Structured Data Representation in Natural Language Interfaces

Yutong Shao, Arun Kumar, and Ndapa Nakashole
University of California, San Diego
{yshao,arunkk,nnakashole}@eng.ucsd.edu

Abstract

A Natural Language Interface (NLI) enables the use of human languages to interact with computer systems, including smart phones and robots. Compared to other types of interfaces, such as command line interfaces (CLIs) or graphical user interfaces (GUIs), NLIs stand to enable more people to have access to functionality behind databases or APIs as they only require knowledge of natural languages. Many NLI applications involve structured data for the domain (e.g., applications such as hotel booking, product search, and factual question answering.) Thus, to fully process user questions, in addition to natural language comprehension, understanding of structured data is also crucial for the model. In this paper, we study neural network methods for building Natural Language Interfaces (NLIs) with a focus on learning structure data representations that can generalize to novel data sources and schemata not seen at training time. Specifically, we review two tasks related to natural language interfaces: i) semantic parsing where we focus on text-to-SQL for database access, and ii) task-oriented dialog systems for API access. We survey representative methods for text-to-SQL and task-oriented dialog tasks, focusing on representing and incorporating structured data. Lastly, we present two of our original studies on structured data representation methods for NLIs to enable access to i) databases, and ii) visualization APIs.

1 Introduction

Natural Language Interfaces (NLIs) seek to enable user-system interactions using natural language. Compared to other types of interfaces, such as command-line or graphic interfaces, NLIs have a much lower learning curve. They can be used by lay users with little to no extra training, thus attractive in practice.

NLIs have been an active research for decades [1, 2]. Early methods had limited success, possibly due to the absence of powerful and effective models for language understanding. Recently, with the rise of deep learning models, especially Transformer-based models in NLP [46], the overall performance on a variety of language understanding tasks, including NLI-related tasks, has seen a significant boost. Nonetheless, the state-of-the-art (SOTA) NLI models are still far from perfect, and are do not yet reach the level of being fully deployable in real products. In this work, we investigate the progress and challenges of NLI-related tasks. Specifically, we focus on two types of tasks: *semantic parsing* and *task-oriented dialogs (TOD)*. In addition, we focus on the understanding and representation of *structured data*, such as knowledge bases (KBs), databases (DBs) or tables, in NLI-related tasks. In practice, NLI applications inevitably need to incorporate and understand the relevant

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

"backend" structured data for the domain (hotel booking, product searching, factual question answering, etc.) Thus, besides language comprehension, a correct understanding of structured data is also crucial for the model to make the correction decisions.

In following sections, we first provide a more detailed descriptions of the NLI-related tasks we study. For each task, we survey representative methods for the tasks, especially regarding structured data representation; then we introduce our own work related to the task. A road map of this paper is illustrated in Figure 1.

Figure 1: The road map of the paper. We discuss several NLI-related tasks, relevant structured data and corresponding representation methods. We introduce two of our own related work, Speech-to-SQL and Plotting Agent, under the topic.

2 Preliminaries

2.1 Popular NLP models

We briefly introduce several model architectures that are widely used today in the NLP community and will be frequently mentioned in this paper.

Sequence-to-sequence (S2S) A sequence-to-sequence (S2S) model aims to map an input sequence to an output sequence [41]. In NLP, the input and output sequences are usually natural language sentences. Many NLP tasks can be modeled by S2S models, such as machine translation, summarization, and dialog.

S2S models are mostly based on an *encoder-decoder* model architecture, in which the *encoder* aims to obtain an intermediate vector or tensor representation that includes necessary information from the input sequence (named the *encoding* of input), and the *decoder* generates the output sentence based on the input encoding. Both encoder and decoder can be modeled by recurrent neural networks (RNNs), such as LSTM or GRU [42, 43]; or by transformer models, which we briefly introduce below.

In the basic version of RNN-based S2S model architecture,the encoder absorbs an input sentence and encodes it into a single *encoding vector*; the decoder takes the encoding vector to generate an output sentence as a conditioned language model (i.e. generate tokens in auto-regressive manner, based on previous tokens). Intuitively, the encoding vector is an information bottleneck in the architecture because all information from the input has to be incorporated into a single vector.

Attention The attention mechanism [44, 45] was proposed to overcome the above-mentioned bottleneck. It allows the decoder states to "look through" the encoding vector and directly interact with encoder states. In

detail, given a decoder state vector d and encoder state vectors $e_{1:N}$, we compute an *attention context vector* c as follow:

$$\alpha_k = \frac{\texttt{sim}(d, e_k)}{\sum_{i=1}^{N} \texttt{sim}(d, e_i)}; c = \sum_{k=1}^{N} \alpha_k e_k$$

where α_k are called *attention weights*, and $\texttt{sim}$ is a function measuring vector similarities, such as dot product ($a^T b$), bilinear product ($a^T W b$), linear combination ($w_1^T a + w_2^T b$), etc. The attention context vector is then added or concatenated with the decoder state vector before predicting the output token.

Self-attention and Transformers The Transformer architecture, proposed in [46], fully replaces RNNs with *self-attention* layers. Self-attention layers encode a sequence by applying attention from the representation of each token to all other tokens in the sequence, and use the attention context vector (with residual connection, i.e. summed with the pre-attention representation vector) as the output token representation. Transformers are shown to be particularly useful for pretrained language models (PLMs), i.e. first *pretrain* the model on an extremely large corpus with self-supervision, and then *fine-tune* the pretrained parameters on a downstream task [47, 48].

2.2 NLI-related Tasks

The topic of NLIs is broad. In this work, we focus on two representative tasks: semantic parsing and task-oriented dialogs (TOD). These tasks are useful in practice and widely studied, especially in the current era of deep learning. We briefly introduce the tasks in this section and take in-depth investigations in the following sections.

Semantic Parsing Semantic parsing aims to map a natural language sentence into a structured, executable logical form to represent the semantics of the sentence. It can be seen as a straightforward formulation of an NLI: parsing the natural language query into system-understandable form. There are different formats of such executable logical forms, such as GeoQuery [3], lambda-DCS [4], SparQL (for knowledge graphs (KG)) and SQL (for relational databases (DB)).

Task-oriented Dialog (TOD) Task-oriented dialog (TOD) systems interact with users to complete certain tasks. The task objectives are usually in the form of making API calls to obtain information (e.g. look for restaurants) or take actions (e.g. make a restaurant reservation) [23, 24, 25, 26]. From a practical perspective, for complex tasks, it is hard for user requests to be described and completed in a single utterance. Therefore, interactions are necessary between the user and system to complete the task, which makes a task-oriented dialog.

3 Semantic Parsing

3.1 Introduction

Early work in semantic parsing are based on grammar rules induction and feature-based rule selection [3, 5]. Since the rise of deep learning, parsers are mostly based on neural models with encoder-decoder architectures that generate the output logical form directly from input text [6, 7, 8]. Moreover, recent work pay increasingly more attention on the *generalization ability* of trained models, especially on generalizing over changes in the backend structured data. The advantages of such generalization is to alleviate the burden of re-collecting in-domain data and re-training the model whenever we update our DB or KG. In order to generalize to unseen structured data, we have to incorporate the structured data as part of the input to the model, instead of fully include them into the model weights. It thus raises the problem of *structured data representation*, which is basically on how to effectively incorporate and leverage the structured data for a better model performance.

In what follows, we focus on structured data representation on semantic parsing with SQL as the logical form, which is also called text-to-SQL. As mentioned, the task of text-to-SQL takes as input the user utterance in natural language and the backend DB. A recent benchmark on text-to-SQL, Spider [9], explicitly tests the generalization ability to unseen DBs by separating the set of DBs used in train/dev/test splits of the benchmark. It provides a good testbed for modern text-to-SQL methods and has been widely used by previous work in this direction.

3.2 Structured Data Representation in Text-to-SQL

3.2.1 Basic Method: Linearization

In the task of text-to-SQL, the structured data to be incorporated are the DBs. The most straightforward idea is to *linearize* the DB, i.e. transform it into a token sequence, and concatenate it with the user question to form the input text. A simple example is shown in Figure 2a. By linearizing, the structured data is mapped into unstructured text modality, thus can fit into regular text-processing models.

Despite of its simplicity, linearization has shown to be a very useful way to represent DBs [12, 13]. There are also work focusing on injecting more useful information into such linear representations. For example, in [14] the authors propose to find *anchor text* which are input text tokens matching DB cell values, and add them to the linearized DB, next to the column of the cell (similar to the format of cell inclusion shown in Figure 2a). Also, DB linearization works well with large-scale pretraining [15, 16]. This is possibly due to the compatibility of linearized DB and unstructured text, thus effective pretraining objectives (such as masked language-modeling (MLM)) can be adopted.

(a) DB linearization. (b) DB graph.

Figure 2: DB linearization and graph illustration. (a) An example of DB linearization. Notice that the included cell values are deliberately selected to showcase the idea; in practice, cells are not always available, depending on the dataset, and cell value selection is non-trivial and method-specific. (b) An example of DB graph. Some columns, tables and cells are omitted for visualization clarity. In detail, the design of relation set is also method-specific. This figure only illustrates the high-level idea.

3.2.2 Table Representation

Essentially, a DB is made up by tables; therefore, we can exploit the tabular structure to better represent the information. A lot of exisiting work target table representation, not specific for text-to-SQL but generally for any task involving tables (other tasks include table QA, table-text entailment, etc.) [10, 11]. Some approaches leverage

properties of tables together with linearization. For example, [16] first applies a *horizontal transformer* to get text-row representations for rows in the table, then a *vertical transformer* between all text-row representations to obtain a single text-table representation.

3.2.3 Graph Representation

There is another line of work to represent the table based on graph structure. We can define relations between question tokens and DB entities (e.g. table names, column names, cell values). Example relations include *table-column affiliation* between tables and columns; *primary-foreign key* between columns; *token-table/column mention* between question tokens and tables or columns; etc. Such relations form a graph where each token / DB entity is a node, and each existing relation is an edge. An example of such graph is shown in Figure 2b. Given such a graph, we can use graph neural network to encode it [17, 18]. We can also use the relations directly to enhance linearization-based methods. For example, we can add a *relational bias* term in self-attention layers in transformers. Each relation has a learnable bias vector; when computing the attention weights between two tokens, if they exhibit a relation, we add the corresponding bias vector into the token representation [19, 20] (similar ideas are also studied in table-only representation, such as in [11]).

3.3 Structured Data Representation in Speech-to-SQL

We introduce our research on speech-to-SQL[1]. Most existing studies focus on text-to-SQL which is to parse natural language *text* utterances into executable SQL queries. Motivated by the rise of speech-driven digital assistants on smartphones, tablets, and other small handheld devices, we study the task of parsing spoken natural language (in audio) to executable SQL queries (speech-to-SQL parsing). A speech-to-SQL parser has a number of potential use scenarios for quick and convenient data look-up, such as patient caring (in healthcare domain), business meeting, or driving. In this work, we study ways to improve speech-to-SQL parsers. We will also highlight the representation of structured data, i.e. the DB, in this study.

3.3.1 Baselines

An obvious solution of speech-to-SQL parsing is to first pass the speech audio through an automatic speech recognizer (ASR), and then issue the top-ranked ASR transcription to a text-to-SQL parser which produces the final SQL. We name it the *blackbox* baseline. A drawback of this baseline is that no attempt is made to deal with ASR errors. Figure 3(A) shows an example of such errors. Passing ASR errors to the text-to-SQL parser is unlikely to produce the correct SQL.

For ASR error correction, we import two more baselines from previous work: *re-ranker* and *S2S-rewriter* baseline. The re-ranker takes the top-K candidate transcriptions from ASR and re-rank them to select the best transcription [21]. In contrast, the S2S-rewriter directly rewrites an ASR transcription into a better sentence, as a S2S task [22]. Both methods do not incorporate the DB information.

3.3.2 Method Overview

We propose a two-stage ASR error correction method. First, we *tag* the input sentence (an ASR transcription) to identify tokens that are incorrect and should be edited. Second, we edit the sentence using a *blank-filling* model, where the tagged incorrect parts of the sentence are regarded as blanks and filled by predicted tokens. Moreover, we incorporate the backend DB information into the model by adding the DB schema and relevant cell values into the input sentence, in the above-mentioned *linearized* manner. Figure 3 illustrates the framework of our method.

[1] Under submission. We focus on the high-level ideas and omit the details in order not to violate submission policies.

Figure 3: Overview of the ASR correction method for speech-to-SQL. (A) Directly passing the erroneous ASR transcription to text-to-SQL parser will likely produce wrong SQL output. We apply a Tagger (B) and an ILM (blank-filling) rewriter (C) to fix the transcription, incorporating audio features and DB information.

3.3.3 Experiments and Analysis

Datasets Experiments are conducted on the Spider dataset mentioned above. To make it usable for evaluation speech-to-SQL methods, we used Amazon Polly, a text-to-speech (TTS) service, to automatically transform the user queries from text to audio. We used Amazon Transcribe as the raw ASR transcriber. Besides, since our method focuses on ASR correction, we utilize existing models as the backend text-to-SQL parser during evaluation. We use RAT-SQL [19] and T5 (T5-base and T5-large) [13], which are representative and competitive text-to-SQL parsers.

Evaluation Metrics We test baseline methods and our proposed methods on two categories of metrics. The first category is on *text accuracy*, for which we use *word error rate (WER)* and *BLEU score*. The second category measures the final *SQL accuracy*, for which we use *exact match rate* and *execution match rate* of the SQL predictions against ground truth.

Results We test and compare our proposed methods to all baseline methods (blackbox, re-ranker, S2S-rewriter) on Spoken Spider. Results show that our proposed method can outperform all baseline methods on both category of metrics, regardless of backend parser. We also conducted ablation studies to profile the source of performance gain. The results show that the tagging + blank-filling pipeline, compared to re-ranker or S2S-rewriter, is the major source of improvement. Nonetheless, the incorporation of structured data, i.e. DB information, also provides non-negligible benefits. We would also like to highlight that, retrieving and adding relevant cells into the input can substantially improve the performance, compared to only using the DB schema.

4 Task-oriented Dialog Systems

4.1 Real Application Example

As mentioned, a TOD system can help users complete actions more easily, especially in complex scenarios where multi-turn interactions are needed. One of the real use cases is on *data plotting*. Plotting is a very common practice for visualizing data and mathematical functions. Existing plotting libraries such as `matplotlib` support a decent range of functionalities. However, using such libraries can be difficult for novice users and time consuming even for experts, due to both the hardness of programming and complexity of our detailed plotting needs. It thus motivates us to build a TOD system that helps human complete plotting tasks by interacting with users through natural language.

In what follows, we first provide a more general and formal introduction to the TOD systems. We then survey related work in this direction, and introduce our own work on the above-mentioned plotting task.

4.2 Formal Introduction

A TOD system runs in one or several *domains* and requires a domain-specific *ontology*. Intuitively, an ontology is a set of slots and values for each slot, representing the domain-specific information required by the system or the user. Several example ontologies are shown in Figure 4.

(a) The ontology of MultiWOZ [26]. For each slot, the upperscript indicates the domain indexes it belongs to. *: universal, 1: restaurant, 2: hotel, 3: attraction, 4: taxi, 5: train, 6: hospital, 7: police.

(b) The ontology of DSTC2 [25]. The domain is restaurant booking.

Figure 4: Sample ontology of existing datasets.

There are two main categories of TOD system design: pipeline-based and end-to-end [27]. A pipeline-based TOD system consists of a collection of separate modules, including natural language understanding (NLU), dialog state tracking (DST), dialog policy, and natural language generation (NLG). In contrast, an end-to-end TOD system directly takes the current user utterance and dialog history as input, and predicts the action or response.

Structured data in TOD systems is a complex topic, because a TOD system involves several different types of structured data, mainly including two categories: *internal* and *external* structured data. External structured data include the backend DB or KG, similar to the structured data for semantic parsing as mentioned above. Internal structured data include *dialog state* and *dialog act*. Dialog state is the output of the dialog state tracking component. It includes the user intents and specified values for each slot. It can be transformed into a DB query to search for relevant information, which may also be used by the model to decide the response. Dialog act is the output of dialog policy component. It controls what the system response wants to achieve. Basically, for system response generation we care about "what to say" and "how to say it"; the dialog act is the "what to say" part.

4.3 Structured Data Representation in Task-oriented Dialog Systems

4.3.1 External

Database (DB) In TOD systems, we utilize the backend DB by queries based on the dialog state (the user-specified slot values are conditions in the query). How the DB query results are utilized depends on the task setting, or practically, the dataset. On certain datasets, the systems are required to use the full content. For example, in the bAbI dialog dataset [23], given all the records satisfying user-specified conditions, the system is expected to "learn" to rank results by the field named "rating". In such scenarios, the system usually regard the records as sentences in the dialog history. On the other hand, many datasets do not have the explicit requirement [25, 26, 28]. As a result, a widely adopted heuristic is to simply use the *number of records* in the query results, instead of the full results content [28, 29, 30]. The input and output of the model may be *delexicalized*, i.e. using placeholders for entities (e.g. `[v.HOTEL_NAME]` for a hotel name), which are replaced by actual entities in the query results during post-processing. In this way, in the dialog model, the query results are no longer structured data, but a single token (representing the number) that can be simply represented using look-up embeddings.

Knowledge Graph (KG) Knowledge graphs (KG) are frequently used in the task of *knowledge-grounded dialogs*, in which the user chats with the dialog system to ask questions, learn knowledge or gather information, and the system responses based on external knowledge. This task does not rigorously belong to the definition of TOD, since the system does not make explicit backend API calls. Nonetheless, this task is highly related to TOD in the sense of retrieving relevant information from external data. One way to represent KG is to simply use the (subj, rel, obj) triplet form. For example, in methods based on memory networks [31, 32], each triplet in the KG are vector-embedded and added to the memory bank. During response generation, the token prediction probability includes a term to copy tokens from the KG triplets. Another line of work leverages the graph-structure of KG by "walking" on the edges, simulating human reasoning. These methods can use the encoding vector of current turn information as the initial state. They may iteratively update the state using a gated recurrent cell, each step attending to all walkable nodes [36], or directly predict a sequence of relations to decide the walking path [37, 38].

4.3.2 Internal

As mentioned, internal structured data in TOD systems mainly include dialog state and dialog act. Some fully end-to-end models do not have explicit modeling for internal structured data [31, 32]. However, it decreases the transparency, interpretability and controllability of the model. Thus, despite of the competitive performance of such models, it is still beneficial to explicitly predict and incorporate the internal structured data. Here we put more attention on incorporation and less on prediction. On prediction side, dialog state tracking and dialog policy learning are both popular research topics and involve a large variety of task-specific methods, which are out of the scope here.

Linearization A straightforward yet useful way to represent internal structured data is *linearization*, similar to above-mentioned DB linearization in semantic parsing. We represent the dialog state and/or dialog act as token sequences, and encode them as text. This type of method can be used for both prediction and incorporation, creating a unified pipeline for end-to-end dialog model where the "internal states" are always text. For example, in [33], the dialog model is formulated as a two-stage sequence-to-sequence (S2S) model. First, based on previous dialog state and dialog history, predict current dialog state (this step is essentially dialog state tracking); second, based on dialog history and current dialog state, generate the response. [34] considers TOD as a language modeling (LM) task by concatenating dialog history, dialog state, dialog act and response all together as a token

sequence. They fine-tune a pretrained LM, GPT2, to predict dialog state, dialog act and response based on the input dialog history.

Other Specific Methods There are methods exploiting the determined structures of such internal structured data in order to effectively model them. For example, [28] assumes a fixed domain and ontology, thus dialog state can be treated simply as a probability distribution per slot. [35] exploits the (intent, slot, value) triplet structure of dialog act to build 3-layer tree to predict and control the encoder of dialog act. Such methods can potentially have simpler design or have better performances on specific datasets; however, it is harder for them to generalize to other datasets or domains.

4.4 Structured Data Representation in a Plotting Agent

We introduce our original work on the novel task of *plotting agent*, by which we refer to the above-mentioned TOD system for data plotting [39]. Notice that this plotting agent is targeted at manipulating the plot appearance (colors, shapes, sizes, etc.) instead of the underlying data. In this work, we collected a large-scale dataset for training a plotting agent, and conducted experiments on competitive methods to compare and analyse their performances. We also showcase the influence of structured data representations on model performances.

4.4.1 Problem Definition

We aim to develop a *conversational plotting agent* that takes natural language instructions and updates the plot accordingly. The agent is designed conversational because plots can be complex, making it difficult to describe everything at once; thus, users may want to tune the appearance of their plot through multiple turns. Technically, a conversational plotting agent is framed as a TOD system. It has only one domain, which is plot control. We manually defined the domain ontology to include several plot types and a large number of slots, based on `matplotlib` documentation and the common needs based on our experiences. Figure 5 illustrates example slots for some of the plot types. The full ontology is shown in Table 1. Different plot types have different sets of slots, yet 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. For modeling purposes, the plot type can also be seen as a slot in the ontology. Notice that, in the current definition, a conversational plotting agent is simpler than a "full" TOD system, because the system directly uses dialog states as responses and does not have to generate text responses[2].

4.4.2 Dataset Overview

To enable training neural models for the new task, we collected a dataset, *ChartDialogs*, which consists of actual human dialogs on completing plotting tasks. We used Amazon Mechanical Turk for dataset collection. For each dialog, we engage two workers simultaneously to play the *user* and *system* respectively, and randomly generate a *target plot*. The target plot is only visible to the user worker. It stands for the plot that an actual user wants in real world scenarios. During the dialog, the user worker has to describe the plot to the system worker to accurately reproduce the exact plot to complete the task.

4.4.3 Methods

We assessed the performance of various methods for training TOD system on our ChartDialogs dataset. We experimented with two categories of methods: S2S-based and classification-based. For S2S-based methods,

[2]We wrote a simple script that can take as input the dialog state (plot type and other slot values), to generate the actual plot image using `matplotlib` and display it to the user. Thus, plot controlling is equivalent to dialog state tracking.

(a) Line Chart (b) 3D surface

Figure 5: Illustration of two of ChartDialog plot types. **(a) Line Chart** has slots such as *Line Style, Marker Interval*. **(b)** A **3D Surface** has slots such as *Surface Color*.

we linearize the dialog state in similar manner with previous work [33]. The model takes as input the user utterance and linearized dialog state, and predicts a linearized *dialog state update*, which includes the slot values that should be updated. We use LSTM as both encoder and decoder architecture for the S2S model. For classification-based methods, we exploit the specific design of our domain ontology that the possible values of each slot come from a finite value pool. Thus, we can treat dialog state tracking as a classification problem for each slot separately. The input format is the same as above, i.e. the concatenation of user utterance and linearized dialog state. We experimented using bag-of-word, LSTM and BERT [47] as input encoding methods, and train a logistic-regression or multi-layer perceptron (MLP) classifier heads[3] for each slot on top of the encoded input representation. In detail, we try three different classification methods: (i) **MaxEnt**, using bag-of-word embedding and logistic regression classifier; (ii) **LSTM+MLP**, using LSTM as input encoder and MLP classifier heads; and (iii) **BERT+MLP**, using BERT (parameters frozen) as input encoder and MLP classifier heads.

As a relevant detail, we experimented using different *granularity* for dialog state linearization, SPLIT, SINGLE and PAIR, detailed in Table 2. Conceptually, SPLIT best leverages the semantic overlap between different slots and values, while PAIR is the most succinct with regard to sequence length. In the result section, we shed light on how this design choice influences the model performances.

4.4.4 Experiments

Evaluation Metrics We evaluate the model performance on a turn-based manner, i.e. use each dialog turn as a data point and compare the model prediction with ground truth (human worker choices). For evaluation metrics, we use *exact match rate* which is the proportion of dialog state predictions fully matching the ground truth; and *Slot-F1* which measures the performance on slot-level[4].

[3] An MLP consists of several fully-connected layers with non-linear activation functions, finally applying a softmax layer to predict the probabilities of each class.

[4] We use F1-score instead of accuracy because in most cases, most slots are inactive and have the *[None]* value. That causes uneven distributions among slot values, thus we use F1-score.

	Plot Types	Slots
1.	Axes	Polarize, X-axis Scale, Y-axis Scale, X-axis Position, Y-axis Position, Invert X-axis, Invert Y-ax Grid Line Type, Grid Line Style,Grid Line Width, Grid Line Color, Font Size
2.	3D Surface	Color map, Invert X-axis, Invert Y-Axis, Invert Z-Axis
3.	Bar Chart	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
4.	Contour/Filled	Contour Plot Type, Number of levels, Color Map, Color Bar Orientation, Color Bar Length, Color Bar Thickness
5.	Contour/Lined	Contour Plot Type, Lined Style, Line Width
6.	Histogram	Number of Bins, Bar Relative Width, Bar Face Color, Bar Edge Width, Bar Edge Color, Data Series Name
7.	Matrix	Color Map, Invert X-axis, Invert Y-axis
8.	Line Chart	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
9.	Pie Chart	Exploding Effect, Precision Digits, Percentage tags' distance from center, Label tag's distance from center, Radius, Section Edge Width, Section Edge Color
10.	Polar	Polarize, Grid Line Type, Grid Line Style, Grid Line Width, Grid Line Color, Font Size
11.	Scatter	Polarize, Marker Type, Marker Size, Marker Face Color, Marker Edge Width, Marker Edge Co Color Map, Color Bar Orientation, Color Bar Length Color Bar Thickness
12.	Streamline	Density, Line Width, Line Color, Color Map, Arrow Size, Arrow Style

Table 1: Plot types and slots in our dataset

Granularity	Description	Example
(Dialog state)	The dialog state to linearize	`(plot_type = line chart, line_color`
PAIR	Combined slot-value pair as a token	"plot_type:line_chart line_color:blue marker_type:c
SINGLE	Each slot or value as a single token	"plot_type line_chart line_color blue marker_type
SPLIT	Split slot or values into natural language tokens	"plot type : line chart — line color : blue — marker

Table 2: Explanation of dialog state linearization granularity.

Results The main results are shown in Table 3. The S2S method largely outperforms classification-based methods, despite the fact that classification-based methods exploit the domain-specific design. Our hypothesis is that, the sequence predictor (decoder) in S2S model learns implicit inter-dependencies between slots (e.g. certain slots are only active for certain plot types), while the separated classifiers do not. Comparing classification methods, the simplest method, MaxEnt (with PAIR granularity) is competitive and outperforms baseline neural models. The unsatisfactory performance is possibly because their non-robustness or overfitting to noises in the dataset.

Based on the best-performing S2S method, we also conducted ablation study to verify the usefulness of user utterance and dialog state in the input. User utterances are useful as expected, as without user utterances the prediction would be an educated guess. The dialog states are also useful, which is a positive sign for the importance of (internal) structured data in TOD. Intuitively, the benefits of dialog state could be to provide a better semantic context, i.e. more relevant information, for the model. Comparing the linearization granularity for the S2S model, the SINGLE setting performs the best. It possibly implies that, among the three granularity settings, SINGLE achieves a proper trade-off between capturing slot / value semantics and controlling the sentence length.

[5]Due to the word-piece tokenization used in BERT, the SINGLE and PAIR linearizations are also tokenized to granularity level of SPLIT, therefore we do not report their performance.

Methods	Exact Match			Slot F1		
	SPLIT	SINGLE	PAIR	SPLIT	SINGLE	PAIR
S2S	0.601	**0.613**	0.591	0.874	**0.893**	0.885
S2S-NoState	0.525	0.549	0.535	0.847	0.866	0.863
S2S-NoUtterance	0.060	0.047	0.046	0.316	0.306	0.155
MaxEnt	0.196	0.265	***0.422***	0.677	0.734	***0.806***
LSTM+MLP	0.328	0.324	0.325	0.714	0.712	0.724
BERT+MLP	0.311	n/a[5]	n/a[5]	0.723	n/a[5]	n/a[5]

Table 3: Exact match plotting performance.

5 Conclusion

In this paper, we reviewed relevant tasks of natural language interfaces, semantic parsing (focusing on text-to-SQL) and task-oriented dialog (TOD) systems. We survey representative methods on these tasks, especially focusing on the perspective of representing and incorporating structured data, both external and internal. In general, linearization methods are widely adopted. They are applicable for almost all types of structured data while exhibiting competitive performances. Nonetheless, for different types of structured data, there are still opportunities for further performance gains by designing methods that capitalize on their distinct properties. We also present our original studies on the relevant tasks, and discuss our findings regarding the topic of structured data representation.

For future work on NLIs and structured data representation, many opportunities and challenges are still ripe for exploration. Despite the increasing performance on benchmarks, neural-based NLIs are still not widely deployed in the real world. Identifying and analyzing the challenges that remain when porting to new domains, tasks, datasets or real-world scenarios is one line of future work. Another future direction is a systematic study to find similarities among a subset of different tasks, or even all NLI-related tasks, to develop methods that are generalizable to new tasks with similar properties. Furthermore, obtaining data for NLIs in new domains, even a small amount of data for few-shot or fine-tuning remains a time-consuming endeavor that requires complex crowd-sourcing pipelines, and can be expensive. Coming up with ways to quickly obtain new data for training NLIs in new domains, and tasks is another direction for future work.

References

[1] Winograd, Terry. "Procedures As A Representation For Data In A Computer Program For Understanding Natural Language." (1971).

[2] Karamcheti, Siddharth, Dorsa Sadigh and Percy Liang. "Learning Adaptive Language Interfaces through Decomposition." ArXiv abs/2010.05190 (2020): n. pag.

[3] Zelle, John M. and Raymond J. Mooney. "Learning to Parse Database Queries Using Inductive Logic Programming." AAAI/IAAI, Vol. 2 (1996).

[4] Pasupat, Panupong and Percy Liang. "Compositional Semantic Parsing on Semi-Structured Tables." ArXiv abs/1508.00305 (2015): n. pag.

[5] Artzi, Yoav and Luke Zettlemoyer. "UW SPF: The University of Washington Semantic Parsing Framework." ArXiv abs/1311.3011 (2013): n. pag.

[6] Dong, Li and Mirella Lapata. "Language to Logical Form with Neural Attention." ArXiv abs/1601.01280 (2016): n. pag.

[7] Dong, Li and Mirella Lapata. "Coarse-to-Fine Decoding for Neural Semantic Parsing." ACL (2018).

[8] Cheng, Jianpeng, Siva Reddy, Vijay A. Saraswat and Mirella Lapata. “Learning an Executable Neural Semantic Parser.” Computational Linguistics 45 (2019): 59-94.

[9] Yu, Tao, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z Li, Qingning Yao, Shanelle Roman, Zilin Zhang and Dragomir R. Radev. “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task.” EMNLP (2018).

[10] Herzig, Jonathan, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno and Julian Martin Eisenschlos. “TaPas: Weakly Supervised Table Parsing via Pre-training.” ArXiv abs/2004.02349 (2020): n. pag.

[11] Yang, Jingfeng, Aditya Gupta, Shyam Upadhyay, Luheng He, Rahul Goel and Shachi Paul. “TableFormer: Robust Transformer Modeling for Table-Text Encoding.” ArXiv abs/2203.00274 (2022): n. pag.

[12] Scholak, Torsten, Nathan Schucher and Dzmitry Bahdanau. “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models.” ArXiv abs/2109.05093 (2021): n. pag.

[13] Xie, Tianbao, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer and Tao Yu. “UnifiedSKG: Unifying and Multi-Tasking Structured Knowledge Grounding with Text-to-Text Language Models.” ArXiv abs/2201.05966 (2022): n. pag.

[14] Lin, Xi Victoria, Richard Socher and Caiming Xiong. “Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing.” EMNLP Findings (2020).

[15] Yu, Tao, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher and Caiming Xiong. “GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing.” ArXiv abs/2009.13845 (2021): n. pag.

[16] Yin, Pengcheng, Graham Neubig, Wen-tau Yih and Sebastian Riedel. “TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data.” ArXiv abs/2005.08314 (2020): n. pag.

[17] Bogin, Ben, Matt Gardner and Jonathan Berant. “Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing.” ArXiv abs/1905.06241 (2019): n. pag.

[18] Cao, Ruisheng, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu and Kai Yu. “LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations.” ACL (2021).

[19] Wang, Bailin, Richard Shin, Xiaodong Liu, Oleksandr Polozov and Matthew Richardson. “RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers.” ACL (2020).

[20] Hui, Binyuan, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun and Yongbin Li. “S2SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers.” ArXiv abs/2203.06958 (2022): n. pag.

[21] Weng, Yue, Sai Sumanth Miryala, Chandra Khatri, Runze Wang, Huaixiu Zheng, Piero Molino, Mahdi Namazifar, Alexandros Papangelis, Hugh Williams, Franziska Bell and Gokhan Tur. “Joint Contextual Modeling for ASR Correction and Language Understanding.” ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2020): 6349-6353.

[22] Mani, Anirudh, Shruti Palaskar, Nimshi Venkat Meripo, Sandeep Konam and Florian Metze. “ASR Error Correction and Domain Adaptation Using Machine Translation.” ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2020): 6344-6348.

[23] Bordes, Antoine and Jason Weston. “Learning End-to-End Goal-Oriented Dialog.” ArXiv abs/1605.07683 (2017): n. pag.

[24] Wei, Wei, Quoc V. Le, Andrew M. Dai and Jia Li. “AirDialogue: An Environment for Goal-Oriented Dialogue Research.” EMNLP (2018).

[25] Henderson, Matthew, Blaise Thomson and J. Williams. “The Second Dialog State Tracking Challenge.” SIGDIAL Conference (2014).

[26] Budzianowski, Paweł, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan and Milica Gasic. "MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling." EMNLP (2018).

[27] Zhang, Zheng, Ryuichi Takanobu, Minlie Huang and Xiaoyan Zhu. "Recent Advances and Challenges in Task-oriented Dialog System." ArXiv abs/2003.07490 (2020): n. pag.

[28] Rojas-Barahona, Lina Maria, Milica Gašić, Nikola Mrksic, Pei-hao Su, Stefan Ultes, Tsung-Hsien Wen, Steve J. Young and David Vandyke. "A Network-based End-to-End Trainable Task-oriented Dialogue System." EACL (2017).

[29] Liu, Bing, Gökhan Tür, Dilek Z. Hakkani-Tür, Pararth Shah and Larry Heck. "Dialogue Learning with Human Teaching and Feedback in End-to-End Trainable Task-Oriented Dialogue Systems." NAACL (2018).

[30] Zhang, Yichi, Zhijian Ou, Huixin Wang and Junlan Feng. "A Probabilistic End-To-End Task-Oriented Dialog Model with Latent Belief States towards Semi-Supervised Learning." ArXiv abs/2009.08115 (2020): n. pag.

[31] Madotto, Andrea, Chien-Sheng Wu and Pascale Fung. "Mem2Seq: Effectively Incorporating Knowledge Bases into End-to-End Task-Oriented Dialog Systems." ACL (2018).

[32] Chen, Xiuyi, Jiaming Xu and Bo Xu. "A Working Memory Model for Task-oriented Dialog Response Generation." ACL (2019).

[33] Lei, Wenqiang, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He and Dawei Yin. "Sequicity: Simplifying Task-oriented Dialogue Systems with Single Sequence-to-Sequence Architectures." ACL (2018).

[34] Ham, Dong-hyun, Jeong-Gwan Lee, Youngsoo Jang and Kyungmin Kim. "End-to-End Neural Pipeline for Goal-Oriented Dialogue Systems using GPT-2." ACL (2020).

[35] Chen, Wenhu, Jianshu Chen, Pengda Qin, Xifeng Yan and William Yang Wang. "Semantically Conditioned Dialog Response Generation via Hierarchical Disentangled Self-Attention." ACL (2019).

[36] Moon, Seungwhan, Pararth Shah, Anuj Kumar and Rajen Subba. "OpenDialKG: Explainable Conversational Reasoning with Attention-based Walks over Knowledge Graphs." ACL (2019).

[37] Cohen, William W., Haitian Sun, R. Alex Hofer and Matthew A. Siegler. "Scalable Neural Methods for Reasoning With a Symbolic Knowledge Base." ArXiv abs/2002.06115 (2020): n. pag.

[38] Tuan, Yi-Lin, Sajjad Beygi, Maryam Fazel-Zarandi, Qiaozi Gao, Alessandra Cervone and William Yang Wang. "Towards Large-Scale Interpretable Knowledge Graph Reasoning for Dialogue Systems." ArXiv abs/2203.10610 (2022): n. pag.

[39] Shao, Yutong and Ndapa Nakashole. "ChartDialogs: Plotting from Natural Language Instructions." ACL (2020).

[40] M. Luong, E. Brevdo, and R. Zhao. Neural machine translation (seq2seq) tutorial. https://github.com/tensorflow/nmt, 2017.

[41] Sutskever, Ilya, Oriol Vinyals and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks." NIPS (2014).

[42] Hochreiter, Sepp and Jürgen Schmidhuber. "Long Short-Term Memory." Neural Computation 9 (1997): 1735-1780.

[43] Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." EMNLP (2014).

[44] Bahdanau, Dzmitry, Kyunghyun Cho and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." CoRR abs/1409.0473 (2015): n. pag.

[45] Luong, Thang, Hieu Pham and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation." EMNLP (2015).

[46] Vaswani, Ashish, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. "Attention is All you Need." ArXiv abs/1706.03762 (2017): n. pag.

[47] Devlin, Jacob, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." ArXiv abs/1810.04805 (2019): n. pag.

[48] Radford, Alec, Jeff Wu, Rewon Child, David Luan, Dario Amodei and Ilya Sutskever. “Language Models are Unsupervised Multitask Learners.” (2019).