FLIN: A Flexible Natural Language Interface for Web Navigation

Sahisnu Mazumder*
Department of Computer Science
University of Illinois at Chicago, USA
sahisnumazumder@gmail.com

Oriana Riva
Microsoft Research
Redmond, USA
oriana.riva@microsoft.com

Abstract

AI assistants can now carry out tasks for users by directly interacting with website UIs. Current semantic parsing and slot-filling techniques cannot flexibly adapt to many different websites without being constantly re-trained. We propose *FLIN*, a natural language interface for web navigation that maps user commands to concept-level actions (rather than low-level UI actions), thus being able to flexibly adapt to different websites and handle their transient nature. We frame this as a ranking problem: given a user command and a webpage, FLIN learns to score the most relevant navigation instruction (involving action and parameter values). To train and evaluate FLIN, we collect a dataset using nine popular websites from three domains. Our results show that FLIN was able to adapt to new websites in a given domain.

1 Introduction

AI personal assistants, such as Google Assistant, can now interact directly with the UI of websites to carry out human tasks (Tech Crunch, 2019). Users issue commands to the assistant, and this executes them by typing, selecting items, clicking buttons, and navigating to different pages in the website. Such an approach is appealing as it can reduce the dependency on third-party APIs and expand an assistant's capabilities. This paper focuses on a key component of such systems: a natural language (NL) interface capable of mapping user commands (e.g., "*find an Italian restaurant for 7pm*") into navigation instructions that a web browser can execute.

One way to implement such an NL interface is to map user commands directly into low-level UI actions (button clicks, text inputs, etc.). The UI elements appearing in a webpage are embedded by concatenating their DOM attributes (tag, classes, text, etc.). Then, a scoring function (Pasupat et al., 2018) or a neural policy (Liu et al., 2018a) are trained to identify which UI element best supports a given command. Learning at the level of UI elements is effective, but only in controlled (UI elements do not change over time (Shi et al., 2017a)) or restricted (single applications (Branavan et al., 2009)) environments. This is not the case in the "real" web, where (i) websites are constantly updated, and (ii) a user may ask an assistant to execute the *same* task in *any* website of their choice (e.g., ordering pizza with dominos.com or pizzahut.com). The transient nature and diversity of the web call for an NL interface that can *flexibly adapt* to environments with a *variable* and *unknown* set of actions, without being constantly re-trained.

To achieve this goal, we take two steps. First, we conceptualize a new way of designing NL interfaces for web navigation. Instead of mapping user commands into low-level UI actions, we map them into meaningful "*concept-level*" actions. Concept-level actions are meant to *express what a user perceives when glancing at a website UI*. In the example shown in Figure 1, the homepage of OpenTable has a concept-level action "*Let's go*" (where "Let's go" is the label of a search button), which represents the concept of searching something which can be specified using various parameters (a date, a time, a number of people and a search term). Intuitively, websites in a given domain (say, all restaurant websites) share semantically-similar concept-level actions and the semantics of a human task tend to be time invariant. Hence, learning at the level of concept-level actions can lead to a more flexible NL interface.

However, while concept-level actions vary less than raw UI elements, they still manifest with different representations and parameter sets across websites. Searching a restaurant in opentable.com, for example, corresponds to an action "*Let's go*" which supports up to four parameters; in yelp.com, the same action is called "*search*" and supports two parameters (search term and location). Websites in

*Work done while interning at Microsoft Research.

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 2777–2788
June 6–11, 2021. ©2021 Association for Computational Linguistics

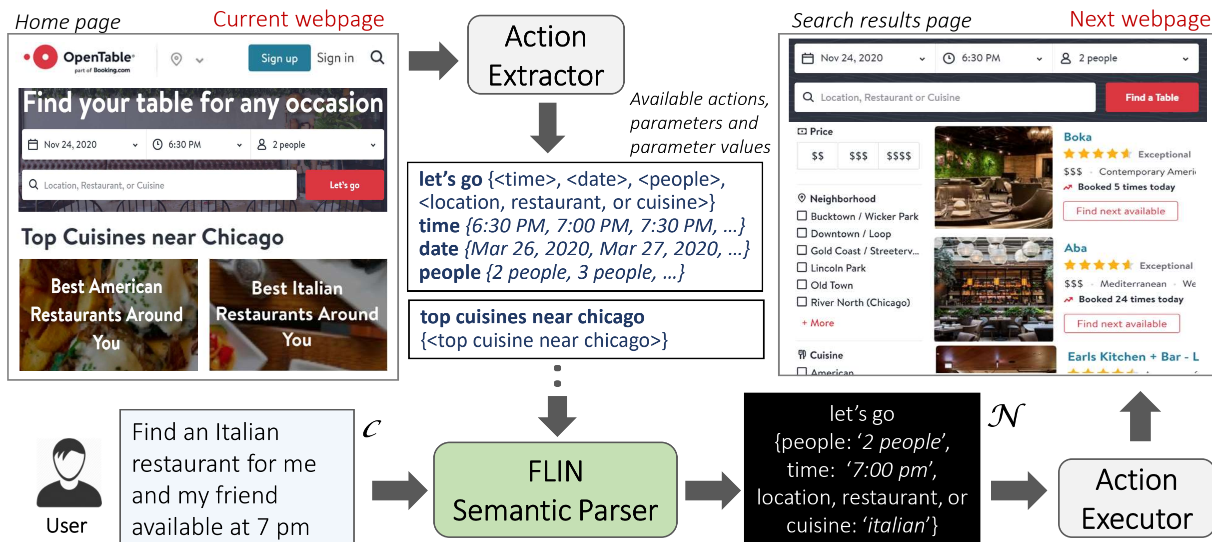

Figure 1: Web task execution driven by NL commands in the OpenTable website. The user command is mapped to the concept-level action "Let's go" whose execution causes the transition from the home to the search results page.

one domain may also have different action types (e.g., making a table reservation vs. ordering food).

Our second insight to tackle this problem is to leverage semantic parsing methods in a novel way. Traditional semantic parsing methods (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2007; Branavan et al., 2009; Lau et al., 2009; Thummalapenta et al., 2012) deal with environments that have a *fixed* and *known* set of actions, hence cannot be directly applied. Instead, we propose *FLIN*, a new semantic parsing approach where instead of learning how to map NL commands to executable logical forms (as in traditional semantic parsing), we leverage the semantics of the symbols (name of actions/parameters and parameter values) contained in the logical form (the navigation instruction) to *learn how to match it* against a given command. Specifically, we model the semantic parsing task as a *ranking problem*. Given an NL command c and the set of actions A available in the current webpage, FLIN scores the actions with respect to c. Simultaneously, for each parameter p of an action, it extracts a phrase m from c that expresses a value of p, and then scores p's values with respect to m to find the best value assignment for p. Each action with its associated list of parameter value assignments represents a candidate navigation instruction to be ranked. FLIN learns a net score for each instruction based on corresponding action and parameter value assignment scores, and outputs the highest-scored instruction as the predicted navigation instruction.

To collect a dataset for training and testing FLIN, we built a simple rule-based *Action Extractor* tool that extracts concept-level actions along with their parameters (names and values, if available) from webpages. The implementation and evaluation of this tool is out of scope for this paper.[1] In a complete system, illustrated in Figure 1, we envision the Action Extractor to extract and pass the concept-level actions present in the current webpage to FLIN, which computes a candidate navigation instruction $\mathcal{N}$ to be executed by an *Action Executor* (e.g., a web automation tool such as Selenium (2020) or Ringer (Barman et al., 2016)).

Overall, we make the following contributions: **(1)** we conceptualize a new design approach for NL interfaces for web navigation based on concept-level actions; **(2)** we build a match-based semantic parser to map NL commands to navigation instructions; and **(3)** we collect a new dataset based on nine websites (from restaurant, hotel and shopping domains) and provide empirical results that verify the generalizability of our approach. Code and dataset are available at https://github.com/microsoft/flin-nl2web.

2 Related Work

Semantic parsing has long been studied in NLP with applications to databases (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2007; Ferré, 2017),

[1] The tool processes a webpage's DOM tree (structure and attributes) and visual appearance (using computer vision techniques) to extract actions along with names and values of their parameters (if any). It is an active area of research (Nguyen and Csallner, 2015; Liu et al., 2018b; Chen et al., 2020b,a).

knowledge-based question answering (Berant et al., 2013; Yih et al., 2015), data exploration and visual analysis (Setlur et al., 2016; Utama et al., 2018; Lawrence and Riezler, 2016; Gao et al., 2015), robot navigation (Artzi and Zettlemoyer, 2013; Tellex et al., 2011; Janner et al., 2018; Guu et al., 2017; Fried et al., 2018; Garcia et al., 2018), object manipulation (Frank and Goodman, 2012), object selection in commands (Golland et al., 2010; Smith et al., 2013), language game learning (Wang et al., 2016), and UI automation (Branavan et al., 2009; Fazzini et al., 2018; Zhao et al., 2019). This work assumes environments with a fixed and known set of actions, while we deal with *variable* and *unknown* sets of actions.

Work on NL-guided web task execution includes learning from demonstrations (Allen et al., 2007), building reinforcement learning agents (Shi et al., 2017b; Liu et al., 2018a), training sequence to sequence models to map natural language commands into web APIs (Su et al., 2017, 2018), and generating task flows from APIs (Williams et al., 2019). These techniques assume different problem settings (e.g., reward functions) or deal with low-level web actions or API calls. Unlike FLIN, they do not generalize across websites.

Pasupat et al. (2018) propose an embedding-based matching model to map natural language commands to low-level UI actions such as hyperlinks, buttons, menus, etc. Unlike FLIN, this work does not deal with predicting parameter values (i.e., actions are un-parametrized).

Models that jointly perform intent detection and slot filling (Guo et al., 2014; Liu and Lane, 2016; Chen et al., 2019) are not applicable to our problem for three reasons. First, they are trained on a per-application basis using application-specific intent and slot labels, and thus cannot generalize across websites. Second, they semantically label words in an utterance, but do not do value assignment, hence they cannot output executable navigation paths. Third, they perform multi-class classification (i.e., they assume only one intent to be true for a user query) and have no notion of state (e.g., current webpage). They do not deal with intents with overlapping semantics which may occur across pages of the same website (e.g., in the example shown in Figure 1, the same user query may map to the action "*Let's go*" in the OpenTable's home page or to the action "*Find a Table*" in the search results page).

3 Problem Formulation

Let $A_w = \{a_1, a_2, ..., a_n\}$ be the set of concept-level actions available in a webpage w. Each action $a \in A_w$ is defined by an *action name* n_a and a set of K *parameters* $P_a = \{p_1, p_2, ..., p_K\}$. Each parameter $p \in P_a$ is defined by a name[2] and a domain $dom(p)$ (i.e., a set of values that can be assigned to parameter p), and can be either *closed domain* or *open domain*.

For closed-domain parameters, the domain is bounded and consists of a finite set of values that p can take; the set is imposed by the website UI, such as the available colors and sizes for a product item or the available reservation times for a restaurant.

For open-domain parameters, the domain is, in principle, unbounded, but, in practice, it consists of all words/phrases which can be extracted from an NL command c. With reference to Figure 1, the "*let's go*" (search) action has $n_a =$"*let's go*" and $P_a = \{$"*time*", "*date*", "*people*", "*location, restaurant, or cuisine*"$\}$. The first three parameters are closed domain and the last one (the search term) is open domain. The Action Extractor module (Figure 1) names actions and parameters after labels and texts appearing in the UI (or, if absent, using DOM attributes); it also automatically scrapes values of closed-domain parameters (from drop-down menus or select lists).

Given the above setting, our goal is to map an NL command c issued in w into a navigation instruction $\mathcal{N}$, consisting of a correct *action name* n_{a^*} corresponding to action $a^* \in A_w$ and an associated list of $m \leq |P_{a^*}|$ correct *parameter-value assignments*, given by $\{(p_i = v'_j) \mid p_i \in P_{a^*}, v'_j \in dom(p_i), 0 \leq i \leq K, 1 \leq j \leq |dom(p_i)| \}$.

4 The FLIN Model

The task of solving the above semantic parsing problem can be decomposed into two sub-tasks: **(i) action recognition**, i.e., recognizing the action $a \in A_w$ intended by c, and **(ii) parameter recognition and value assignment**, i.e., deciding whether a parameter of an action is expressed in c and, if so, assigning the value to that parameter. A parameter *is expressed* in c by a mention (word or phrase). For example, in Figure 1, "*me and my friend*" is a mention of parameter "*people*" in c and a correct parsing should map it to the domain value "*2 people*". Thus, the second sub-task involves first

[2]We use the same notation p to refer to the parameter and the parameter name interchangeably.

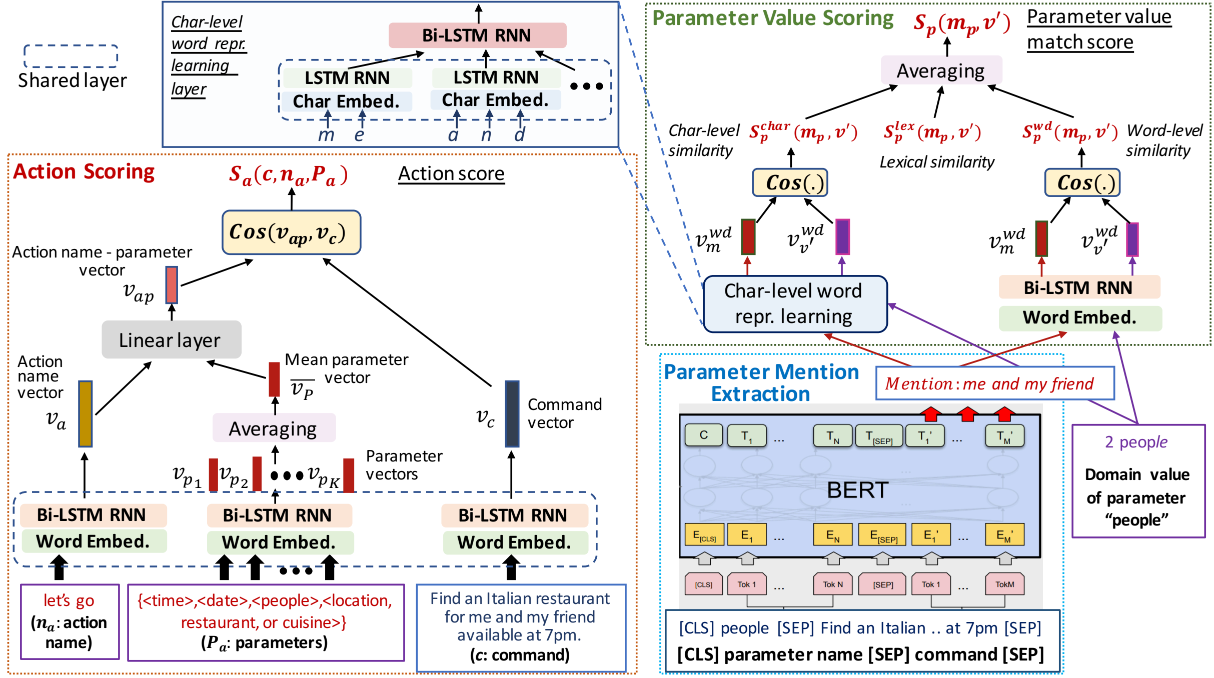

Figure 2: Architecture of FLIN including Action Scoring, Parameter Mention Extraction and Parameter Value Scoring components [BERT block diagram courtesy: (Devlin et al., 2019)].

extracting a mention of a given parameter from c, and then matching it against a set of domain values to find the correct value assignment. For an open-domain parameter, the extracted mention becomes the value of the parameter and no matching is needed, e.g. in Figure 1, the mention "*Italian*" should be assigned to the parameter "*location, restaurant, or cuisine*".

With reference to Figure 2, FLIN consists of four components, designed to solve the aforementioned two sub-tasks: **(1) Action Scoring**, which scores each available action with respect to the given command (§4.1); **(2) Parameter Mention Extraction**, which extracts the mention (phrase) from the command for a given parameter (§4.2); **(3) Parameter Value Scoring**, which scores a given mention with a closed-domain parameter value or rejects it if no domain values can be mapped to the mention (§4.3); and **(4) Inference** (not shown in the figure), which uses the scores of actions and parameter values to infer the action-parameter-value assignment with the highest score as the predicted navigation instruction (§4.4).

4.1 Action Scoring

Given a command c, we score each action $a \in A_w$ to measure the similarity of a and c's intent. We loop over the actions in A_w and their parameters to obtain a list of action name and parameters pairs (n_a, P_a), and then score them with respect to c.

To score each pair (n_a, P_a), we learn a neural network based scoring function $S_a(.)$ that computes its similarity with c. We represent c as a sequence $\{w_1, w_2, ..., w_R\}$ of R words. To learn a vector representation of c, we first convert each w_i into corresponding one-hot vectors x_i, and then learn embeddings of each word using an embedding matrix $E_w \in \mathbb{R}^{d \times |V|}$ as $\mathbf{v}_i{=}E_w.x_i$, where V is the word vocabulary. Next, given the word embedding vectors $\{\mathbf{v}_i \mid 1 \leq i \leq |R|\}$, we learn the forward and backward representation using a Bi-LSTM network (Schuster and Paliwal, 1997). Let the final hidden state for forward LSTM and backward LSTM, after consuming $\{\mathbf{v}_i \mid 1 \leq i \leq R\}$, be $\overrightarrow{\mathbf{h}_R}$ and $\overleftarrow{\mathbf{h}_1}$ respectively, we learn a joint representation of c as $\mathbf{v}_c = [\overrightarrow{\mathbf{h}_R}; \overleftarrow{\mathbf{h}_1}] \in \mathbb{R}^{2d}$, where [;] denotes concatenation.

Next, we learn a vector representation of (n_a, P_a). We use the same word embedding matrix E_w and Bi-LSTM layer to encode the action name n_a into a vector $\mathbf{v}_a{=}BiLSTM(n_a)$. Similarly, we encode each parameter $p \in P_a$ into a vector $\mathbf{v}_p{=}BiLSTM(p)$, and compute the *net parameter semantics* of action a as the mean of the parameter vectors ($\overline{\mathbf{v}_p}{=}mean\{\mathbf{v}_p \mid p \in P_a\}$). Finally, to learn the overall semantic representation of (n_a, P_a), we concatenate $\mathbf{v}_a$ and $\overline{\mathbf{v}_p}$ and learn a combined representation using a feed-forward (FF)

layer as

$$\mathbf{v}_{ap} = tanh(W_a.[\mathbf{v}_a; \overline{\mathbf{v}_p}] + b_a) \quad (1)$$

where $W_a \in \mathbb{R}^{4d\times 2d}$ and $b_a \in \mathbb{R}^{2d}$ are weights and biases of the FF layer, respectively.

Given $\mathbf{v}_c$ and $\mathbf{v}_{ap}$, we compute the intent similarity between c and (n_a, P_a) using cosine similarity:

$$\begin{aligned} S_a(c, n_a, P_a) &= \frac{1}{2}\,[\,cosine(\mathbf{v}_c, \mathbf{v}_{ap}) + 1] \\ &= \frac{1}{2}\,(\frac{\mathbf{v}_c.\mathbf{v}_{ap}}{||\mathbf{v}_c||||\mathbf{v}_{ap}||} + 1) \end{aligned} \quad (2)$$

where $||\cdot||$ denotes euclidean norm of a vector. $S_a(\cdot) \in [0, 1]$ is computed for each $a \in A_w$ (and is used in inference, §4.4).

The parameters of $S_a(.)$ are learned by minimizing a margin-based ranking objective $\mathcal{L}_a$, which encourages the scores of each positive (n_a, P_a) pair to be higher than those of negative pairs in w:

$$\mathcal{L}_a = \sum_{q\in Q^+}\sum_{q'\in Q^-} max\{S(q') - S(q) + 0.5,\ 0\} \quad (3)$$

where Q^+ is a set of positive (n_a, P_a) pairs in w and Q^- is a set of negative (n_a, P_a) pairs obtained by randomly sampling action name and parameter pairs (not in Q^+) in w.

4.2 Parameter Mention Extraction

Given a command c and a parameter p, the goal of this step is to extract the correct mention m_p of p from c. In particular, we aim to predict the text span in c that represents m_p. We formulate this task as a question-answering problem, where we treat p as a question, c as a paragraph, and m_p as the answer. We fine-tune a pre-trained BERT (Devlin et al., 2019) model[3] to solve this problem.

As shown in Figure 2 (bottom-right), we represent p and c as a pair of sentences packed together into a single input sequence of the form [CLS] p [SEP] c, where [CLS], [SEP] are special BERT tokens. For tokenization, we use the standard WordPiece Tokenizer (Wu et al., 2016). From BERT, we obtain T_i as output token embedding for each token i in the packed sequence. We only introduce a mention start vector $S \in \mathbb{R}^H$ and a mention end vector $E \in \mathbb{R}^H$ during fine-tuning. The probability of word i being the start of the mention is computed as a dot product between T_i and S followed by a softmax over all of the words in c: $P_i = \frac{e^{S\cdot T_i}}{\sum_j e^{S\cdot T_j}}$. The analogous formula is used to compute the end probability P_j. The position i (position $j > i$) with highest start (end) probability P_i (P_j) is predicted as start (end) index of the mention and the corresponding tokens in c are combined into a word sequence to extract m_p. We fine-tune BERT by minimizing the training objective as the sum of the log-likelihoods of the correct start and end positions. We train BERT to output [CLS] as m_p, if p is not expressed in c (no mention is identified, hence p is discarded from being predicted).

[3]https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/1

4.3 Parameter Value Scoring

Once the mention m_p is extracted for a closed-domain parameter p, we learn a neural network based scoring function $S_p(\cdot)$ to score each p's value $v' \in dom(p)$ with respect to m_p. If p is open-domain, parameter value scoring is not needed.

The process is similar to that of action scoring, but, in addition to *word-level semantic similarity*, we also compute *character-level* and *lexical-level* similarity between v' and m_p. In fact, v' and m_p often have partial lexical matching. For example, given the domain value “7:00 PM" for the parameter “*time*", possible mentions may be “***7** in the evening*”, “*19:**00** hrs*", “*at **7 pm***", etc., where partial lexical-level similarity is observed. However, learning word-level and character-level semantic similarities is also important as “PM" and “evening" as well as “7:00 PM" and “19:00" are lexically distant to each other, but semantically closer.

Word-level semantic similarity. We use the same word embedding matrix E_w used in action scoring to learn the word vectors for both m_p and v'. We use a Bi-LSTM layer (not shared with Action Scoring) to encode mention (value) into a word-level representation vectors $\mathbf{v}_m^{wd}$ ($\mathbf{v}_{v'}^{wd}$). We compute the word-level similarity between m_p and v' as

$$S_p^{wd}(m_p, v') = \tfrac{1}{2}\,[\,cosine(\mathbf{v}_m^{wd}, \mathbf{v}_{v'}^{wd}) + 1] \quad (4)$$

Character-level semantic similarity. We use a character embedding matrix E_c to learn the character vectors for each character composing the words in m_p and v'. To learn the character-level vector representation $\mathbf{v}_m^{char}$ of m_p, we first learn the word vector for each word in m_p by composing the character vectors in sequence using an LSTM network (Hochreiter and Schmidhuber,

1997), and then compose the word vectors for all mention words using a BiLSTM layer to obtain $\mathbf{v}_m^{char}$. Similarly, the character-level vector representation $\mathbf{v}_{v'}^{char}$ of v' is obtained. Next, we compute the character-level similarity between m_p and v' as

$$S_p^{char}(m_p, v') = \frac{1}{2}\left[cosine(\mathbf{v}_m^{char}, \mathbf{v}_{v'}^{char}) + 1\right] \quad (5)$$

Lexical-level similarity. We use a *fuzzy string matching score* (using the Levenshtein distance to calculate the differences between sequences)[4] and a custom *value matching score* which is computed as the fraction of words in v' that appear in m_p; then we compute a linear combination of these similarity scores (each score $\in [0, 1]$) as the net lexical-level similarity score, denoted as $S_p^{lex}(m_p, v') \in [0, 1]$.

Net value-mention similarity score. It is the mean of the three scores above: $S_p(m_p, v') = mean\{S_p^{wd}(m_p, v'), S_p^{char}(m_p, v'), S_p^{lex}(m_p, v')\}$.

The parameters of $S_p(\cdot)$ are learned by minimizing a margin-based ranking objective $\mathcal{L}_p$, which encourages the scores $S_p(.)$ of each mention and positive value pair to be higher than those of mention and negative value pairs for a given p, and can be defined following the previously defined $\mathcal{L}_a$ (See eq. 3).

4.4 Inference

The inference module takes the outputs of Action Scoring, Parameter Mention Extraction and Parameter Value Scoring to compute a net score $S_{ap}(.)$ for each action $a \in A_w$ and associated list of parameter value assignment combinations. Then, it uses $S_{ap}(.)$ to predict the navigation instruction.

Parameter value assignment. We first infer the value to be assigned to each $p \in P_a$, where the predicted value $\hat{v_p}$ for a closed-domain p is given by $\hat{v_p} = \arg\max_{v' \in dom(p)} S_p(m_p, v')$ *provided* $S_p(m_p, v') \geq \rho$. Here, ρ is a threshold score (tuned empirically) for parameter value prediction.

While performing the value assignment for p, we consider $S_p(m_p, \hat{v_p})$ as the confidence score for p's assignment. If $S_p(m_p, v') < \rho$ for all $v' \in dom(p)$, we consider the confidence score for p as 0, implying m_p refers to a value which does not exist in $dom(p)$; p is discarded and no value assignment is done. If p is an open-domain parameter, m_p is inferred as $\hat{v_p}$ with a confidence score of 1.

If *all* $p \in P_a$ are discarded from the prediction for a *parametrized* action $a \in A_w$, we discard a, as a no longer becomes executable in w.

Once we get all confidence scores for all value assignments for all $p \in P_a$, we compute the average confidence score $\overline{S_p}(P_a)$, and consider it to be the *net parameter value assignment score* for a.

Navigation instruction prediction. Finally, we compute the overall score for a given action a and associated list of parameter value assignments as $S_{ap} = \alpha * S_a(c, n_a, P_a) + (1 - \alpha) * \overline{S_p}(P_a)$, where α is a linear combination coefficient empirically tuned. The predicted navigation instruction for command c is the action and associated parameter value assignments with the highest S_{ap} score.

5 Evaluation

We evaluate FLIN on nine popular websites from three representative domains: *(i) Restaurants (R)*, *(ii) Hotels (H)*, and *(iii) Shopping (S)*. We collect labelled datasets for each website (§5.1) and perform *in-domain cross-website evaluation*. Specifically, we train one FLIN model for each domain using one website, and test on the other (two) websites in the same domain. Ideally, a single model could be trained by using the training data of *all* three domains and applied to all test websites, but we opt for *domain-specific* training/evaluation to better analyze how FLIN leverages the semantic overlap of concept-level actions (that exists across in-domain websites) to generalize to new websites. We discard *cross-domain* evaluation because the semantics of actions and parameters do not significantly overlap across our three domains.

5.1 Experimental Setup

To train and evaluate FLIN, we collect two datasets: (i) **WebNav** consists of (English) command and navigation instruction pairs, and (ii) **DialQueries** consists of (English) user utterances extracted from existing dialogue datasets paired with navigation instructions.

To collect WebNav, given a website and a task it supports, we first identity which pages are related to the task. For example, in OpenTable we find 8 pages related to the task "making a restaurant reservation": the page for searching restaurants, for browsing search results, for viewing a restaurant's profile, for submitting a reservation, etc. Then, using our Action Extractor tool, we enumerate all actions present in each task-related page.

[4] pypi.org/project/fuzzywuzzy/

For each action the extractor provides a name, parameters and parameter values, if any. The action names are inferred from various DOM attributes (aria-label, value, placeholder, etc.) and text associated with the relevant DOM element. The goal of the Action Extractor is to label UI elements as humans see them. For example, the search box in the OpenTable website, (instead of being called "search input") is called "Location, Restaurant or Cuisine", which is in fact the placeholder text associated with that input, and what users see in the UI. Parameter values are scraped automatically from DOM select elements (e.g., option value tag). We manually inspect the output of the Action Extractor and correct possible errors (e.g., missing actions). However, for every website, we obtain a different action/parameter scheme. There is no generalized mapping between similar actions/parameters across websites as building such mapping would require significant manual effort. Table 1 reports number of pages, actions and parameters extracted for all websites used in our experiments.

With this data, we construct <page_name, action_name, [parameter_name]> triplets for all actions across all websites, and we ask two annotators to write multiple command templates corresponding to each triplet with parameter names as placeholders. A command template may be "*Book a table for <time>*". For closed-domain parameters, the Action Extractor automatically scrapes their values from webpages (e.g., {12:00 pm, 12:15 pm, etc.} for the *time* parameter), and we ask annotators to provide paraphrases for them (e.g., "at noon"). For open-domain parameters, we ask annotators to provide example values (e.g., "pizza" for a restaurant search term). We assemble the final dataset by instantiating command templates with randomly-chosen parameter value paraphrases, and then split it into train, validation and test datasets. Overall, we generate a total of 53,520 command and navigation instruction pairs. We use train and validation splits for *opentable.com*, *hotels.com* and *rei.com* for model training. Table 1 summarizes the sizes of the train, validation and test splits for all websites.

The second dataset, DialQueries, consists of real user queries extracted from the SGD dialogue dataset (Datasets, 2020) and from Restaurants, Hotels and Shopping "pre-built agents" of Dialogflow (dialogflow.com). We extract queries that are mappable to our website's tasks and adapt them by replacing out-of-vocabulary mentions of restaurants, hotels, cities, etc. with equivalent entities from our vocabulary. We manually map 421, 155 and 63 dialogue queries into navigation instructions for opentable.com, hotels.com, rei.com, respectively. We use this dataset only for evaluation purposes.

Table 1: WebNav dataset statistics. In the last column, -/-/x denotes the website is only used for evaluation, not for training, and X is the number of commands used.

Website (Domain)	# Pg	# Act	# Par	Train / Valid / Test
opentable.com (R)	8	26	38	14332 / 2865 / 1911
yelp.com (R)	8	14	25	- / - / 993
bookatable.co.uk (R)	7	14	19	- / - / 587
hotels.com (H)	7	25	46	15693 / 3137 / 1240
hyatt.com (H)	8	17	48	- / - / 1150
radissonhotels.com (H)	7	20	42	- / - / 1104
rei.com (S)	11	25	40	7001 / 1399 / 933
ebay.com (S)	9	24	38	- / - / 556
macys.com (S)	11	26	37	- / - / 619

Training details. All hyper-parameters are tuned on the validation set. Batch-size is 50. Number of training epochs for action scoring is 7, for parameter mention extraction is 3, and for parameter value scoring is 22. One negative example is sampled for Q^- (in Eq. 3) in every epoch. Dropout is 0.1. Hidden units and embedding size are 300. Learning rate is 1e-4. Regularization parameter is 0.001, $\rho = 0.67$ and $\alpha = 0.4$ (§4.4). The Adam optimizer (Kingma and Ba, 2014) is used for optimization. We use a Tesla P100 GPU and TensorFlow for implementation.

Compared models. There is no direct baseline for this work as related approaches differ in the type of output or problem settings. As discussed in §2, Pasupat et al. (2018) do not perform parameter recognition and value assignments. Liu et al. (2018a) require a reward function for neural policy learning. Joint intent detection and slot filling models perform multi-class classification and do not consider the current state (current webpage), thus not being able to deal with similar intents in different webpages; further, they perform slot filling (equivalent to parameter mention extraction) but do not perform parameter value assignments, thus being unable to output executable paths for web navigation.

Nonetheless, we compare FLIN against two of its variants that use its match-based semantic parsing approach but with the following differences: (i) **FLIN-sem** uses only word-level and character-level semantic similarity for parameter value scoring (no lexical similarity); and (ii) **FLIN-lex** uses

Table 2: In-website and cross-website performance comparison of FLIN variants. WebNav is used for training and/or testing, as specified. All metric scores are scaled out of 1.0.

	A-acc	P-F1	EMA	PA-100	A-acc	P-F1	EMA	PA-100	A-acc	P-F1	EMA	PA-100
	R: opentable.com (training website)				R: yelp.com				R: bookatable.co.uk			
FLIN-sem	0.935	0.499	0.306	0.310	**0.949**	0.483	0.321	0.380	**0.954**	0.415	0.318	0.339
FLIN-lex	0.618	0.496	0.272	0.582	0.493	0.423	0.243	0.488	0.654	0.348	0.235	0.362
FLIN	**0.937**	**0.815**	**0.679**	**0.756**	0.926	**0.836**	**0.703**	**0.824**	0.732	**0.639**	**0.543**	**0.603**
	H: hotels.com (training website)				H: hyatt.com				H: radissonhotels.com			
FLIN-sem	0.933	0.749	0.423	0.468	**0.806**	0.505	0.154	0.286	**0.883**	**0.630**	**0.221**	0.399
FLIN-lex	0.859	0.720	0.311	0.811	0.675	0.467	0.101	0.536	0.364	0.250	0.067	0.325
FLIN	**0.939**	**0.874**	**0.643**	**0.869**	0.740	**0.551**	**0.187**	**0.570**	0.455	0.353	0.146	**0.413**
	S: rei.com (training website)				S: ebay.com				S:macys.com			
FLIN-sem	0.852	0.786	**0.668**	0.704	0.559	0.414	0.239	0.298	0.833	0.526	0.119	0.218
FLIN-lex	0.875	0.781	0.530	**0.828**	**0.902**	0.721	0.248	0.437	0.873	**0.604**	0.166	**0.411**
FLIN	**0.913**	**0.826**	**0.668**	0.782	0.897	**0.729**	**0.327**	**0.453**	**0.878**	0.598	**0.176**	0.358

only lexical similarity in parameter value scoring.

Evaluation metrics. We use accuracy (**A-acc**) to evaluate action prediction and average F1 score (**P-F1**) to evaluate parameter prediction performance. P-F1 is computed using the average *parameter precision* and *parameter recall* over test commands[5]. Given a command, parameter precision is computed as the fraction of parameters in the predicted instruction which are correct and parameter recall as the fraction of parameters in the gold instruction which are predicted correctly. If the predicted action is incorrect or no action has been predicted for a given test command, we consider both parameter precision and recall to be 0 for the command.

We also compute (i) *Exact Match Accuracy* (**EMA**), defined as the percentage of test commands where the predicted instruction exactly matches the gold navigation instruction, and (ii) 100% Precision Accuracy (**PA-100**), defined as the percentage of test commands for which the parameter precision is 1.0 and the predicted action is correct, but parameter recall $\leq$ 1.0. [6] Similar to parameter precision and recall, while computing EMA and PA-100 for test commands, if the predicted action is incorrect or no action has been predicted for a test command, we consider both the exact match and PA-100 value to be 0 for that command.

[5]The P-F1 (parameter F1 score) is computed as the "harmonic mean" of average parameter precision and average parameter recall (averaged over all test queries).

[6]Although we formulate the mapping problem as a ranking one, we do not consider standard metrics such as mean average precision (MAP) or normalized discounted cumulative gain (NDCG) because FLIN outputs only one navigation instruction (instead of a ranked list), given that in a real web navigation system only one predicted action can be executed.

5.2 Performance Results

Table 2 reports the performance comparison of FLIN on the WebNav dataset. We evaluate both in-website (model trained and tested on the same website, 2nd–5th columns) and cross-website (model trained on one website and tested on a different one, 6th–13th columns) performance. FLIN and its two variants adapt relatively well to previously-unseen websites thanks to FLIN's match-based semantic parsing approach. FLIN achieves best overall performance, and is able to adapt to new websites by achieving comparable (or higher) action accuracy (A-acc) and parameter F1 (P-F1) score. Considering PA-100, in the Restaurants domain, 75.6% of commands in the training website (OpenTable), and 60.3% (bookatable) and 82.4% (yelp) of commands in the two test websites are mapped into correct and executable actions (no wrong predictions). PA-100 is generally high also for the other two domains. EMA is lower than PA-100, as it is much harder to predict *all* parameter value assignments correctly. Regarding the FLIN variants, both FLIN-sem and FLIN-lex generally perform worse than FLIN because by combining both lexical and semantic similarity FLIN can be more accurate in doing parameter value assignments and generalize better.

From a generalizability point of view, the most challenging domain is Hotels. While the performance of action prediction (A-acc) for Hotels is in the 45.5%–93.9% range, EMA is in the 14.6%–64.3% range. The drop is mainly due to commands with many parameters (e.g., check-in date, check-out date, number of rooms, etc.), definitely more than in the queries for the other two domains. Shopping is more challenging than Restaurants because

Table 3: Performance on the DialQueries dataset (FLIN models trained on the WebNav dataset).

	A-acc	P-F1	EMA	PA-100
opentable.com (R)	0.719	0.582	0.565	0.584
hotels.com (H)	0.730	0.514	0.381	0.500
rei.com (S)	0.507	0.464	0.428	0.460

while in the Restaurants domain, FLIN must deal with actions/parameters that relate to the *same* entity type (restaurant) with a relatively-contained vocabulary, shopping products can range so widely to be effectively *different entity types* with a diverse set of actions/parameters that can vary significantly across websites. For a more detailed discussion of both aspects see §5.3.

We also test FLIN on the real user queries of the DialQueries dataset available for three websites. As Table 3 shows, despite FLIN not being trained on DialQueries, overall its A-acc is above 50% and its PA-100 is above 46% which demonstrate the robustness of FLIN in the face of new commands.

5.3 Error Analysis

We randomly sampled 135 wrongly-predicted WebNav commands (15 for each of the 9 websites), and classified them into 5 error types (see Table 4).

Overall, 13% of the failures were cases in which an action was not predicted (e.g., for the command "*only eight options*" with ground truth "*filter by size*{'size'='8'}", no action was predicted). 29% of the failures were action miss-predictions (the predicted action did not match the gold action) mainly caused by multiple actions in the given webpage having overlapping semantics. E.g., the command "*options for new york for just 6 people and 2 kids*" got mapped to "*select hot destination*{'destination'= 'new york'}" instead of "*find hotel*{'adults'='6'; 'children'='2'; 'destination'='new york'}". Similarly, "*apply the kids' shoes filter*" got mapped to "*filter by gender*{'gender'= 'kids'}" instead of "*filter by category*{'category': 'kids footwear'}".

Together with action miss-predictions, failures in identifying closed-domain parameters (third row in Table 4) were the most common, especially in hotels and shopping websites. This is because these in-domain websites tend to have a more diverse action and parameter space than that for restaurant websites, thus leading to action and parameter types that were not observed in training data. For example, the search action in Hyatt has *special rates* and *use points* parameters, not present in that of Hotels.com (training site); or eBay has an action *filter by style* not present in Rei (training site).

Table 4: Error analysis based on 135 test commands from WebNav. Columns do not sum up to 100% as multiple parameters in the same command may be problematic for different reasons.

Error Type	%(R)	%(H)	%(S)	%(all)
Action not predicted	17.7	8.9	13.3	13.3
Action miss-predicted	17.7	28.8	40.0	28.9
Failed to identify closed-domain parameter	20.0	57.7	31.1	36.3
Closed-domain parameter value miss-predicted	31.1	6.7	4.4	14.1
Fail to extract open-domain parameter value	15.5	11.1	24.4	17.0

Failures in predicting the value of a correctly-identified closed-domain parameter mention (forth row in the table) were often due to morphological variations in parameter values not frequently observed in training (e.g. "*8:00 in the evening*" got mapped to the value '*18:00*' instead of '*20:00*').

Errors in extracting open-domain parameters were due to parameter names too generic (e.g. "*search keyword*"), extracted mentions partially matching gold mentions (e.g., "*hyatt*" vs. the gold mention "*hyatt regency grand cypress*"), or multiple formats of the parameter values (e.g., various formats for *phone number* or *zip code*).

6 Conclusion

To generalize to many websites, NL-guided web navigation assistants require an NL interface that can work with new website UIs without being re-trained each time. To this end, we proposed FLIN, a matching-based semantic parsing approach that maps user commands to concept-level actions. While various optimizations are possible, FLIN adapted well to new websites and delivered good performance. We have used it in restaurant, shopping and hotels websites, but its design can apply to more domains.

Acknowledgments

We would like to thank Jason Kace (Microsoft) for many constructive discussions and for designing and implementing the *Action Extractor* and *Action Executor* modules used by the NL-guided web navigation system.

References

James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. PLOW: A Collaborative Task Learning Agent. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pages 1514–1519. AAAI Press.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, pages 49–62.

Shaon Barman, Sarah Chasins, Rastislav Bodik, and Sumit Gulwani. 2016. Ringer: Web Automation by Demonstration. In *Proc. f the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2016, pages 748–764. ACM.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement Learning for Mapping Instructions to Actions. In *Proc. of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 82–90, USA. Association for Computational Linguistics.

Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020a. Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning. In *Proc. of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, pages 322–334.

Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. 2020b. Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination? In *Proc. of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE'20)*, ESEC/FSE 2020, pages 1202–1214, New York, NY. ACM.

Qian Chen, Zhu Zhuo, and W. Wang. 2019. BERT for Joint Intent Classification and Slot Filling. *ArXiv*, abs/1902.10909.

Google Research Datasets. 2020. The schema-guided dialogue dataset. https://github.com/google-research-datasets/dstc8-schema-guided-dialogue.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186. Association for Computational Linguistics.

Mattia Fazzini, Martin Prammer, Marcelo d'Amorim, and Alessandro Orso. 2018. Automatically Translating Bug Reports into Test Cases for Mobile Apps. In *Proc. of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, page 141–152, New York, NY, USA. Association for Computing Machinery.

Sébastien Ferré. 2017. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web*, 8(3):405–418.

Michael C Frank and Noah D Goodman. 2012. Predicting pragmatic reasoning in language games. *Science*, 336(6084):998–998.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified Pragmatic Models for Generating and Following Instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 1951–1963. Association for Computational Linguistics.

Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G. Karahalios. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 489–500. Association for Computing Machinery.

Francisco Javier Chiyah Garcia, David A. Robb, Xingkun Liu, Atanas Laskov, Pedro Patrón, and Helen F. Hastie. 2018. Explain Yourself: A Natural Language Interface for Scrutable Autonomous Robots. *CoRR*, abs/1803.02088.

Dave Golland, Percy Liang, and Dan Klein. 2010. A Game-Theoretic Approach to Generating Spatial Descriptions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 410–419, USA. Association for Computational Linguistics.

D. Guo, G. Tur, W. Yih, and G. Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 554–559.

Kelvin Guu, Panupong Pasupat, Evan Liu, and Percy Liang. 2017. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1051–1062, Vancouver, Canada. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Michael Janner, Karthik Narasimhan, and Regina Barzilay. 2018. Representation learning for grounded spatial reasoning. *Transactions of the Association of Computational Linguistics*, pages 49–61.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tessa Lau, Clemens Drews, and Jeffrey Nichols. 2009. Interpreting Written How-to Instructions. In *Proc. of the 21st International Jont Conference on Artifical Intelligence*, IJCAI '09, pages 1433–1438, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Carolin Lawrence and Stefan Riezler. 2016. NLmaps: A natural language interface to query OpenStreetMap. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 6–10.

Bing Liu and Ian Lane. 2016. Joint Online Spoken Language Understanding and Language Modeling With Recurrent Neural Networks. In *Proc. of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 22–30. Association for Computational Linguistics.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018a. Reinforcement Learning on Web Interfaces using Workflow-Guided Exploration. In *6th International Conference on Learning Representations, ICLR 2018*.

Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018b. Learning Design Semantics for Mobile Apps. In *Proc. of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pages 569–579, New York, NY, USA. Association for Computing Machinery.

Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI. In *Proc. of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, pages 248–259. IEEE Press.

Panupong Pasupat, Tian-Shun Jiang, Evan Liu, Kelvin Guu, and Percy Liang. 2018. Mapping natural language commands to web elements. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4970–4976. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Selenium. 2020. https://www.selenium.dev.

Vidya Setlur, Sarah E. Battersby, Melanie Tory, Rich Gossweiler, and Angel X. Chang. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, page 365–377. Association for Computing Machinery.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017a. World of Bits: An Open-Domain Platform for Web-Based Agents. In *Proc. of the 34th International Conference on Machine Learning*, volume 70, pages 3135–3144.

Tianlin Tim Shi, Andrej Karpathy, Linxi Jim Fan, Jonathan Hernandez, and Percy Liang. 2017b. World of bits: An open-domain platform for web-based agents. In *Proc. of the 34th International Conference on Machine Learning-Volume 70*, pages 3135–3144. JMLR. org.

Nathaniel J Smith, Noah Goodman, and Michael Frank. 2013. Learning and using language via recursive pragmatic reasoning about other agents. In *Advances in neural information processing systems*, pages 3039–3047.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to web APIs. In *Proc. of the 2017 ACM on Conference on Information and Knowledge Management*, pages 177–186. ACM.

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 *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 855–864.

Tech Crunch. 2019. Google is bringing AI assistant Duplex to the web. https://techcrunch.com/2019/05/07/google-is-bringing-ai-assistant-duplex-to-the-web/.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 1507–1514. AAAI Press.

Suresh Thummalapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra. 2012. Automating Test Automation. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, page 881–891. IEEE Press.

Prasetya Utama, Nathaniel Weir, Fuat Basik, Carsten Binnig, Ugur Cetintemel, Benjamin Hättasch, Amir Ilkhechi, Shekar Ramaswamy, and Arif Usta. 2018. An end-to-end neural natural language interface for databases. *arXiv preprint arXiv:1804.00401*.

Sida I. Wang, Percy Liang, and Christopher D. Manning. 2016. Learning Language Games through Interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2368–2378. Association for Computational Linguistics.

Kyle Williams, Seyyed Hadi Hashemi, and Imed Zitouni. 2019. Automatic Task Completion Flows from Web APIs. In *Proc. of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1009–1012.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

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 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke Zettlemoyer and Michael Collins. 2007. Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687. Association for Computational Linguistics.

Yu Zhao, Tingting Yu, Ting Su, Yang Liu, Wei Zheng, Jingzhi Zhang, and William G. J. Halfond. 2019. ReCDroid: Automatically Reproducing Android Application Crashes from Bug Reports. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 128–139. IEEE Press.