Mapping natural language commands to web elements

Panupong Pasupat **Tian-Shun Jiang** **Evan Liu** **Kelvin Guu** **Percy Liang**
Stanford University
{ppasupat,jiangts,evanliu,kguu,pliang}@cs.stanford.edu

Abstract

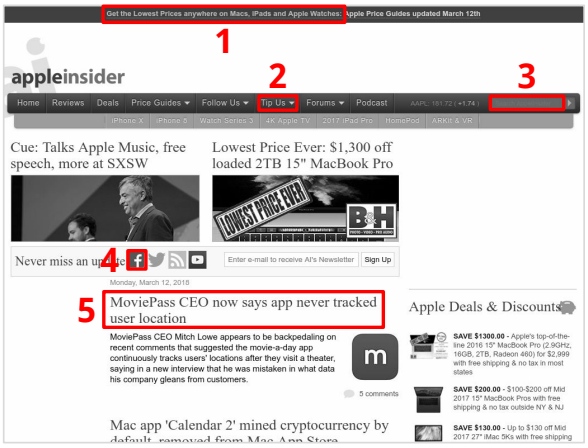
The web provides a rich, open-domain environment with textual, structural, and spatial properties. We propose a new task for grounding language in this environment: given a natural language command (e.g., "click on the second article"), choose the correct element on the web page (e.g., a hyperlink or text box). We collected a dataset of over 50,000 commands that capture various phenomena such as functional references (e.g. "find who made this site"), relational reasoning (e.g. "article by john"), and visual reasoning (e.g. "top-most article"). We also implemented and analyzed three baseline models that capture different phenomena present in the dataset.

1 Introduction

Web pages are complex documents containing both structured properties (e.g., the internal tree representation) and unstructured properties (e.g., text and images). Due to their diversity in content and design, web pages provide a rich environment for natural language grounding tasks.

In particular, we consider the task of mapping natural language commands to web page elements (e.g., links, buttons, and form inputs), as illustrated in Figure 1. While some commands refer to an element's text directly, many others require more complex reasoning with the various aspects of web pages: the text, attributes, styles, structural data from the document object model (DOM), and spatial data from the rendered web page.

Our task is inspired by the semantic parsing literature, which aims to map natural language utterances into actions such as database queries and object manipulation (Zelle and Mooney, 1996; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Berant et al., 2013; Misra et al., 2015; Andreas and Klein, 2015). While these actions usually act on an environment with a fixed and known schema,



1: click on apple deals
2: send them a tip
3: enter iphone 7 into search
4: follow on facebook
5: open most recent news update

Figure 1: Examples of natural language commands on the web page `appleinsider.com`.

web pages contain a larger variety of structures, making the task more open-ended. At the same time, our task can be viewed as a reference game (Golland et al., 2010; Smith et al., 2013; Andreas and Klein, 2016), where the system has to select an object given a natural language reference. The diversity of attributes in web page elements, along with the need to use context to interpret elements, makes web pages particularly interesting.

Identifying elements via natural language has several real-world applications. The main one is providing a voice interface for interacting with web pages, which is especially useful as an assistive technology for the visually impaired (Zajicek et al., 1998; Ashok et al., 2014). Another use case is browser automation: natural language commands are less brittle than CSS or XPath selectors (Hammoudi et al., 2016) and could generalize across different websites.

We collected a dataset of over 50,000 natural language commands. As seen in Figure 1, the

Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 4970–4976
Brussels, Belgium, October 31 - November 4, 2018.

Phenomenon	Description	Example	Amount
substring match	The command contains only a substring of the element's text (after stemming).	"view internships with energy.gov" → "Careers & Internship" link	7.0 %
paraphrase	The command paraphrases the element's text.	"click sign in" → "Login" link	15.5 %
goal description	The command describes an action or asks a question.	"change language" → a clickable box with text "English"	18.0 %
summarization	The command summarizes the text in the element.	"go to the article about the bengals trade" → the article title link	1.5 %
element description	The command describes a property of the element.	"click blue button"	2.0 %
relational reasoning	The command requires reasoning with another element or its surrounding context.	"show cookies info" → "More Info" in the cookies warning bar, not in the news section	2.5 %
ordinal reasoning	The command uses an ordinal.	"click on the first article"	3.5 %
spatial reasoning	The command describes the element's position.	"click the three slashes at the top left of the page"	2.0 %
image target	The target is an image (no text).	"select the favorites button"	11.5 %
form input target	The target is an input (text box, check box, drop-down list, etc.).	"in the search bar, type testing"	6.5 %

Table 1: Phenomena present in the commands in the dataset. Each example can have multiple phenomena.

commands contain many phenomena, such as relational, visual, and functional reasoning, which we analyze in greater depth in Section 2.2. We also implemented three models for the task based on retrieval, element embedding, and text alignment. Our experimental analysis shows that functional references, relational references, and visual reasoning are important for correctly identifying elements from natural language commands.

2 Task

Given a web page w with elements $e_1, \ldots, e_k$ and a command c, the task is to select the element $e \in \{e_1, \ldots, e_k\}$ described by the command c. The training and test data contain (w, c, e) triples.

2.1 Dataset

We collected a dataset of 51,663 commands on 1,835 web pages. To collect the data, we first archived home pages of the top 10,000 websites[1] by rendering them in Google Chrome. After loading the dynamic content, we recorded the DOM trees and the geometry of each element, and stored the rendered web pages. We filtered for web pages in English that rendered correctly and did not contain inappropriate content. Then we asked crowdworkers to brainstorm different actions for each web page, requiring each action to reference exactly one element (of their choice) from the filtered list of interactive elements (which include visible links, inputs, and buttons). We encouraged the workers to avoid using the exact words of the elements by granting a bonus for each command that did not contain the exact wording of the selected element. Finally, we split the data into 70% training, 10% development, and 20% test data. Web pages in the three sets do not overlap.

The collected web pages have an average of 1,051 elements, while the commands are 4.1 tokens long on average.

2.2 Phenomena present in the commands

Apart from referring to the exact text of the element, commands can refer to elements in a variety of ways. We analyzed 200 examples from the training data and broke down the phenomena present in these commands (see Table 1).

Even when the command directly references the element's text, many other elements on the page also have word overlap with the command. On average, commands have word overlap with 5.9 leaf elements on the page (not counting stop words).

3 Models

3.1 Retrieval-based model

Many commands refer to the elements by their text contents. As such, we first consider a simple retrieval-based model that uses the command as a search query to retrieve the most relevant element based on its TF-IDF score.

Specifically, each element is represented as a bag-of-tokens computed by (1) tokenizing and stemming its text content, and (2) tokenizing the

[1] https://majestic.com/reports/majestic-million

attributes (id, class, placeholder, label, tooltip, aria-text, name, src, href) at punctuation marks and camel-case boundaries. When computing term frequencies, we downweight the attribute tokens from (2) by a factor of α. We use $\alpha = 3$ tuned on the development set for our experiments.

The document frequencies are computed over the web pages in the training dataset. If multiple elements have the same score, we heuristically pick the most prominent element, i.e., the one that appears earliest in the pre-order traversal of the DOM hierarchy.

3.2 Embedding-based model

A common method for matching two pieces of text is to embed them separately and then compute a score from the two embeddings (Kiros et al., 2015; Tai et al., 2015). For a command c and elements $e_1, \ldots, e_k$, we define the following conditional distribution over the elements:

$$p\left(e_i \mid c\right) \propto \exp\left[s(f(c), g(e_i))\right]$$

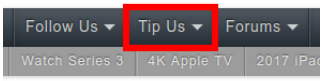
where s is a scoring function, $f(c)$ is the embedding of c, and $g(e_i)$ is the embedding of e_i, described below. The model is trained to maximize the log-likelihood of the correct element in the training data.

Command embedding. To compute $f(c)$, we embed each token of c into a fixed-dimensional vector and take an average[2] over the token embeddings. (The token embeddings are initialized with GloVe vectors.)

Element embedding. To compute $g(e)$, we embed the properties of e, concatenate the results, and then apply a linear layer to obtain a vector of the same length as $f(c)$. Figure 2 shows an example of the properties that the model receives. The properties include:

- *Text content.* We apply the command embedder f on the text content of e. As the text of most elements of interest (links, buttons, and inputs) are short, we find it sufficient to limit the text to the first 10 tokens to save memory.
- *Text attributes.* Several attributes (aria, title, tooltip, placeholder, label, name) usually contain natural language. We concatenate their values and then apply the command embedder f on the resulting string.

[2]We tried applying LSTM but found no improvement.




```
<a class="dd-head" id="tip-link"
href="submit_story/">Tip Us</a>
```

Text content:	tip us
String attributes:	tip link dd head
Visual features:	location = (0.53, 0.08)
	visible = true

Figure 2: Example of properties used to compute the embedding $g(e)$ of the element e.

- *String attributes.* We tokenize other string attributes (tag, id, class) at punctuation marks and camel-case boundaries. Then we embed them with separate lookup tables and average the resulting vectors.
- *Visual features.* We form a vector consisting of the coordinates of the element's center (as fractions of the page width and height) and visibility (as a boolean).

Scoring function. To compute the score $s(f(c), g(e))$, we first let $\hat{f}(c)$ and $\hat{g}(e)$ be the results of normalizing the two embeddings to unit norm. Then we apply a linear layer on the concatenated vector $[\hat{f}(c); \hat{g}(e); \hat{f}(c) \circ \hat{g}(e)]$ (where $\circ$ denotes the element-wise product).

Incorporating spatial context. Context is critical in certain cases; for example, selecting a text box relies on knowing the neighboring label text, and selecting an article based on the author requires locating the author's name nearby. Identifying which related element should be considered based on the command is a challenging task.

We experiment with adding *spatial context* to the model. For each direction $d \in \{\text{top, bottom, left, right}\}$, we use g to embed a neighboring element $n_d(e)$ directly adjacent to e in that direction. (If there are multiple such elements, sample one; if there is no such element, use a zero vector.) After normalizing the results to get $\hat{g}(n_d(e))$, we concatenate $\hat{g}(n_d(e))$ and $\hat{f}(c) \circ \hat{g}(n_d(e))$ to the linear layer input in the scoring function.

3.3 Alignment-based model

One downside of the embedding-based model is that the text tokens from c and e do not directly interact. Previous works on sentence matching usually employ either unidirectional or bidirectional attention to tackle this issue (Seo et al., 2016; Yin

et al., 2016; Xiong et al., 2017; Yu et al., 2018). We opt for a simple method based on a single alignment matrix similar to Hu et al. (2014) as described below.

Let $t(e)$ be the concatenation of e's text content and text attributes of e, trimmed to 10 tokens. We construct a matrix $A(c, e)$ where each entry $A_{ij}(c, e)$ is the dot product between the embeddings of the ith token of c and the jth token of $t(e)$. Then we apply two convolutional layers of size 3×3 on the matrix, apply a max-pooling layer of size 2×2, concatenate a tag embedding, and then apply a linear layer on the result to get a 10-dimensional vector $h(c, e)$.

We apply a final linear layer on $h(c, e)$ to compute a scalar score, and then train on the same objective function as the encoding-based model. To incorporate context, we simply concatenate the four vectors $h(c, n_d(e))$ of the neighbors $n_d(e)$ to the final linear layer input.

4 Experiments

We evaluate the models on *accuracy*, the fraction of examples that the model selects the correct element. We train the neural models using Adam (Kingma and Ba, 2014) with initial learning rate 10^{-3}, and apply early stopping based on the development set. The models can choose any element that is visible on the page at rendering time.

The experimental results are shown in Table 2. Both neural models significantly outperform the retrieval model.

4.1 Ablation analysis

To measure the importance of each type of information in web pages, we perform an ablation study where the model does not observe one of the following aspects of the elements: text contents, attributes, and spatial context.

Unsurprisingly, the results in Table 2 show that text contents are the most important input signal. However, attributes also play an important role in both the embedding and alignment models. Finally, while spatial context increases alignment model performance, the gain is very small, suggesting that incorporating appropriate contexts to the model is a challenging task due to the variety in the types of context, as well as the sparsity of the signals.

Model	Accuracy (%)
retrieval	36.55
embedding	56.05
no texts	23.62
no attributes	55.43
no spatial context	58.87
alignment	50.74
no texts	15.94
no attributes	48.51
no spatial context	50.66

Table 2: Accuracies of the models and their ablations.

Error Type	Embed	Align
Fail to match strings	26.8%	11.6%
Incorrectly match strings	3.8%	14.2%
Fail to understand paraphrases	8.9%	7.9%
Fail to understand descriptions	12.1%	17.4%
Fail to perform reasoning	15.9%	13.7 %
Select a less prominent element	19.8%	24.8%
Noisy annotation	12.7%	10.5%

Table 3: Error breakdowns of the embedding and alignment models on 100 examples. The embedding model handles implicit descriptions well, while the alignment model excels at string matching.

4.2 Error analysis

To get a better picture of how the models handle different phenomena, we analyze the predictions of the embedding-based and alignment-based models on 100 development examples where at least one model made an error. The errors, summarized in Table 3, are elaborated below:

Fail to match strings. Many commands simply specify the text content of the element (e.g., "click customised garages" $\rightarrow$ the link with text "Customised Garages, Canopies & Carports"). The encoding model, which encodes the whole command as a single vector, occasionally fails to select the element with partially matching texts. In contrast, the alignment model explicitly models text matching, and thus is better at this type of commands.

Incorrectly match strings. Due to its reliance on text matching, the alignment model struggles when many elements share substrings with the command (e.g., "shop for knitwear" when many elements contain the word "shop"), or when an element with a matching substring is not the correct target (e.g., "get the program" $\rightarrow$ the "Download" link, not the "Microsoft developer program" link).

Fail to understand descriptions. As seen in Table 1, many commands indirectly describe the elements using paraphrases, goal descriptions, or properties of the elements. The encoding model,

which summarizes various properties of the elements as an embedding vector, is better at handling these commands than the alignment model, but still makes a few errors on harder examples (e.g., "please close this notice for me" → the "X" button with hidden text "Hide").

Fail to perform reasoning. For the most part, the models fail to handle relational, ordinal, or spatial reasoning. The most frequent error mode is when the element is a text box, and the command uses nearby label as the reference. While a few text boxes have semantic annotations which the model can use (e.g., tooltip or aria attributes), many web pages do not provide such annotations. To handle these cases, a model would have to identify the label of the text box based on logical or visual contexts.

Other errors. Apart from the annotation noise, occasionally multiple elements on the web page satisfy the given command (e.g., "log in" can match any "Sign In" button on the web page). In these cases, the annotation usually gives the most *prominent* element among the possible candidates. To provide a natural interface for users, the model should arguably learn to predict such prominent elements instead of more obscure ones.

5 Related work and discussion

Mapping natural language to actions. Previous work on semantic parsing learns to perform actions described by natural language utterances in various environments. Examples of such actions include API calls (Young et al., 2013; Su et al., 2017; Bordes and Weston, 2017), database queries (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2007; Berant et al., 2013; Yih et al., 2015), navigation (Artzi and Zettlemoyer, 2013; Janner et al., 2018), and object manipulation (Tellex et al., 2011; Andreas and Klein, 2015; Guu et al., 2017; Fried et al., 2018).

For web pages and graphical user interfaces, there are previous works on using natural language to perform computations on web tables (Pasupat and Liang, 2015; Zhong et al., 2017) and submit web forms (Shi et al., 2017). Our task is similar to previous works on interpreting instructions on user interfaces (Branavan et al., 2009, 2010; Liu et al., 2018). While their works focuses on learning from distant supervision, we consider shallower interactions but on a much broader domain.

Previous work also explores the reverse problem of generating natural language description of objects (Vinyals et al., 2014; Karpathy and Fei-Fei, 2015; Zarriaiß and Schlangen, 2017). We hope that our dataset could also be useful for exploring the reverse task of describing actions on web pages.

Reference games. In a reference game, the system has to select the correct object referenced by the given utterance (Frank and Goodman, 2012). Previous work on reference games focuses on a small number of objects with similar properties, and applies pragmatics to handle ambiguous utterance (Golland et al., 2010; Smith et al., 2013; Çelikyilmaz et al., 2014; Andreas and Klein, 2016; Yu et al., 2017). Our task can be viewed as a reference game with several challenges: higher number of objects, diverse object properties, and the need to interpret objects based on their contexts.

Interacting with web pages. Automated scripts are used to interact with web elements. While most scripts reference elements with logical selectors (e.g., CSS and XPath), there have been several alternatives such as images (Yeh et al., 2009) and simple natural language utterances (Soh, 2017). Some other interfaces for navigating web pages include keystrokes (Spalteholz et al., 2008), speech (Ashok et al., 2014), haptics (Yu et al., 2005), and eye gaze (Kumar et al., 2007).

6 Conclusion

We presented a new task of grounding natural language commands on open-ended and semi-structured web pages. With different methods of referencing elements, mixtures of textual and non-textual element attributes, and the need to properly incorporate context, our task offers a challenging environment for language understanding with great potential for real-world applications.

Our dataset and code are available at `https://github.com/stanfordnlp/phrasenode`. Reproducible experiments are available on the CodaLab platform at `https://worksheets.codalab.org/worksheets/0x0097f249cd944284a81af331093c3579/`.

Acknowledgments

This work was supported by NSF CAREER Award under No. IIS-1552635 and an Amazon Research Award.

References

J. Andreas and D. Klein. 2015. Alignment-based compositional semantics for instruction following. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Andreas and D. Klein. 2016. Reasoning about pragmatics with neural listeners and speakers. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1182.

Y. Artzi and L. Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)*, 1:49–62.

V. Ashok, Y. Borodin, S. Stoyanchev, Y. Puzis, and I. V. Ramakrishnan. 2014. Wizard-of-Oz evaluation of speech-driven web browsing interface for people with vision impairments. In *Web for All Conference*.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

A. Bordes and J. Weston. 2017. Learning end-to-end goal-oriented dialog. In *International Conference on Learning Representations (ICLR)*.

S. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Association for Computational Linguistics and International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 82–90.

S. Branavan, L. Zettlemoyer, and R. Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Association for Computational Linguistics (ACL)*, pages 1268–1277.

A. Çelikyilmaz, Z. Feizollahi, D. Z. Hakkani-Tür, and R. Sarikaya. 2014. Resolving referring expressions in conversational dialogs for natural user interfaces. In *Empirical Methods in Natural Language Processing (EMNLP)*.

D. L. Chen and R. J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 859–865.

M. Frank and N. D. Goodman. 2012. Predicting pragmatic reasoning in language games. *Science*, 336:998–998.

D. Fried, J. Andreas, and D. Klein. 2018. Unified pragmatic models for generating and following instructions. In *North American Association for Computational Linguistics (NAACL)*.

D. Golland, P. Liang, and D. Klein. 2010. A game-theoretic approach to generating spatial descriptions. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 410–419.

K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Association for Computational Linguistics (ACL)*.

M. Hammoudi, G. Rothermel, and P. Tonella. 2016. Why do record/replay tests of web applications break? *IEEE International Conference on Software Testing, Verification and Validation*.

B. Hu, Z. Lu, H. Li, and Q. Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *Advances in Neural Information Processing Systems (NIPS)*.

M. Janner, K. Narasimhan, and R. Barzilay. 2018. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics (TACL)*, 6.

A. Karpathy and L. Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3128–3137.

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, R. Urtasun, A. Torralba, and S. Fidler. 2015. Skip-thought vectors. In *Advances in Neural Information Processing Systems (NIPS)*.

M. Kumar, A. Paepcke, and T. Winograd. 2007. Eyepoint: practical pointing and selection using gaze and keyboard. In *Conference on Human Factors in Computing Systems (CHI)*.

E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*.

D. K. Misra, K. Tao, P. Liang, and A. Saxena. 2015. Environment-driven lexicon induction for high-level instructions. In *Association for Computational Linguistics (ACL)*.

P. Pasupat and P. Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Association for Computational Linguistics (ACL)*.

M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv*.

T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning (ICML)*.

N. J. Smith, N. D. Goodman, and M. C. Frank. 2013. Learning and using language via recursive pragmatic reasoning about other agents. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3039–3047.

K. Soh. 2017. TagUI: RPA / CLI tool for automating user interactions. https://github.com/kelaberetiv/TagUI.

L. Spalteholz, K. F. Li, N. Livingston, and F. Hamidi. 2008. Keysurf: a character controlled browser for people with physical disabilities. In *World Wide Web (WWW)*.

Y. Su, A. H. Awadallah, M. Khabsa, P. Pantel, M. Gamon, and M. J. Encarnación. 2017. Building natural language interfaces to web apis. In *Conference on Information and Knowledge Management (CIKM)*.

K. S. Tai, R. Socher, and C. D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Association for Computational Linguistics (ACL)*.

S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2014. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*.

C. Xiong, V. Zhong, and R. Socher. 2017. Dynamic coattention networks for question answering. In *International Conference on Learning Representations (ICLR)*.

T. Yeh, T. Chang, and R. Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *User Interface Software and Technology (UIST)*.

W. Yih, M. Chang, X. He, and J. Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Association for Computational Linguistics (ACL)*.

W. Yin, H. Schütze, B. Xiang, and B. Zhou. 2016. ABCNN: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics (TACL)*, 4.

S. Young, M. Gašić, B. Thomson, and J. D. Williams. 2013. POMDP-based statistical spoken dialog systems: A review. In *Proceedings of the IEEE*, 5, pages 1160–1179.

A. W. Yu, D. Dohan, M. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le. 2018. QANet: Combining local convolution with global self-attention for reading comprehension. In *International Conference on Learning Representations (ICLR)*.

L. Yu, H. Tan, M. Bansal, and T. L. Berg. 2017. A joint speaker-listener-reinforcer model for referring expressions. In *Computer Vision and Pattern Recognition (CVPR)*.

W. Yu, R. Kuber, E. Murphy, P. Strain, and G. McAllister. 2005. A novel multimodal interface for improving visually impaired people's web accessibility. *Virtual Reality*, 9.

M. Zajicek, C. Powell, and C. Reeves. 1998. A web navigation tool for the blind. In *International ACM Conference on Assistive Technologies*.

S. Zarriaiß and D. Schlangen. 2017. Obtaining referential word meanings from visual and distributional information: Experiments on object naming. In *Association for Computational Linguistics (ACL)*.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*, pages 678–687.

V. Zhong, C. Xiong, and R. Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.