slots related to each type in Table 8. All the plot types and slots are illustrated in Figure 6.

B Model Implementation Details

Model implementations are based on OpenNMT (Klein et al., 2017) and HuggingFace Transformers (Wolf et al., 2019).

B.1 S2S-TXT

LSTM hidden size is 128, batch size is 16. Model is trained for 100,000 steps. Learning rate is initialized to 1.0; starting from step 50,000, the learning rate is halved every 10,000 steps.

B.2 S2S-PLT+TXT

Text Encoder and Decoder have the same configuration as in S2S-TXT. The Plot (image) Encoder is a CNN with following layers: conv1 (64x3x3) - pooling1 (2x2) - conv2 (128x3x3) - pooling2 (2x2) - conv3 (128x3x3) - batch normalization3 - conv4 (256x3x3) - pooling5 (2x1) - conv5 (256x3x3) - batch normalization5 - pooling5 (1x2) - conv6 (256x3x3) - pooling6 (5x5). The output size from CNN is original image size reduced 40 times on both height and width. After CNN, a row-wise RNN is applied and the output for each row are concatenated to form the plot image encoding. The learning rate scheme is the same as in S2S-TXT.

B.3 RNN+MLP

LSTM hidden size is 64, batch size is 32. Each MLP head has 2 layers, mapping from 128 (LSTM cell and output concatenated) to 32 and from 32 to number of classes. Model is trained for 100,000 steps. Learning rate is initialized to 1.0; starting from step 50,000, the learning rate is halved every 10,000 steps.

B.4 Transformer+MLP

The version of pretrained BERT we used is bert-base-uncased. It is fine-tuned with our classification heads. Batch size is 8 and gradient is accumulated over every 4 steps. Each MLP head has only 1 layer, mapping from BERT hidden size (768) to number of classes. Learning rate is 2e-5. Model is trained for 30 epochs.

C Amazon Mechanical Turk HIT Screenshots

We show several screenshots of our HIT in Figure 7 and 8.

---

This model structure is adapted from OpenNMT.
<table>
<thead>
<tr>
<th>Plot Types</th>
<th>Slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Axes</td>
<td>Polarize, X-axis Scale, Y-axis Scale, X-axis Position, Y-axis Position, Invert X-axis, Invert Y-axis, Grid Line Type, Grid Line Style, Grid Line Width, Grid Line Color, Font Size</td>
</tr>
<tr>
<td>2. 3D Surface</td>
<td>Color map, Invert X-axis, Invert Y-axis, Invert Z-axis</td>
</tr>
<tr>
<td>3. Bar Chart</td>
<td>Bar Orientation, Bar Height, Bar Face Color, Bar Edge Width, Bar Edge Color, Show Error Bar, Error Bar Color, Error Bar Cap Size, Error Bar Cap Thickness, Data Series Name</td>
</tr>
<tr>
<td>4. Contour/Filled</td>
<td>Contour Plot Type, Number of levels, Color Map, Color Bar Orientation, Color Bar Length, Color Bar Thickness</td>
</tr>
<tr>
<td>5. Contour/Lined</td>
<td>Contour Plot Type, Lined Style, Line Width</td>
</tr>
<tr>
<td>6. Histogram</td>
<td>Number of Bins, Bar Relative Width, Bar Face Color, Bar Edge Width, Bar Edge Color, Data Series Name</td>
</tr>
<tr>
<td>7. Imshow</td>
<td>Color map, Invert X-axis, Invert Y-axis</td>
</tr>
<tr>
<td>8. Line Chart</td>
<td>Line Style, Line Width, Line Color, Marker Type, Marker Size, Marker Face Color, Marker Edge Color, Marker Interval, Data Series Name, Show Error Bar, Error Bar Color, Error Bar Cap Size, Error Bar Cap Thickness</td>
</tr>
<tr>
<td>9. Pie Chart</td>
<td>Exploding Effect, Precision Digits, Percentage tags’ distance from center, Label tag’s distance from center, Radius, Section Edge Width, Section Edge Color</td>
</tr>
<tr>
<td>10. Polar</td>
<td>Polarize, Grid Line Type, Grid Line Style, Grid Line Width, Grid Line Color, Font Size</td>
</tr>
<tr>
<td>11. Scatter</td>
<td>Polarize, Marker Type, Marker Size, Marker Face Color, Marker Edge Color, Marker Edge Color, Color Map, Color Bar Orientation, Color Bar Length Color Bar Thickness</td>
</tr>
</tbody>
</table>

Table 8: Plot types and slots in our dataset
Figure 6: Plot types and slots.
(a) Describer: starting the HIT

(b) Operator: starting the HIT

Figure 7: HIT screenshots.
(a) Describer: in progress

(b) Operator: in progress

Figure 8: HIT screenshots.