Natural Language Interfaces with Fine-Grained User Interaction: A Case Study on Web APIs

Yu Su
University of California, Santa Barbara
ysu@cs.ucsb.edu

Ahmed Hassan Awadallah
Microsoft Research
hassanam@microsoft.com

Miaosen Wang
Microsoft Research
miwa@microsoft.com

Ryen W. White
Microsoft Cortana
ryenw@microsoft.com

ABSTRACT

The rapidly increasing ubiquity of computing puts a great demand on next-generation human-machine interfaces. Natural language interfaces, exemplified by virtual assistants like Apple Siri and Microsoft Cortana, are widely believed to be a promising direction. However, current natural language interfaces provide users with little help in case of incorrect interpretation of user commands. We hypothesize that the support of fine-grained user interaction can greatly improve the usability of natural language interfaces. In the specific setting of natural language interfaces to web APIs, we conduct a systematic study to verify our hypothesis. To facilitate this study, we propose a novel modular sequence-to-sequence model to create interactive natural language interfaces. By decomposing the complex prediction process of a typical sequence-to-sequence model into small, highly-specialized prediction units called modules, it becomes straightforward to explain the model prediction to the user, and solicit user feedback to correct possible prediction errors at a fine-grained level. We test our hypothesis by comparing an interactive natural language interface with its non-interactive version through both simulation and human subject experiments with real-world APIs. We show that with interactive natural language interfaces, users can achieve a higher success rate and a lower task completion time, which lead to greatly improved user satisfaction.

ACM Reference Format:
Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W. White. 2018. Natural Language Interfaces with Fine-Grained User Interaction: A Case Study on Web APIs. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3209978.3210013

1 INTRODUCTION

With the meteoric growth of the digital world and the popularization of computing devices like smartphones and Internet-of-Things (IoT) devices among less technically proficient people, new ways of human-computer interfacing are in great demand. Natural language (NL) is the most common communication method used by humans. Not surprisingly, natural language interfaces (NLIs) have been an aspirational goal in human-computer interaction since the very early days of digital computers [34]. They bear the promise of providing a unified interface for even technically non-proficient users to access a wide range of heterogeneous data, services, and devices.

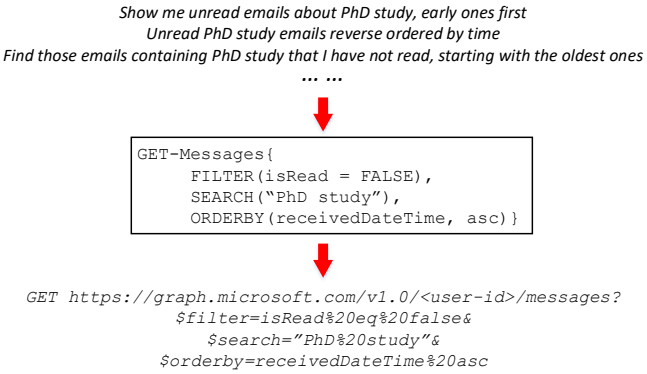

Figure 1: Example of natural language interface to web API. *Top*: Natural language utterances (commands). *Middle*: API frame. An abstract representation that can be converted into the real API call deterministically. *Bottom*: Real API call to the Microsoft email search API.

The core challenge of natural language interfaces is to map natural language utterances (commands) from users to some formal meaning representation, be it SQL for relational databases, SPARQL for knowledge bases, or API (application program interface) for software applications, that is understandable by computers. Recent advances in deep learning make it possible to develop generic natural language interfaces that are free of feature engineering and can more easily generalize to different domains. As a result, we have recently witnessed a growth in neural network based natural language interfaces to a wide range of data types such as knowledge bases [21, 35], relational database-like tables [24, 30, 38], and APIs to web services and Internet-of-Things devices [7, 27].

One of the main challenges facing natural language interfaces is that natural language is inherently ambiguous. Hence, it is unrealistic to expect a natural language interface to perfectly understand all

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210013

natural language commands. Additionally, it is difficult for a user to assess the results and decide whether or not the model was able to correctly interpret their commands. Even when they can do that, in case of erroneous results their only resort is to reformulate their command and try again. This is especially true with mainstream neural network models, which provide little insights to help users interpret the predictions made by the model.

In this paper, we study *interactive* natural language interfaces, which allow users to interact with the system and correct possible errors. In particular, we hypothesize that the support of fine-grained user interaction can greatly improve the usability of natural language interfaces. To test this hypothesis, we conduct a case study in the context of natural language interfaces to web APIs (NL2API). An example of NL2API can be found in Figure 1.

The mainstream neural network model for natural language interfaces is the sequence-to-sequence model [31]. However, it is difficult to create interactive natural language interfaces with the vanilla sequence-to-sequence model. To facilitate our case study on interactive natural language interfaces, we propose a novel modular sequence-to-sequence model. The main idea is to decompose the complex prediction process of a typical sequence-to-sequence model into small prediction units called *modules*. Each module is highly specialized at predicting a pre-defined kind of sequence output, and their prediction can therefore be easily explained to the user. The user can then verify the correctness of the prediction of each module, and give feedback to correct possible errors in the module predictions. For every specific command only a few modules will be triggered, and a specifically designed controller will read the input command to decide which modules to trigger. Both the controller and the modules are neural networks. We further propose an interaction mechanism based on the proposed model.

To test the hypothesis on interactive natural language interfaces, we design both simulation and human subject experiments with two deployed Microsoft APIs, which are used for searching emails and calendar events, respectively. In the simulation experiment, we show that the interactive NLI can greatly improve the prediction accuracy via only a small amount of extra user effort: with only one round of user interaction, testing accuracy can be improved from around 0.5 to over 0.9. In the human-subject experiment, we conduct a comparative study. We compare the interactive NLI with its non-interactive counterpart, which is similar to a traditional search engine: If the model prediction is incorrect, a user will reformulate the command and try again. Through log-based analysis and user survey, we find that the interactive NLI outperforms the non-interactive NLI on a variety of measures: The interactive NLI leads to higher task success rate, shorter task completion time (less user effort), and remarkably higher user satisfaction. 85% of the participants indicate that they prefer the interactive NLI over the non-interactive NLI.

In summary, this work makes major contributions in problem formulation, model, and experimentation:

- We conduct a systematic study on fine-grained user interaction in natural language interfaces with a focus on web APIs.
- We propose a novel modular sequence-to-sequence model to facilitate the creation of interactive natural language interfaces.
- We design both simulation and human subject experiments with real-world APIs to demonstrate the benefits of interactive natural language interface along several dimensions including task completion, user effort, and user satisfaction.

Table 1: API parameter types.

Parameter Type	Description
`SEARCH(String)`	Search for resources containing specific keywords
`FILTER(BoolExpr)`	Filter resources by some criteria, e.g., `isRead=False`
`ORDERBY(Property,Order)`	Sort resources on a property in 'asc' or 'desc' order
`SELECT(Property)`	Instead of full resources, only return a certain property
`COUNT()`	Count the number of matched resources
`Top(Integer)`	Only return the first certain number of results

2 NATURAL LANGUAGE INTERFACE TO WEB API

A web API is a set of operations, associated data definitions, and the semantics of the operations for accessing a Web-based software application. Web APIs provide the foundations for interacting with applications such as email and calendar, customer relation management [23], photo sharing services, social media platforms, online shopping, and the Internet-of-Things [11]. NL2API enables users to access a wide range of applications in a unified, natural way, while staying agnostic to the heterogeneity of data and services that they must handle when using traditional graphical user interfaces (e.g., learn and adapt to different graphical user interfaces to use different applications). As a result, NL2APIs have attracted increased attention in recent times [7, 25, 27].

The core task of NL2API is to map natural language utterances given by users into API calls. More specifically, we will follow the setting defined by Su et al.[27] and focus on web APIs that follow the REST architectural style [10], i.e., *RESTful APIs*. RESTful APIs are widely used for web services [2], IoT devices [11], as well as smartphone apps [37]. An example from [27] based on the Microsoft email search API[1] is shown in Figure 1. The top portion of the figure shows multiple natural language utterances. The same user intent can be expressed in syntactically-divergent ways in natural language, i.e., paraphrases, which should all be mapped to the same API call. The middle portion shows an API frame; which represents a more compact representation of RESTful API calls defined in [27], and can be mapped to the real API calls in a deterministic way. The bottom portion shows a real API call. It contains many irrelevant constituents such as URL conventions that could be distracting in natural language interfaces. We will use API frame in the following, and will use API frame and API call interchangeably.

A RESTful API (e.g., `GET-Messages`) consists of an HTTP verb (e.g., `GET`, `PUT`, and `POST`) and a set of resources (e.g., a user's emails). In addition, one can call an API with different parameters to specify advanced search requests, for example, filter by some properties of the resource (e.g., `subject`, `isRead`, `receivedDateTime` of an email) or search for some keywords. The full list of parameter

[1] https://developer.microsoft.com/en-us/graph/

types can be found in Table 1. An *API call* is an API with a list of parameters. It can be linearized into a sequence (Figure 1 middle).

Definition 2.1 (Natural language interface to Web API). Given an input utterance $\boldsymbol{x} = \{x_1, x_2, \ldots, x_m\}$, the task of a natural language interface to web API is to map $\boldsymbol{x}$ to the corresponding linearized API call $\boldsymbol{y} = \{y_1, y_2, \ldots, y_n\}$.

3 INTERACTIVE NATURAL LANGUAGE INTERFACE

In this section, we discuss the different levels of user interaction that a natural language interface may support, and propose a modular sequence-to-sequence model which naturally supports user interaction at the fine-grained parameter level.

3.1 User Interaction

NL2API maps a command to an API call, which can be executed and return the results to the user. Correspondingly, it is possible to enable interaction and solicit feedback from users at three levels: (1) Result level, by asking users to verify result correctness; (2) API call level, by asking users to verify API call correctness, and; (3) Parameter level, by asking users to interact with each parameter in the predicted API call.

The most straightforward way to interact is to execute the command and ask users to judge the correctness of the returned results. However, this approach has two problems. First, it is not always possible for a user to easily verify result correctness. If a user asked "*how many provinces are there in China?*" and a system said "*23*", how could the user know that the system's understanding is not "*the 9th prime number*" or "*the atomic number of vanadium*?" Second, the information provided by result correctness may be limited. If a user indicates that the provided results are incorrect, how much help does this new information provide to the system to select the correct API call from possibly thousands of candidates?

Alternatively, we can ask users to verify the correctness of the predicted API call. Such information is more definitive than result correctness. Although it may be difficult for general users to directly understand API calls, it is possible to design some rules to automatically convert API calls into natural language utterances (e.g., [27]), which can be easily understood. However, similar to result-level interaction, there is still the challenge of how to use this new information and how much help it can bring. It is not efficient if a user needs to decline tens of incorrect API calls before obtaining the correct one.

We believe it is more helpful if users can interact with the natural language interface at a finer-grained parameter level. For the example in Figure 1, if the natural language interface incorrectly predicts a parameter `FILTER(isRead = TRUE)`, the user may interact with the system and indicate that the parameter value should be changed to `FALSE`. Next, we will first review the mainstream sequence-to-sequence model for natural language interfaces. We then propose a modular sequence-to-sequence model, which naturally supports parameter-level interaction.

3.2 Sequence-to-Sequence Model

The core task of natural language interfaces, including NL2APIs, can often be cast into a sequence to sequence prediction problem:

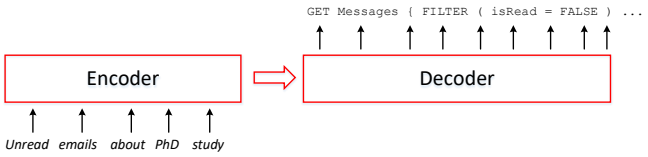

Figure 2: Vanilla sequence-to-sequence model for NL2API. In practice, constructs like bi-directional RNN encoder and attention mechanisms (see definitions below) are usually added to the vanilla model for better performance.

utterance sequence as input, and formal meaning representation sequence as output. The sequence-to-sequence (Seq2Seq) neural model [31] is a natural choice for this task, and has been widely used for natural language interfaces to knowledge bases [17, 28], relational databases [38], and web APIs [27]. Since we will use the Seq2Seq model as a building block in the modular Seq2Seq model, we first give its formal definition.

For an input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$, the Seq2Seq model estimates the conditional probability distribution $p(\boldsymbol{y}|\boldsymbol{x})$ for all possible output sequences $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$. The lengths m and n can be different, and both of them can be varied. An illustrative example is shown in Figure 2.

The *encoder*, which is implemented as a bi-directional recurrent neural network (RNN), first encodes $\boldsymbol{x}$ into a sequence of state vectors $(h_1, h_2, \ldots, h_m)$. Suppose ϕ is a randomly initialized word embedding layer that embeds every word into a low-dimensional vector, the state vectors of the forward RNN and the backward RNN are respectively computed as:

$$\begin{aligned} \overrightarrow{h}_i &= GRU_{fw}(\phi(x_i), \overrightarrow{h}_{i-1}) \\ \overleftarrow{h}_i &= GRU_{bw}(\phi(x_i), \overleftarrow{h}_{i+1}) \end{aligned} \tag{1}$$

where gated recurrent unit (GRU) as defined in [8] is used as the recurrence. We then concatenate the forward and backward state vectors, $h_i = [\overrightarrow{h}_i, \overleftarrow{h}_i], i = 1, \ldots, m$.

We use an attentive RNN as the *decoder*, which will generate the output tokens one at a time. We denote the state vectors of the decoder RNN as $(d_1, d_2, \ldots, d_n)$. The attention takes a form similar to [32] (also known as *additive attention*). For the decoding step j, the decoder is defined as follows:

$$\begin{aligned} d_0 &= \tanh(W_0[\overrightarrow{h}_m, \overleftarrow{h}_1]) \\ u_{ji} &= v^T \tanh(W_1 h_i + W_2 d_j) \\ \alpha_{ji} &= \frac{u_{ji}}{\sum_{i'=1}^{m} u_{ji'}} \\ h'_j &= \sum_{i=1}^{m} \alpha_{ji} h_i \\ d_{j+1} &= GRU([\phi(y_j), h'_j], d_j) \\ p(y_j|\boldsymbol{x}, y_{1:j-1}) &\propto \exp(U[d_j, h'_j]) \end{aligned} \tag{2}$$

where W_0, W_1, W_2, v and U are model parameters. The decoder first calculates normalized attention weights α_{ji} over encoder states, and get a summary state h'_j. The summary state is then used to calculate the next decoder state d_{j+1} and the output probability

distribution $p(y_j|\boldsymbol{x}, y_{1:j-1})$. During training, the sequence $y_{1:j-1}$ is supplied using the gold output sequence; during testing, it is generated by the decoder.

3.3 Modular Sequence-to-Sequence Model

We propose a novel modular sequence-to-sequence model (Figure 3) to enable fine-grained interaction of natural language interfaces. To achieve that, we decompose the decoder in the original Seq2Seq model into multiple interpretable components called *modules*. Each module is specialized at predicting a pre-defined kind of output, e.g., instantiating a specific parameter by reading the input utterance in NL2API. After some simple mapping, users can easily understand the prediction of any module, and interact with the system at the module level. It is similar in spirit to modular neural networks [3, 4, 26]. But to the best of our knowledge, this is the first work to study interactive natural language interfaces with modular neural networks. Also, different from previous modular neural networks, each module in our model generates a sequential output instead of a continuous state.

Module. We first define modules. A module is a specialized neural network, which is designed to fulfill a specific sequence prediction task. In NL2API, different modules correspond to different parameters. For example, for the `GET-Messages` API the modules are `FILTER(sender)`, `FILTER(isRead)`, `SELECT(attachments)`, `ORDERBY(receivedDateTime)`, `SEARCH`, etc. The task of a module, if triggered, is to read the input utterance and instantiate a full parameter. To do that, a module needs to determine its parameter values based on the input utterance. For example, given an input utterance "*unread emails about PhD study*", the `SEARCH` module needs to predict that the value of the `SEARCH` parameter is "*PhD study*", and generate the full parameter, "`SEARCH PhD study`", as its output sequence. Similarly, the `FILTER(isRead)` module needs to learn that phrases such as "*unread emails*", "*emails that have not been read*", and "*emails not read yet*" all indicate its parameter value is `False`.

It is natural to implement the modules as attentive decoders, similar to the original Seq2Seq model. However, instead of a single decoder for everything, now we have multiple decoders each of which is specialized in predicting a single parameter. Moreover, as we will show in Section 4, because each module has clearly defined semantics, it becomes straightforward to enable user interaction at the module level. Formally, a module M_k is an attentive decoder as defined in Eq (2), with the goal to estimate the conditional probability distribution $p_k(\boldsymbol{y}|\boldsymbol{x})$, where $\boldsymbol{y}$ is from the set of API frame symbols.

Controller. For any input utterance, only a few modules will be triggered. It is the job of the controller to determine which modules to trigger. Specifically, the controller is also implemented as an attentive decoder. Using the encoding of the utterance as input, it generates a sequence of modules, called the *layout*. The modules then generate their respective parameters, and finally the parameters are composed to form the final API call. Formally, the controller is an attentive decoder as defined in Eq (2), with the goal to estimate the conditional probability distribution $p_c(\boldsymbol{l}|\boldsymbol{x})$, where the layout $\boldsymbol{l}$ is from the set of modules.

Table 2: Example mapping of module output to natural language explanation. A few rules suffice for the mapping.

Parameter Syntax	Natural Language Explanation
`FILTER isRead = BOOLEAN`	is (not) read
`SEARCH KEYWORDS`	contains keyword KEYWORDS
`SELECT receivedDateTime`	return the receive time

Example. Take Figure 3 as example. The controller first reads the input utterance and generates a sequence of modules, `API`, `FILTER(isRead)`, and `SEARCH`. Each module then reads the input utterance again to generate their respective parameter, where the main work is to determine the correct parameter values based on the utterance.

Training Objective. Given a set of training examples $\{(\boldsymbol{x}_i, \boldsymbol{l}_i, \boldsymbol{y}_i)\}_{i=1}^N$, the loss function of the whole modular Seq2Seq model consists of three kinds of losses:

$$\Theta = \frac{1}{N}\sum_{i=1}^{N}(\Theta_{c,i} + \Theta_{m,i}) + \lambda\Theta_{L2}. \tag{3}$$

For the i-th example, the controller loss is a cross-entropy loss on the layout prediction:

$$\Theta_{c,i} = -\log p_c(\boldsymbol{l}_i|\boldsymbol{x}_i). \tag{4}$$

Suppose the gold layout of the i-th example $\boldsymbol{l}_i = \{M_1, M_2, \ldots, M_t\}$ with respective gold parameters $\{\boldsymbol{y}_{i,1}, \boldsymbol{y}_{i,2}, \ldots, \boldsymbol{y}_{i,t}\}$, the module loss is the average cross-entropy loss on the module predictions:

$$\Theta_{m,i} = -\frac{1}{t}\sum_{j=1}^{t}\log p_j(\boldsymbol{y}_{i,j}|\boldsymbol{x}_i). \tag{5}$$

Finally, we add an L2 regularization term Θ_{L2} with balance parameter λ to alleviate overfitting. We also apply dropout [12] on both the input and the output of GRU cells to alleviate overfitting.

4 INTERACTION MECHANISM

In this section we present our interaction mechanism based on the proposed modular Seq2Seq model.

Interpretable module output. The output of each module can be easily explained to the user. Because each module is highly specialized at predicting one pre-defined parameter, its output highly conforms to the syntax of that parameter. For example, for the `FILTER(isRead)` module, the parameter syntax is "`FILTER isRead = BOOLEAN`", where `BOOLEAN` is either `TRUE` or `FALSE`. Similarly, for the `SEARCH` module, the parameter syntax is "`SEARCH KEYWORDS`", where `KEYWORDS` is a sequence of keywords. Therefore, it is easy to use a simple rule to map the output of a module to a natural language phrase that is understandable by general users. Several examples are shown in Table 2.

Parameter value suggestion. Since the modules are neural decoders, each of them can generate a ranked list of outputs. For example, for the input utterance "*unread emails about PhD study*", the `SEARCH` module may generate the following list:

(1) `SEARCH PhD`

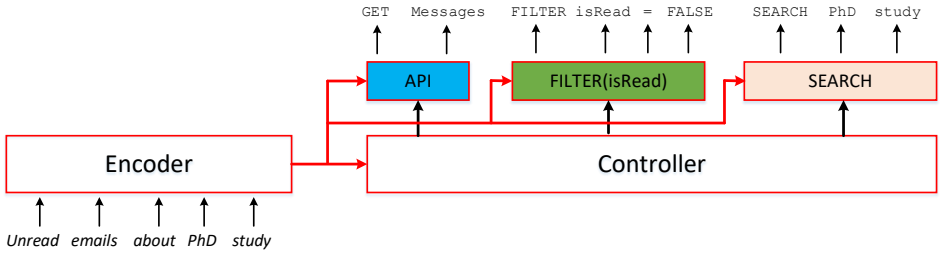

Figure 3: Modular sequence-to-sequence model. The controller triggers a few modules, each of which instantiates a parameter.

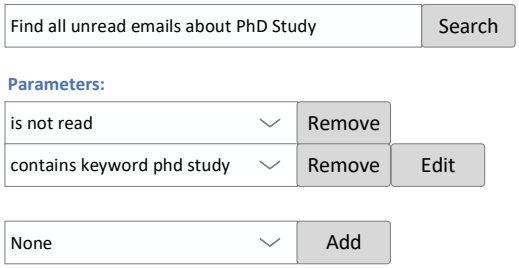

Figure 4: Interactive natural language interface. Once the user types in the command and clicks "Search," the system will generate the most probable API call from the modular Seq2Seq model, convert the output of each module into natural language, and show the results to the user. The user can then interact with the system using a number of operations such as adding or removing modules, selecting alternative parameter values from drop-down menus, or editing parameter values.

(2) `SEARCH PhD Study`
(3) `SEARCH PhD study emails`

Therefore, in addition to the top-ranked output, we can present to the user several plausible suggestions (mapped to natural language explanations as in Table 2). If the top-ranked output is incorrect, the user may find the correct one in the suggestion list[2].

Module suggestion. Sometimes the controller makes a mistake when predicting the layout and misses some module. We also provide a list of module suggestions and allow the user to add modules from the list. Currently we run all the modules of an API and include the top-ranked output in the suggestion list. One can also only keep a few most probable ones to reduce the number of suggestions.

Module removal. Similarly, the controller may make a mistake when predicting the layout and adds an unnecessary module. To address this, we allow the user to remove modules from the list. Currently, we allow the user to remove any module from the list returned by the model.

We design a graphical user interface (Figure 4) to accommodate all the above interaction components. The user is initially shown a query box where she can type her query and click search. Given an utterance, our model will come up with the most likely interpretation of the utterance and show it to the user. Additionally, a drop-down menu is shown corresponding to each module in the interpretation. For example, the utterance *"find all unread emails about PhD study"* shown in Figure 4 will result in the following API call: `GET-Messages{FILTER(isRead = FALSE), SEARCH("PhD study")}`. Hence, the interface will show the two modules for filtering based on `isRead` and searching. If any of the module output is incorrect, the user can click on the module output to select from a list of suggestions in a drop-down menu. In rare cases, the user can also click the "edit" button to input the desired parameter value. Finally, the user can also remove a module completely, or add a module from a drop-down list if some desired modules are missing.

It is worth noting that the interaction mechanism can also be implemented based on natural language communication instead of display and click in a graphical user interface. We have opted for a graphical user interface mainly because it naturally leads to a compact interface to accommodate all interaction components as in Figure 4, and allows for more efficient user interaction.

5 EVALUATION

In this section we experimentally evaluate the proposed modular Seq2Seq model and the interaction mechanism. The main goal is to test the hypothesis that *fine-grained user interaction can greatly improve the usability of natural language interfaces*. We carry out the study in two experimental settings: (1) Using a *simulated* user on a standard NL2API dataset, we show that the interaction mechanism can significantly improve the accuracy of NL2API, with only a small number of interactions. (2) Through a *human* user experiment, we show that an interactive natural language interface, compared with its non-interactive counterpart, leads to higher success rate, less user effort, and higher user satisfaction.

While the main goal is to study fine-grained user interaction, We also compare several models in a non-interactive experiment that performs a traditional evaluation over held-out test data. The goal is to show that modular Seq2Seq model can achieve competitive performance in comparison with other models, to support its use as the base model for the subsequent study on interactive natural language interfaces.

5.1 Experimental Setup

Dataset. We use the NL2API dataset released in [27] to train our model. It contains utterance-API call pairs for two deployed Microsoft APIs respectively for searching a user's emails (`GET-Messages`)

[2]The output space of a module is much smaller than the whole API call space, which makes the suggestion task easier.

Table 3: Dataset statistics.

API	Training	Validation	Testing
GET-Messages	3670	917	157
GET-Events	5036	1259	190

and calendar events (GET-Events). The dataset was collected via crowdsourcing, and is split into a training set and a testing set. The training set contains some noise from crowdsourcing, while the testing set is smaller but each example is manually checked for quality. For model selection purpose we further hold out 20% of the training data to form a validation set, and use the rest for training. The statistics can be found in Table 3. For the modular Seq2Seq model, there are 19 modules for each API.

This is a challenging dataset. A good portion of the testing set (close to 40%) involves API calls that are more complex than those covered by the training set (larger number of parameters than ever seen in the training set). It is designed to test model generalizability on more complex and unseen API calls. Also, because of the flexibility of natural language, the same API call can be represented using different natural language utterances, i.e., paraphrases. So even if an API call is covered by the training set with several utterances, the utterances in the testing set are still unseen in training. A good natural language interface therefore needs to be able to generalize to both unseen API calls and unseen utterances for covered API calls.

Measures. For the non-interactive experiment (Section 5.2) and the simulation experiment (Section 5.3), following the literature [17, 27], we use accuracy as the evaluation measure. It is the proportion of testing examples for which the top API call generated by the model exactly matches the correct API call. For the human subject experiment (Section 5.4), we use a variety of measure such as task success rate, completion time, and user satisfaction (more details later).

Implementation details. We implement the proposed modular Seq2Seq model in Tensorflow [1]. The Tensorflow Fold [22] library is employed to dynamically build the computation graph according to the layout prediction from the controller. We use Adam [18] as the optimizer. Hyper-parameters of the model are selected based on the validation set. State size of the encoder is 100, and state size of all the decoders, including the controller and the modules, are 200. The word embedding size is 300 for the encoder, and 50 for the decoders since their vocabulary is smaller. Input and output dropout rate of the GRU cells are 0.3 and 0.5, respectively. The balance parameter for L2 regularization is 0.001. We use a large mini-batch size. 2048, to fully take advantage of the dynamic batching [22], which significantly improves training speed. Early stopping based on the validation set is used.

5.2 Non-interactive Experiment

We first evaluate the modular Seq2Seq model in a non-interactive setting, where there is no user interaction involved. The goals are two-fold. First, through error analysis we can get additional insights into the challenge of NL2API. Second, we show that the modular Seq2Seq can achieve competitive performance compared with other alternatives, which supports its use as the basis for an interactive natural language interface.

Table 4: Model accuracy in the non-interactive experiment. Su et al. [27] use a vanilla Seq2Seq model for ranking API calls. The Seq2Seq model (second row) is the one with bi-directional RNN encoder and attentive decoder as defined in Section 3.2. Modular Seq2Seq model is the proposed model as defined in Section 3.3. Both of these models directly generate an API call as output. For GET-Events, the three models happen to make the same number of errors on the second test set, but on different examples.

Model/API	GET-Messages	GET-Events
Su et al. [27]	0.573	0.453
Seq2Seq	0.586	0.453
Modular Seq2Seq	0.599	0.453

The testing accuracies on the NL2API dataset are shown in Table 4. Each model is trained on the training set and evaluated on the testing set. As can be seen, the modular Seq2Seq model achieves comparable performance with other models.

We present an error analysis of the modular Seq2Seq model. The prediction of the model can have three types of errors, two from the controller, i.e., having extra modules or missing required modules in the predicted layout, and one from the modules, i.e., having incorrect prediction of parameter values (e.g., return read emails while the user wants to find unread emails). For GET-Messages, 87.3% of the error cases have missing modules, 25.4% have extra modules, and 9.5% have erroneous parameter values. For GET-Events, 77.9% of the error cases have missing modules, 23.1% have extra modules, and 8.6% have erroneous parameter value. Note that some error cases involve more than one type of errors. Therefore, most of the errors come from the controller. A promising future direction is to develop more advanced models for the controller. One possible way is to allow the controller to access the module states in addition to the input utterance, so that it knows which parts in the input utterance have been processed by which modules, and which parts are left unprocessed that may need some additional modules.

The current best accuracy is not sufficient for a practical natural language interface in real use: it will fail on roughly one half of the user commands. However, it should be noted that accuracy is a strict binary measure: A model is correct on a testing example only if the predicted API call exactly matches the correct one; otherwise, it gets zero score. But most of the time the predicted API calls are very close to the correct API calls, only missing one module or getting a parameter value slightly wrong. If users can interact with the model to correct such errors, the model accuracy can be greatly improved. With the original Seq2Seq model, it is difficult for users to correct possible errors. The modular Seq2Seq model makes it easier for users to understand model prediction, and interact with the model at the fine-grained module level to correct errors. In the next two experiments, we show the effectiveness of the interaction mechanism with both simulated users and real human subjects.

5.3 Simulation Experiment

Because the dataset contains the correct API call for each testing example, we can use it to simulate a human user to interact with the UI in Figure 4. Given a testing example, it first issues the utterance as input to the model. After obtaining the model prediction, the simulated user will use the interaction actions introduced in Section 4 to correct possible errors until the prediction matches the correct API call. We record the number of actions taken in this procedure. More specifically,

Behavior. At the beginning of a task, the simulated user has an utterance and the correct API call. It issues the utterance to the search box in Figure 4. After getting the initial model prediction, it will try to match the prediction with the correct API call, and if there are mismatches, it will carry out necessary actions to correct the mismatches in the following order: (1) If there are modules missing from the correct API call, add from the module list. (2) If there are extra modules not in the correct API call, remove the modules. (3) If there are modules with erroneous parameter value, first try to select from the drop-down suggestion list. If the correct parameter value is absent from the suggestion list, click the "edit" button and type in the correct parameter value.

Example. Suppose the utterance is "*unread emails about PhD study*" and the correct API call consists of two parameters, "`FILTER isRead = FALSE`" and "`SEARCH PhD study`", and the initial model prediction has three parameters, "`FILTER isRead = FALSE`", "`SEARCH PhD`", and "`SELECT attachments`". The simulated user will first remove the `SELECT` parameter because it knows this one is not in the correct API call. Then the simulated user will change the value of the `SEARCH` parameter from "`PhD`" to "`PhD study`" by selecting from the drop-down suggestion list. In total it takes two actions to convert the initial model prediction to the correct API call.

The experiment results are shown in Figure 5. When no interaction is involved (# of actions = 0), the model achieves the same accuracy as in the non-interactive experiment (Table 4). A small amount of user interaction can greatly improve the accuracy. Most remarkably, *with only one action from the simulated user, the accuracy can be improved to around 0.92 for both APIs.* This shows that most of the time the initial model prediction is quite reasonable, only one step away from the correct API call. However, this does not necessarily mean that one can easily develop a better model to do this last step without user interaction. One difficulty is that some utterances are inherently ambiguous and the correct interpretation depends on the specific user or context. For example, with the same utterance "*find the first unread email*", some users may mean the earliest one, while some other users may mean the last one. User interaction may be necessary to resolve such ambiguities and improve personalization and context awareness. In summary, the simulation experiment results show that the designed interactive NLI can lead to remarkably better accuracy with only a small amount of user interaction.

5.4 Human Subject Experiment

Study Methodology: To better understand the impact of the interactive and the standard approaches for NL2API on the user experience, we conducted a lab study using the web-based interface described earlier for both the standard and interactive modes. Both

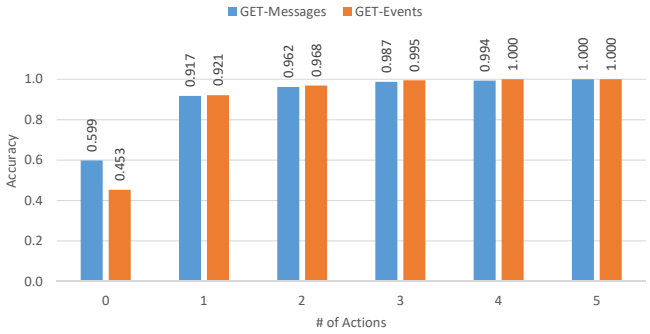

Figure 5: Simulation experiment results.

Table 5: Task examples.

Task Description	Difficulty
List unread messages	Easy
Find emails with high priority about 'PhD Study'	Medium
Find unread emails from John Smith with early ones first	Hard
Find the attachment of the most recent email in the Red category	Very Hard

modes are based on the same trained modular Seq2Seq model. The only difference is that the standard mode does not allow user interaction. The study used within-subject design with the interaction mode as the factor.

For the standard interaction mode, the user issues a query and gets the results back. The user examines the results and then decides if they satisfy her need or not. If they do, she stops. Otherwise, she may decide to try again by reformulating the query or give up. For the interactive mode, the user gets to interact with the results using the UX controls shown in Figure 4. For example, if the user decides that the keyword in the keyword filter should be changed, she may simply edit the filter. Similarly if she decides that the results should be ordered by the received time, she may select to add such a filter. The suggestions for adding, removing or editing the filters are provided by the model using the hypothesis space it builds as it interprets the natural language command.

Participants: Twenty people participated in the study. Participants were recruited via email advertising to a group of people affiliated with a large university in the US. Most participants were students from various backgrounds with ages ranging between 20 and 30 years old. All participants were experienced with using search engines and intelligent assistants such as Siri, Cortana or Alexa.

Protocol: Upon starting, participants were given an overview of the study. To familiarize themselves with the system, they were given 6 experimental trail tasks (3 for each interaction mode). Data from the trial tasks were not used for the results of this study. After completing the experimental trails, participants were given 10 tasks (5 for each mode), resulting a total of 200 tasks. The order of the tasks and which interaction mode they belong to was randomized for each participant. Examples of the task are shown in Table 5. Each task was assigned a difficulty level (based on the number of parameters in the target API call). Tasks across the two interaction modes had balanced difficulty level. To encourage participants to come up with their own formulation of the query text, we showed them the task description in another language (we used Chinese and recruited

Table 6: Average Number of Actions and Time to Completion for successful and abandoned tasks for the Standard and Interactive Modes.

Mode	Successful	#Action	Time to Completion
Standard	No	6.39	119.08
Standard	Yes	4.67	84.08
Standard	All	5.10	92.83
Interactive	No	4.30	47.40
Interactive	Yes	3.45	29.81
Interactive	All	3.73	35.54

participants that are fluent in both English and Chinese). Previous work has used similar techniques such as giving participants task descriptions in a different language [20] or in a recorded voice message [19].

After completing all the tasks, participants were asked to complete a questionnaire about the system they preferred and they were also asked to provide general feedback about the system. Participants also answered questions about their background and familiarity with the use of search and intelligent assistants.

Measures: Our overarching research question is: *what are the costs and benefits of the interactive NL2API compared to the standard search engine-like approach?* To answer this question, we used a combination of log-based analysis and survey questions. We implemented a rich instrumentation that records all interactions between the participants and the system. For example, all queries, clicks, query reformulation, etc. were logged using an event-based schema that also recorded the time stamp of the event, a task id and an anonymized study id. We also collected answers to survey questions after the experiment and linked it to the same study id. We describe more details of our measures as follows:

Task Completion: To study the effect of the interaction mode on the task completion rate, we measured the outcome of the completion of each task. Since the target result was known a priori, participants get feedback about whether the system was able to retrieve the correct answer or not. A task is considered successfully completed, only when the system is able to generate the interpretation that would retrieve the correct answer. Note that the participants were given feedback about whether the model got a task correct or not. In a real scenario, the users would be retrieving their own emails, appointments, etc. and they can decide whether the current answer satisfied their need or not. If the user gives up without getting the correct result, the task is considered as not successfully completed.

Effort: We also wanted to study the effort needed to achieve success in each interaction mode. We do that by measuring the total number of actions (e.g. queries, clicks, etc.) and the time to completion from the start to the end of each task.

User Satisfaction: Finally, we assessed the overall user satisfaction with the two interaction modes. We asked users to assess their satisfaction with both systems and to assess their relative preference between the two modes using a 5-point Likert scale.

Results:

Task Completion: The top portion of Figure 6 compares the success rate for the standard and interactive modes. Interactive mode helped participants complete tasks successfully at a higher rate than the

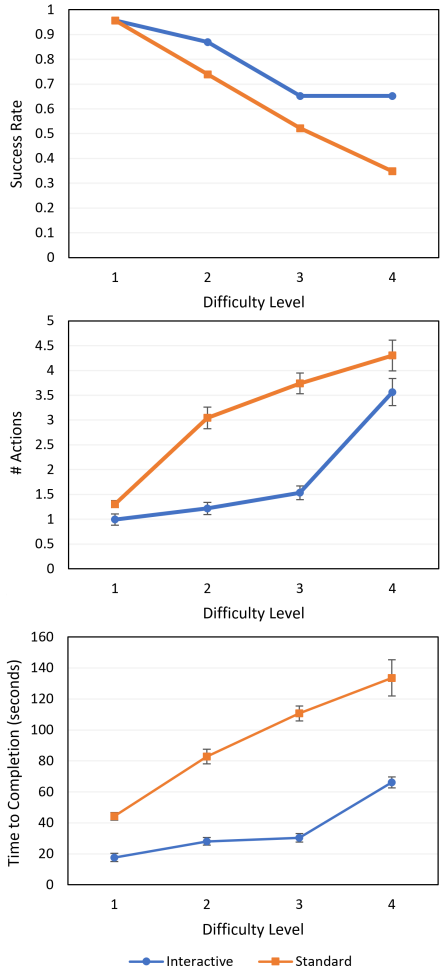

Figure 6: Success Rate (Top), Number of Actions (Middle) and Time to Completion (Bottom) for the Standard and Interactive Modes.

standard mode. It was particularly helpful with harder tasks where the model is more likely to make mistakes in translating the natural language command to the correct API call.

Effort: A 2 (interaction modes) by 4 (difficulty levels) ANOVA was performed for the the number of actions and time to completion for the standard and the interactive modes across different task difficulty levels. The result is also shown at middle and bottom portions of Figure 6. The interactive mode resulted in a smaller number of actions for all task difficulty levels ($p \ll 0.001$). The difference is smaller though for very hard tasks. This suggests that with harder tasks, participants had to either reformulate with the query or interact with the results to get to complete their tasks. Note that the actions are not equal though in terms of cost to the user. For example, reformulating the query is likely more expensive than editing the parameter of a filter module. To capture this, we use time to completion as a proxy to effort and compare the two modes as shown in the bottom portion of Figure 6. We see here that the interactive model resulted in faster task completion than the standard one ($p \ll 0.0001$), but unlike the number of actions, the

Table 7: Examples of tasks using the standard and the interactive mode. Each example is the sequence of actions taken by a user to solve a task. The examples are representative of user behaviors with different modes.

Standard Mode	
Action Type	Task Description
Query	show me unseen emails about PhD study
Query	show me emails about PhD study that I did not read
Query	show me the latest emails about PhD study that I did not read
Interactive Mode	
Action Type	Task Description
Query	show me unseen emails about PhD study
Add Module	New filter:"is not read"

gap was consistently large even for harder tasks. Since not all tasks have been completed successfully, we break down the different measure of effort (number of actions and time to completion) by whether the task was successfully completed or not in Table 6. A 2 (interaction modes) by 2 (successful or not) ANOVA was performed. As expected, we see that the participants had to perform a higher number of actions ($p \ll 0.001$) and longer time to completion ($p \ll 0.0001$) when the task was not completed successfully and the interactive mode resulted in less effort across the board.

User Satisfaction: Overall the interactive mode was overwhelmingly preferred over the standard mode for the scenario we studied, with 17 participants preferring the interactive mode to the standard mode. Participants also reported higher overall satisfaction level with the interactive mode (60% were satisfied or strongly satisfied with the mode) compared to only 35% reporting they were satisfied or strongly satisfied with the standard mode. Participants also indicated that they had to put in extra effort to complete tasks with the standard system, with only 25% of them reporting that they only needed little effort to complete the tasks. This number increases to 70% for the interactive system.

In summary, the user study showed that interactive mode provides several benefits over the standard mode and results in higher task completion rate, lower effort and higher overall user satisfaction. This can be more evident if we examine the utterances submitted by a user using the standard mode (see Table 7). In this example, the standard model interpreted the utterance mostly correctly except for missing the "is not read" filter. The user reformulated the query and this time the model got the missing filter right but missed the order by received time operator. After a third reformulation, the model was able to get the correct interpretation. Alternatively, if the user had used the interactive mode, she could have simply added the "is not read" filter which was ranked among the top 3 in the module suggestions. This would have resulted in much faster task completion and hence higher user satisfaction.

6 RELATED WORK

Natural language interface (also called semantic parsing in the computational linguistics community) research has spanned several decades [34]. Early NLIs are mostly rule-based. A set of rules are carefully designed to map natural language utterances in a domain to the corresponding meaning representation [5, 34]. Rule-based systems are characterized by a high precision on their admissible inputs. However, also salient is the brittleness of the systems when facing inputs not covered by the pre-defined rules. Over the past decade, statistical learning-based methods have gained momentum as they can naturally handle the uncertainty and ambiguity of natural language in a well-established statistical framework. Early learning-based methods were based on manually-defined features [6, 36]. With recent advances in deep learning, neural network based methods have become the mainstream for natural language interfaces [9, 21, 30, 35, 38], which are free of feature engineering and can more easily generalize to different domains. Our work follows this trend toward neural-network-based methods.

With the growth of web services, IoT devices, and mobile apps, natural language interfaces to API (NL2API) have attracted significant attention [7, 25, 27]. For example, Quirk et al. [25] study how to enable users to issue If-This-Then-That commands over a rich set of APIs including social media, mobile apps, and web services. Campagna et al. [7] present a virtual assistant system, at the core of which is a natural language interface to map user commands into APIs to IoT devices and mobile apps. Su et al. [27] study how to train an NL2API model by collecting training data from crowdsourcing, and propose a sequence-to-sequence model for NL2API. While the main goal of this paper is to study user interaction in natural language interfaces, we conduct our study in the context of NL2API, and benefit from the insights from previous studies.

Natural language interfaces that can seek feedback from users to improve prediction accuracy have also received significant recent attention [16, 20, 29, 33]. For example, Li and Jagadish [20] develop an interactive natural language interface to relational databases (NLIDB). The mapping from natural language commands to SQL queries is mainly done using a rule-based mechanism, and user feedback is solicited to resolve ambiguities in the rule-based mapping. In contrast, we focus on neural-network-based natural language interfaces targeting web APIs. Iyer et al. [16] and Su et al. [29] study NLIDB and knowledge base search, respectively, and ask users to verify the correctness of the final results generated by the systems, and employ user feedback to improve system accuracy. However, none of the previous studies allows for fine-grained (e.g., module level) user interaction with neural network models.

Also related is a line of research on crowd-powered dialog systems [13, 14]. Different from out approach of semantic parsing with user feedback, these approaches leverage crowd workers to address user commands, which reduces workload on users possibly at the expense of response latency. Our work also resembles mixed-initiative approaches [15], leveraging human-machine collaboration.

The idea of modular neural networks are also explored in related problems such as visual question answering [3, 4] and program synthesis [26]. For example, Rabinovich et al. [26] propose a novel abstract syntax network to generate the abstract syntax tree of programs. In abstract syntax network, different modules are composed together to generate a full abstract syntax tree. Each module usually only fulfills a simple task, like choosing a value from a predefined list. In our model, each module is itself an attentive decoder, and needs to generate a full parameter sequence by reading the input utterance. Moreover, the main goal of the proposed modular Seq2Seq model is to help create interactive natural language interfaces, which has not been explored previously.

7 CONCLUSIONS

We conducted a systematic study on fine-grained user interaction in natural language interfaces, focused on web APIs. To facilitate the creation of interactive natural language interfaces, we proposed a novel modular sequence-to-sequence model. By decomposing the prediction of a neural network into small, interpretable units called modules, the proposed model allows users to easily interpret predictions and correct possible errors. Through extensive simulation and human subject experiments with real-world APIs, we demonstrated that fine-grained user interaction can greatly improve the usability of natural language interfaces. Specifically, in the human subject experiment, we found that with the interactive natural language interface, users achieve a higher task success rate and a lower task completion time, greatly improving user satisfaction.

In this work, we focused on soliciting user feedback to improve prediction accuracy in a single session. Going forward, we are interested in the following question: *Given a new API, can we first cold-start an NL2API model with a reasonable prediction accuracy, and then improve it through user interaction?* In this vision, the interactivity of the NL2API helps form a closed data loop: It improves usability and thus attracts more users to use the system, which in turn accumulates more training data to improve the system.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467 [cs.DC]* (2016).

[2] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. 2004. Web services. In *Web Services*. Springer, 123–149.

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Learning to compose neural networks for question answering. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

[4] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[5] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases–an introduction. *Natural language engineering* 1, 1 (1995), 29–81.

[6] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

[7] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the International Conference on World Wide Web*. 341–350.

[8] Kyunghyun Cho, Bart van Merriënboer, ÇaÄğlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*. 1724–1734.

[9] Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[10] Roy T Fielding and Richard N Taylor. 2000. *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation.

[11] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. 2010. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE transactions on Services Computing* 3, 3 (2010), 223–235.

[12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580 [cs.NE]* (2012).

[13] Ting-Hao K. Huang, Joseph Chee Chang, and Jeffrey P. Bigham. 2018. Evorus: A Crowd-powered Conversational Assistant Built to Automate Itself Over Time. In *Proceedings of Conference on Human Factors in Computing Systems*.

[14] Ting-Hao K. Huang, Walter S. Lasecki, and Jeffrey P. Bigham. 2015. Guardian: A crowd-powered spoken dialog system for web apis. In *Third AAAI conference on human computation and crowdsourcing*.

[15] Yifen Huang and Tom M Mitchell. [n. d.]. Exploring hierarchical user feedback in email clustering. In *AAAI Enhanced Messaging Workshop*.

[16] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[17] Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[18] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]* (2014).

[19] Julia Kiseleva, Kyle Williams, Jiepu Jiang, Ahmed Hassan Awadallah, Aidan C. Crook, Imed Zitouni, and Tasos Anastasakos. 2016. Understanding User Satisfaction with Intelligent Assistants. In *Proceedings of the ACM SIGIR Conference on Human Information Interaction and Retrieval*. 121–130.

[20] Fei Li and H. V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proceedings of VLDB Endowment* 8, 1 (Sept. 2014), 73–84.

[21] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2016. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv:1611.00020 [cs.CL]* (2016).

[22] Moshe Looks, Marcello Herreshoff, DeLesley Hutchins, and Peter Norvig. 2017. Deep learning with dynamic computation graphs. In *Proceedings of the International Conference on Learning Representations*.

[23] Eric WT Ngai, Li Xiu, and Dorothy CK Chau. 2009. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert systems with applications* 36, 2 (2009), 2592–2602.

[24] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[25] Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[26] Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract Syntax Networks for Code Generation and Semantic Parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[27] Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building Natural Language Interfaces to Web APIs. In *Proceedings of the International Conference on Information and Knowledge Management*.

[28] Yu Su and Xifeng Yan. 2017. Cross-domain Semantic Parsing via Paraphrasing. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

[29] Yu Su, Shengqi Yang, Huan Sun, Mudhakar Srivatsa, Sue Kase, Michelle Vanni, and Xifeng Yan. 2015. Exploiting relevance feedback in knowledge graph search. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1135–1144.

[30] Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the International Conference on World Wide Web*.

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems*. 3104–3112.

[32] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Proceedings of the Annual Conference on Neural Information Processing Systems*.

[33] Sida I Wang, Percy Liang, and Christopher D Manning. 2016. Learning language games through interaction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[34] William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the American Federation of Information Processing Societies Conference*.

[35] Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

[36] Luke S. Zettlemoyer and Michael Collins. 2005. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*. 658–666.

[37] Li Zhang, Chris Stover, Amanda Lins, Chris Buckley, and Prasant Mohapatra. 2014. Characterizing mobile open apis in smartphone apps. In *Networking Conference, 2014 IFIP*. IEEE, 1–9.

[38] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv:1709.00103 [cs.CL]* (2017).