

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

**Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, Ryen W. White**

# Motivation

- There are a lot of Natural Language interfaces like Apple Siri, Microsoft Cortana that perform exceptionally well in human-machine interaction tasks.
- The core challenge of these interfaces is to map natural language utterances (commands) from users to some formal meaning representation.
- However, there are a few incorrect interpretations of user commands because natural language can be inherently ambiguous.
- It is difficult for a user to assess the results and decide whether or not the model was able to correctly interpret their commands.

# Motivation

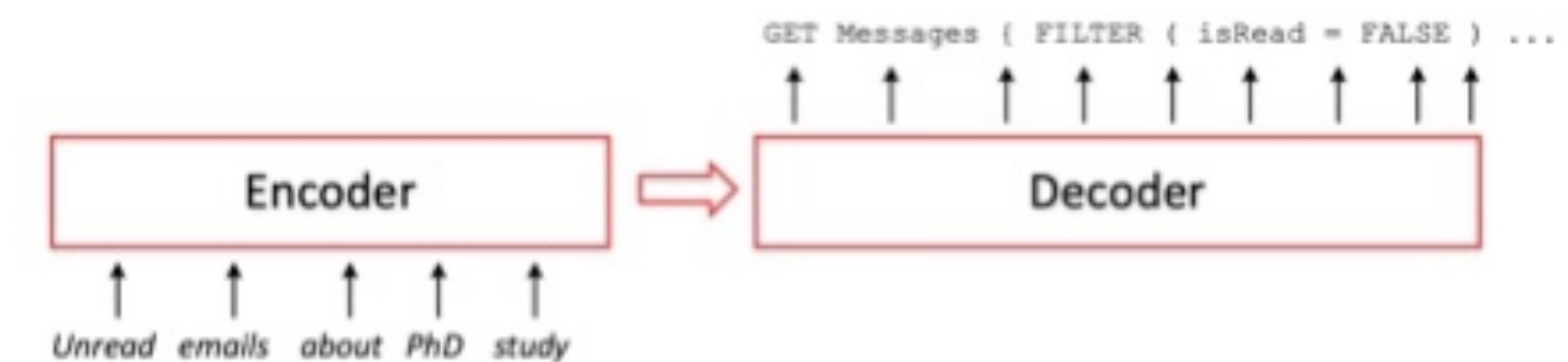
- The scope of this paper is to design a model where the complex process of translating user needs into machine understandable commands can be broken down into a few smaller modules that have a specific goal.
- These modules and their outputs can be easily understood by a common user with no technical background.
- This would facilitate the user to understand where the machine went wrong and in turn help to correct possible prediction errors at a fine-grained level.
- To test this hypothesis, a case study is conducted in the context of natural language interfaces to web APIs (NL2API). (<https://www.microsoft.com/en-us/download/details.aspx?id=58222>) whose core task is map natural language utterances given by users into API calls.

# Technical Contribution - User Interaction

- It is possible to enable interaction and solicit feedback from users at three levels:
  - A. Result level, by asking users to verify result correctness;
  - B. API call level, by asking users to verify API call correctness, and;
  - C. Parameter level, by asking users to interact with each parameter in the predicted API call.
- The most straightforward way is to execute the command and ask users to judge the correctness of the returned results.
- Alternatively, we can ask users to verify the correctness of the predicted API call. This is definitive than the final result itself.
- As it is not easy for the users to directly understand and interpret the API call itself, it is possible to design rules that automatically convert API calls to natural language that the user understands.
- Though this could be effective, the user might need to reject a lot of API calls before finally arriving at the one he intended.
- Therefore, it would be more helpful if the users could interact with the interface at a fine-grained parameter level.

# Technical Contribution - Related Work

- Most of the natural language interfaces use a mainstream neural network model called **sequence-to-sequence model**.
- For an input sequence  $x = (x_1, x_2, \dots, x_m)$ , the sequence-to-sequence model estimates the conditional probability distribution  $p(y|x)$  for all possible output sequences  $y = (y_1, y_2, \dots, y_n)$ .
- The **Encoder** is implemented as a bi-directional RNN which encodes the input  $x$  into sequence of state vectors  $h$  where a GRU (Gated Recurrence Unit) is used.
- **Decoder** is implemented as an attentive RNN which generates the output tokens one at a time using the state vectors generated by the encoder.



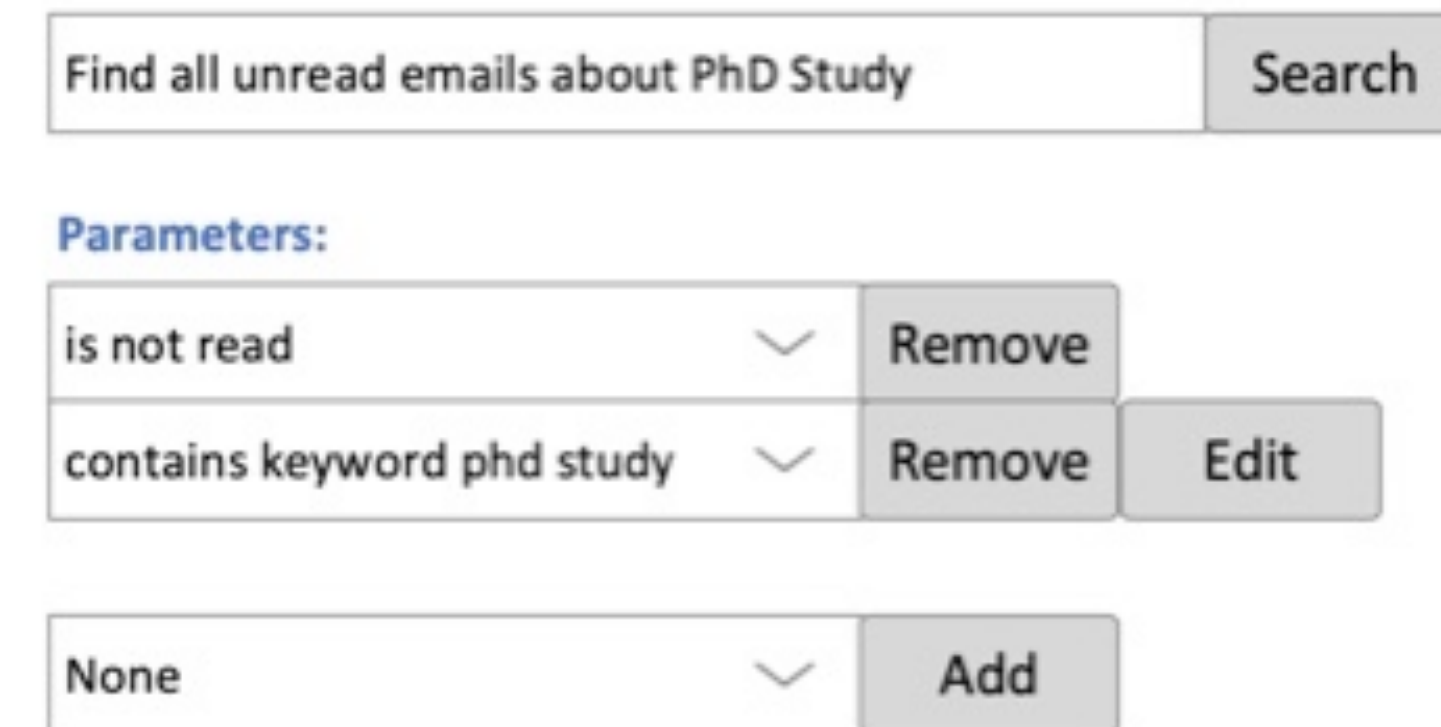
**Figure 2: Vanilla sequence-to-sequence model for NL2API.** In practice, constructs like bi-directional RNN encoder and attention mechanisms (see definitions below) are usually added to the vanilla model for better performance.

# Technical Contribution - Proposed Methodology

- The decoder which is used in sequence-to-sequence model is decomposed into multiple easily interpretable components called **modules** where a module is defined as a specialised neural network, which is designed to fulfil a specific sequence prediction task.
- The task of the module here is to read the input utterance and and instantiate a full parameter.
  - For example, for the GET-Messages API found in NL2API, the modules are FILTER(sender), FILTER(isRead), SELECT(attachments), ORDERBY(receivedDateTime), SEARCH, etc.
- The original decoder is now modified to include multiple decoders each of which is specialized in predicting a single parameter. Eg: A module sets the parameter FILTER(isRead=True).
- There is also a Controller whose job is to determine which modules to be triggered based on the utterance. Using the encoding of the utterance as input, it generates a sequence of modules, called the **layout**.
- The model is implemented in TensorFlow and Adam is used as the optimizer.

# Technical Contribution - Interaction Mechanism

- **Interpretable module output** : The output of each module can be easily explained to the user because each module is highly specialised at predicting one pre-defined parameter.
- **Parameter value suggestion** : Since the modules are neural decoders, each of them can generate a ranked list of outputs and these can be shown to user as plausible suggestions along with the top-ranked result.
- **Module suggestion** : A list of valid suggestions can be provided in case the controller makes a mistake while generating the layout.
- **Module removal** : The user can also remove modules he thinks are not relevant to his utterance.
- A user interface including all the above mentioned interactions is designed.



The screenshot shows a search bar at the top with the text "Find all unread emails about PhD Study" and a "Search" button. Below the search bar, the word "Parameters:" is displayed in blue. Underneath, there are two rows of parameter suggestions. The first row shows "is not read" with a downward arrow and a "Remove" button. The second row shows "contains keyword phd study" with a downward arrow, a "Remove" button, and an "Edit" button. At the bottom, there is a third row with "None" and a downward arrow, followed by an "Add" button.

| Parameters:                |        |
|----------------------------|--------|
| is not read                | Remove |
| contains keyword phd study | Remove |
| None                       | Add    |

# Evaluation

- **Dataset** : The NL2API is used to train the model. It contains utterance-API call pairs for two deployed Microsoft APIs respectively for searching a user's emails (GET-Messages) and calendar events (GET-Events).
- A good portion of the testing set involves API calls that are more complex than those covered by the training set and therefore good at testing generalizability on more complex and unseen API calls.
- The model is tested under three experimental settings :
  - A. Non-Interactive
  - B. Using a simulated user
  - C. Through a human user experiment
- **Accuracy** is used as the evaluation measure for the non interactive experiment (1) and various measures like success rate, completion time, and user satisfaction are used for (2&3).



# Results - Non-Interactive

- For the non-interactive experiment, the modular Seq2Seq model achieves comparable performance with other models.
- There is a possibility of three types errors : two from controller (having extra modules or missing required modules) and one from the modules, (having incorrect parameter values).
- It is observed that most of the errors are from the controller side.
- The model gets a positive score only when the prediction exactly matches the requirement. However, in most of the cases, the predictions are very close to the actual utterance.
- Therefore, user interaction can be very effective in both simulated users and human user.

| 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      |

# Results - Simulated and Human

- The simulated user issues an utterance and upon receiving the output, tries to correct (missing modules/removing modules/modifying parameter values) the prediction if it is mismatched.
- It can be noted that small amount of user interaction can greatly improves the accuracy. This shows that most of the time the initial model prediction is quite reasonable, only one step away from the correct API call.
- The process is similar for a human user - he issues a query and corrects the prediction if necessary using the user interface.
- It can be concluded that user interaction may be necessary to resolve ambiguities and improve personalization and context awareness.
- In summary, the simulation experiment results show that the designed interactive natural language interfaces can lead to remarkably better accuracy with only a small amount of user interaction.



Figure 5: Simulation experiment results.

# Future Work

- 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, it was demonstrated that fine-grained user interaction can greatly improve the usability of natural language interfaces.
- Going forward, we can ask 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?
- 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.
- After enough data is collected in this manner, there could be no need for any user interaction to get reasonable accuracy.

# My Thoughts

- This model was implemented only on the NL2API model where the RESTful API GET-MESSAGES had very limited properties (e.g., subject, isRead, receivedDateTime of an email)
- It was easier to decompose this model into modules and it was also easy to convert the output of the modules into user understandable utterances using a set of predefined rules.
- However, the API call could be complicated with a lot of properties and filters.
- A lot of modules need to be designed and it is more likely that the controller makes mistakes picking the layout (the selected modules given the utterance).
- The set of predefined rules to convert output of these numerous modules into utterances needs to be expanded to a great degree.
- All this effort could still be worth it if the model performs really well even for complicated API calls and achieves a great amount of user satisfaction with limited user interaction.