Grounding Open-Domain Instructions to Automate Web Support Tasks

Nancy Xu
Computer Science Dept.
Stanford University
xnancy@stanford.edu

Sam Masling
Computer Science Dept.
Stanford University
smasling@stanford.edu

Michael Du
Computer Science Dept.
Stanford University
mdu7@stanford.edu

Giovanni Campagna
Computer Science Dept.
Stanford University
gcampagn@stanford.edu

Larry Heck
Viv Labs
Samsung Research
larry.heck@ieee.org

James Landay
Computer Science Dept.
Stanford University
landay@stanford.edu

Monica S Lam
Computer Science Dept.
Stanford University
lam@stanford.edu

Abstract

Grounding natural language instructions on the web to perform previously unseen tasks enables accessibility and automation. We introduce a task and dataset to train AI agents from open-domain, step-by-step instructions originally written for people. We build RUSS (Rapid Universal Support Service) to tackle this problem. RUSS consists of two models: First, a BERT-LSTM with pointers parses instructions to ThingTalk, a domain-specific language we design for grounding natural language on the web. Then, a grounding model retrieves the unique IDs of any webpage elements requested in ThingTalk. RUSS may interact with the user through a dialogue (e.g. ask for an address) or execute a web operation (e.g. click a button) inside the web runtime. To augment training, we synthesize natural language instructions mapped to ThingTalk. Our dataset consists of 80 different customer service problems from help websites, with a total of 741 step-by-step instructions and their corresponding actions. RUSS achieves 76.7% end-to-end accuracy predicting agent actions from single instructions. It outperforms state-of-the-art models that directly map instructions to actions without ThingTalk. Our user study shows that RUSS is preferred by actual users over web navigation.

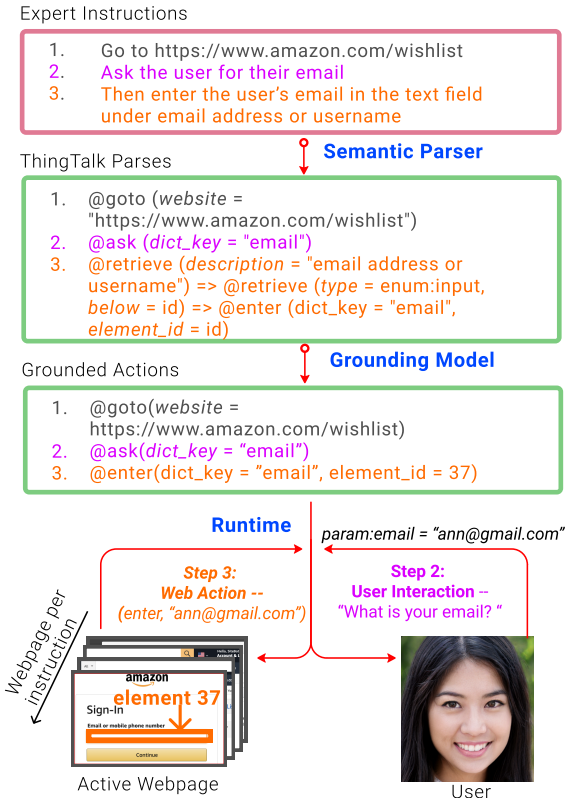

Figure 1: RUSS's semantic parser maps natural language instructions into ThingTalk (our DSL) and uses a grounding model to resolve elements in ThingTalk for grounded actions. The runtime executes the actions.

1 Introduction

Grounding natural language is a key to building robots and AI agents (Chen and Mooney, 2011) that interact seamlessly with people. Besides grounding tasks visually (Mirowski et al., 2018; Venugopalan et al., 2015), future AI agents must be able to ground language and execute actions on the web.

We build a general-purpose, interactive agent to master tasks from open-domain natural language instructions on websites. We focus on the service domain for tasks such as redeeming a gift card, logging out of all your accounts, or resetting a password.

Conversational agents capable of providing universal access to the web through a language interface are an important step towards achieving information equity. These agents empower those who are visually impaired or situationally preoccupied (e.g. driving) to obtain web-based knowledge and services for which they would otherwise require a laptop or mobile device for (Sarsenbayeva, 2018). Already, virtual assistants and call centers demonstrate a large number of scenarios where language interfaces backed by web backends are required by

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1022–1032
June 6–11, 2021.

companies and users. However, unlike virtual assistants, web agents like RUSS are universal, navigating the Web, interacting with users, and bypassing the need for domain-specific APIs.

On average over 60% of Americans have contacted customer service in a month (Statista Research Department, 2019). A call center manager might instruct its agents to do the following to help a customer through a password reset: "*go to passwordreset.com; ask the user for their desired new password; click the reset button*". As the agent performs the instructions on the web behind-the-scenes, the user is read information or asked questions periodically over a conversational interface (such as a phone).

Our approach, RUSS (Figure 1), trains an agent that masters any web task specified from open-domain instructions. To do so, we design a domain-specific language (DSL) for grounding on the web and implement it as a subset of the ThingTalk programming language (Campagna et al., 2019). Each natural language instruction maps to one of six agent actions that interact with users or operate on webpages. Actions that operate on the web are passed element IDs that are retrieved from high-level user language by grounding its corresponding ThingTalk on the active webpage. In the following, we use ThingTalk to refer to our subset taylored to web operations, where not ambiguous. We break down the problem into two components: (1) a semantic parser that takes single-step natural language instructions and maps to ThingTalk statements using a BERT-LSTM pointer network, and (2) a grounding model that takes ThingTalk and retrieves an element ID on the active webpage where needed.

The contributions of this work include:

1. **Task**: The new problem of building an interactive web agent capable of mastering tasks from open-domain natural language instructions.
2. **RUSS**: A fully functioning agent that services user support requests from natural language instructions. RUSS consists of a semantic parser, a grounding model, and a runtime. We release RUSS as an open-source repository [1]
3. **ThingTalk**: A typed DSL that grounds natural language instructions on the web. ThingTalk is designed to be an expressive target for natural language semantic parsing, and amenable to training data synthesis.
4. **RUSS Dataset**: a) Evaluation: a collection of 741 real-world step-by-step natural language instructions (raw and annotated) from the open web, and for each: its corresponding webpage DOM, ground-truth ThingTalk, and ground-truth actions; and b) Synthetic: a synthetic dataset of 1.5M natural language instructions mapped to ThingTalk.
5. **Evaluation of RUSS**: 76.7% accuracy on our RUSS evaluation dataset. Our semantic parser maps natural language instructions to ThingTalk at 85% accuracy and our grounding model achieves 75% accuracy in resolving web element descriptions. A user study of RUSS shows preference of the natural language interface over existing Web UIs.

2 Related Work

Grounding in the Visual and Physical Worlds (Robotics). Grounding language in both the physical world (Chen and Mooney, 2011) and in images and videos ((Venugopalan et al., 2015), (Hendricks et al., 2018)) through systems like visual question-answering (Antol et al., 2015) have been extensively explored. For example, Thomason et al. (2016) describe the game "I Spy" where human and robot take turns describing one object among several in a physical environment, requiring grounding of natural language to the physical world, and robot-human dialogues are explored in (Thomason et al., 2019). Previous work has proposed adaptive language interfaces for robots in dyanmic settings such as (Liu et al., 2018), (Ito et al., 2020; Liu et al., 2018), (Karamcheti et al., 2020), and (Kim et al., 2020). Other work builds physical world agents that operate through sequential actions (Chen and Mooney, 2011; Misra et al., 2017; Mirowski et al., 2018).

Natural Language Digital Interfaces. An intelligent automated software assistant that collaborates with humans to complete tasks was first introduced in (Allen et al., 2007). Since then, identifying UI components from natural language commands has been an important area of research in grounding, with prior work investigating approaches to map natural language instructions to mobile interfaces such as Android (Li et al., 2020) and Adobe photo editing GUIs (Manuvinakurike et al., 2018). Earlier work mapped natural lan-

[1] https://github.com/xnancy/russ

Agent Action	Description
@goto(*url*)	Navigate to the given URL
@enter(*element_id*, *dict_key*)	Find the closest match to the given dictionary key and enter its value in the given input element
@click(*element_id*)	Click on the given element
@read(*element_id*)	Read the content of the given element to the user
@say(*message*)	Read the given message to the user
@ask(*dict_key*)	Ask the user for the value of a dictionary key
Grounding Function	**Description**
@retrieve(*descr*, *type*, *loc*, *above*, *below*, *right_of*, *left_of*) : *element_id*	Retrieves the elements matching the descriptors, returns an *element_id*.

Table 1: WebLang Agent Actions and a Grounding Function

guage commands to web elements on a TV screen through a combination of lexical and gesture intent (Heck et al., 2013). More recently, Pasupat et al. (2018) attempted to map natural language commands written by Amazon Mechanical Turkers to web elements (without actions). Unlike prior research, our work focuses on a new domain of parsing natural language instructions into *executable actions* on the web, where instead of mapping directly to elements using a neural model, we semantically parse natural language instructions to formal actions that support web navigation as well as user interactivity.

Dialogue Agents for The Web. Other web-based dialogue agents are developed through single-use heuristics and more recently through programming-by-demonstration (PBD) tools. This approach allows users and developers to author programs that operate on the web and invoke those programs in natural language (Li et al., 2017; Li and Riva, 2018; Fischer et al., 2020; Sarmah et al., 2020). CoScripter (Leshed et al., 2008) additionally allows the user to edit the demonstration in natural language, and parses a limited natural language into executable form. While related in end goal, our work does not require user demonstration and can operate using existing real-world instructions. We note though that the WebLang intermediate representation and our grounding model can be used to improve the robustness of PBD systems as well.

3 Task and Model

Given a set of natural language instructions $S = (i_1, \ldots, i_n)$ and a starting web page, our task is to construct an agent that follows the instructions through a series of action $A = (a_1, \ldots, a_n)$. Actions include web navigation and end-user interaction in order to obtain necessary information. Surveying online customer service tasks, 6 action operations were identified as necessary for agents: open a URL page, enter text, click on buttons, say something to the user, read the results to the user, and ask user for some information. Details are described in Table 1, where elements on a web page are assumed to be given unique element IDs.

RUSS is trained to execute tasks by grounding natural language instructions on the web. The modular design of RUSS, with separate semantic parser and grounding model, is motivated by the high cost of training data acquisition, and the ability to improve each component independently.

We first describe ThingTalk, then the three components of Russ: the semantic parser model, the grounding model, and the runtime.

3.1 ThingTalk

ThingTalk is designed to be (1) robust to open-domain natural language, (2) a suitable target for semantic parsing from natural language, and (3) trainable with only synthetic data.

The primitives in ThingTalk include all the agent actions and a grounding function @retrieve (Table 1). The latter is informed by the descriptions in the instructions we found in the wild. The input features accepted by @retrieve are:

- *descr*: textual description of the element
- *type*: type of element (button, input box, paragraph, header, etc.)
- *loc*: absolute position of the element on the page
- *above/below/...*: position of the element relative to another; above, below, right, and left.

To support element descriptions involving multiple features or other elements, ThingTalk is com-

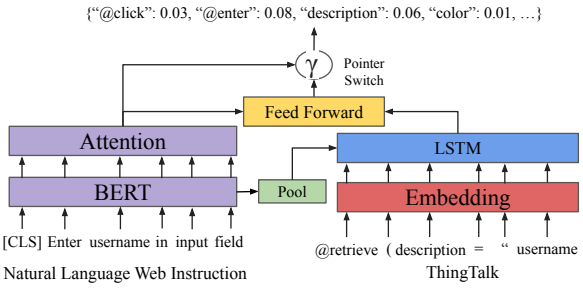

Figure 2: The RUSS semantic parser, using the BERT-LSTM architecture

Instruction: "Enter the user's order number in the text field that says order number"
DOM:
element_id: 1, type = "body"
element_id: 2, type = "h1", text = "Your Orders"
element_id: 3, type = "form"
...
element_id: 48, type = "label", text = "order number"
element_id: 49, type = "input"
...
ThingTalk:
@retrieve(*description* = "order number", *type* = input)
⇒ @enter(*text* = *order_number*, *element* = *id*)
Action: @enter(*text* = *order_number*, *element* = 49)

Figure 3: Representation of an instruction in RUSS

positional in design. A ThingTalk program is a sequence of statements with syntax $[r \Rightarrow]^* a$, where r is the retrieve operation and a is an agent action. @retrieve returns an element_id that is passed to @click (to click on the element), @read (to read the text in the element to the user), or @enter (to enter text in the element). For agent actions that require an element id, we call the sequence of @retrieve functions used to obtain the final element id used in the agent action the *query*. See Figure 1 for sample ThingTalk parses from natural language instructions. The orange ThingTalk parse demonstrates a query with 2 @retrieve functions.

3.2 Semantic Parser Model

To translate natural language instructions into ThingTalk, we use the previously proposed BERT-LSTM model (Xu et al., 2020). BERT-LSTM is an encoder-decoder network that uses a pre-trained BERT encoder (Devlin et al., 2019) and LSTM (Hochreiter and Schmidhuber, 1997) decoder with a pointer-generator (See et al., 2017; Paulus et al., 2018). The architecture is shown in Fig. 2. The model is trained to encode natural language utterances and produce the ThingTalk code token-by-token. The pointer network in the decoder allows the model to predict out-of-vocabulary words by copying from the input utterances.

We preprocess the natural language by performing *entity extraction*, where entity strings are mapped to placeholder tokens (URL, LOC, TYPE), and the strings are substituted back into the ThingTalk code after parsing with the placeholder tokens. This resolves errors related to long URLs being broken into tokens that are not always copied to ThingTalk together and helps disambiguate important input features. For example: "Click the button on the top of the amazon.com page" maps to "Click the TYPE on the LOC of the URL page". We use a simple set of heuristics to identify the entity strings for each placeholder token, such as the presence of a 'www.', '.com', 'http' substring to indicate a URL entity.

3.3 Grounding Model

The webpage is modeled using the Document Object Model (DOM), which is a hierarchical representation of all elements in the page. Our DOM representation records *element features* such as the following for each element:

- inner text content of the element
- HTML *id*, *tag*, *class*
- *hidden* state (True/False if element is visible on the webpage)
- height/width of the element
- left/right/top/bottom coords of the element
- list of child elements in the DOM.

An example is shown in Fig. 3.

RUSS's grounding model grounds a ThingTalk @retrieve function by mapping it to an element ID. The input features in the @retrieve function are mapped against scores derived from the element features in the DOM to identify the best match.

The grounding model consists of the following steps. It filters elements by their type and absolute location. Next it handles relative positioning by identifying those elements with the right relational context to, and not too far away from, the given element's coordinates. It passes the text of the remaining candidates through a Sentence-BERT (Reimers and Gurevych, 2019) neural network and computes the cosine similarities of their embeddings with the embedding of the input text

description. The elements with the highest score are returned.

3.4 The Run-Time

To execute the grounded ThingTalk program, RUSS starts a new automated Chrome session for each task and uses Puppeteer to automate web actions in the browser. RUSS uses a Google Voice API to implement actions involving user interactions (@say, @ask, or @read). For @ask actions, RUSS uses a preprogrammed dialogue to ask the user for a dictionary key (such as "name"), verifies the dictionary key is a valid string, and stores the value given by the user in a user's dictionary under that key. In @enter actions, we retrieve information to be entered by finding its closest match among the user's dictionary keys.

4 Datasets

This paper contributes two detasets, the RUSS Evaluation Dataset with real-world instructions and the RUSS Synthetic Dataset for training semantic parsers.

4.1 RUSS Evaluation Dataset

The RUSS Evaluation Dataset consists of real-world tasks from customer service help centers of popular online companies. To make our task-open domain, the online help centers we use span a diverse range of domains including music, email, online retail, software applications, and more. For each instruction in a task, the dataset includes:

- the English instruction in natural language as it appears in the original website, and the human-edited version of the instruction
- the DOM of the web page where the instruction can be executed, with the element features associated with each element
- the ThingTalk code corresponding to the instruction
- the grounded action of the instruction

To collect the RUSS Evaluation dataset, we acquire a list of "Top 100 visited websites" and locate tasks that offer line-by-line help instructions from those. An author of the paper walked through each task, performed the actions as instructed, scraped the webpage in the browser, and annotated the instruction with the corresponding ThingTalk code. Steps found missing from the instructions were inserted. If an instruction mapped to several actions, the text was broken into individual

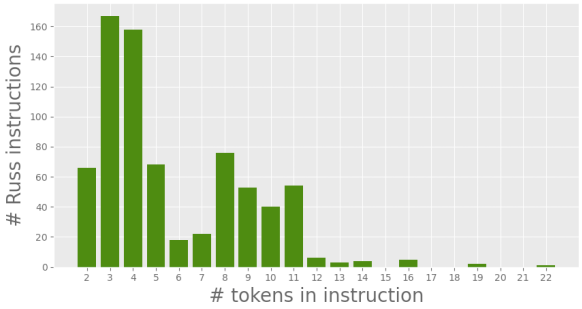

Figure 4: Lengths of instructions in the RUSS Evaluation Dataset

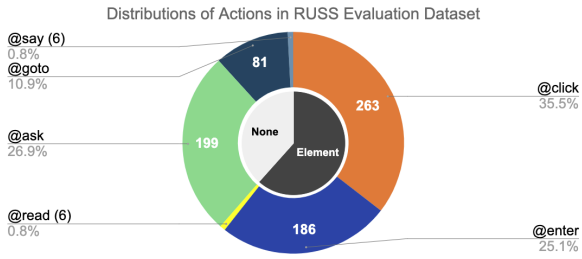

Figure 5: Distribution of actions in the RUSS Evaluation Dataset. @click, @enter, and @read require a webpage element.

instructions. Note that the human worker did not participate in the design of ThingTalk; they were asked to write instructions as if they were teaching another human step-by-step.

We collected a total of 80 tasks and 741 lines of instructions from 22 different online help centers. The dataset is split into a dev set and a test set, with 304 instructions from 30 tasks in the dev set and 437 instructions from 50 tasks in the test set. The RUSS Evaluation dataset is not used for training. On average, instructions in RUSS contain 9.6 tokens (Fig. 4), significantly longer than the crowdsourced web instructions in PhraseNode which average 4.1 tokens. The three most common actions in the dataset are "click", "ask" and "enter" (Fig. 5). 61.4% of the natural-language instructions require retrieving an element from the webpage (click, enter, read). Table 2 illustrates different types of reasoning supported by the @retrieve descriptors and their frequency in the RUSS Evaluation Dataset. Lastly, 76 of the 455 element queries use two @retrieve functions, with the rest all just using one, and 53.7%, 42.7%, and 3.6% of the @retrieve functions have 1, 2, and 3 descriptors, respectively (Fig. 6).

While the language has just 7 core actions, the combinatorial space of possible actions and web elements is much larger – on the order of 1000s

ThingTalk Includes: (@retrieve feature)	Description	Frequency
Type reasoning (type)	Requires specific HTML type (e.g. button, checkbox)	29.0%
Input target (type = input)	Requires target element is a text input	25.0%
Relational reasoning (below/above/left of...)	References neighboring features of the element	10.3%
Spatial reasoning (location)	References element location on the webpage	4.6%
No web element (No @retrieve)	No element (operation is @ask / @goto / @say)	38.6%

Table 2: Subset of reasoning types (with the @retrieve input feature used to indicate it) supported by ThingTalk and their frequency in the RUSS dataset. Some statements require multiple reasoning types.

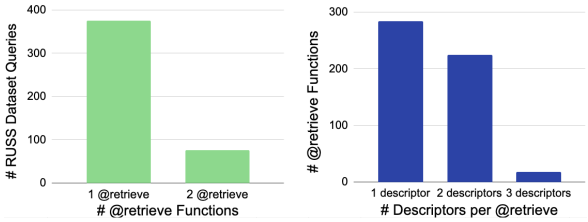

Figure 6: # @retrieve functions in each RUSS instruction and # descriptors in each @retrieve.

Model	Accuracy (test)
RUSS (1.5M training parses)	**87.0%**
Ablations	**Accuracy (dev)**
RUSS (1.5M training parses)	88.2%
− entity extraction	77.6%
− 1M training parses, entity extraction	70.0%

Table 3: Evaluation of Semantic Parsing Model (trained on 1.5M parses) on RUSS Evaluation test set. Ablations are performed on the dev set. "−" in Ablations subtracts a feature from the RUSS model, the second ablation is trained on 500k training parses.

of possible combinations per instruction. On average the DOMs of the webpages contain 689 web elements each.

The total vocabulary size of the Evaluation Dataset found in the wild is 684 words. We find that at least one of the most frequent 300 words in the Evaluation vocabulary is present in >50% of the Evaluation Dataset instructions. There are also many domain-specific words throughout the instructions.

4.2 Synthetic Dataset

Labeling large numbers of instructions in ThingTalk for training is time consuming and demands expertise. To address this, we use a typed template-based synthesis method to generate our training data. We write templates for each ThingTalk primitive and common combinations thereof. We also scrape a large dataset of naturally occurring DOM element text, webpage URLs, and phrases that are likely to be variable names to use for each parameter. The synthesizer compositionally expands the templates and sample values from the scraped dataset to construct a large training set of instructions mapped to ThingTalk automatically. We generate hundreds of different types of natural language templates which are combined to create a Synthetic Dataset with 1.5M training samples. This composition method creates roughly 840 distinct templates. To promote generalizability of our model, the total vocabulary size of the Synthetic corpus is large compared to the evaluation vocabulary size at 9305 words.

An example of a simple template is:

"At the **loc** of the page,
@**click** the button that says **descr**"

which is mapped to the ThingTalk:

@retrieve(*descr* = **descr**, *loc* = **loc**) ⟶
@**click**(element = id)

5 Evaluation

RUSS achieves **76.7%** overall accuracy on the Evaluation Dataset, even though all of RUSS, including the semantic parser is trained with only synthetic data.

We perform 3 experiments to evaluate the individual components and the system as a whole: 1) Accuracy evaluation of RUSS's Parsing Model with ablation studies. 2) Accuracy evaluation and baseline comparisons of RUSS's Grounding Model. 3) User study evaluating RUSS's ability to master 5 tasks on-the-job. We test usability and efficacy of RUSS compared with existing customer service help websites.

5.1 Semantic Parsing Accuracy

Our first experiment evaluates the accuracy of our semantic parser on the RUSS Evaluation dataset. We measure *Exact Match Accuracy*: a parse is considered correct only if it matches the gold annotation token by token.

Model	Grounding Acc (test)
RUSS	**63.6%**
End-to-End Baseline	51.1%
PhraseNode	46.5%

Table 4: RUSS outperforms state-of-the-art PhraseNode in the grounding subtask on the RUSS Evaluation test set.

Reasoning	RUSS	PhraseNode
Type	67.8%	61.5%
Input	75.6%	60.4%
Relational	70.0%	53.5%
Spatial	36.7%	30.3%

Table 5: Grounding Accuracy Comparison of RUSS and PhraseNode by Reasoning type on the RUSS Evaluation test set.

The results are shown in Table 3. The parser obtains **87.0%** accuracy on the test set. Despite using no real-world training data, the semantic parser achieves high accuracy on the challenging evaluation set. It achieves an accuracy of 81.4% for instructions involving web elements, and 94.6% for the rest. This suggests the semantic parser can handle both types of instructions with high accuracy, especially instructions that parse to user interactions (no web element).

We perform an ablation study on the RUSS Evaluation dev set as seen in Table 3. RUSS achieves 88.2% accuracy on the dev set. The entity extraction technique where string entities are replaced with placeholders during training, as discussed in Section 3.2, contributes 10.6% improvement in accuracy. Training without this pre-processing step and with only 500K parses will reduce the accuracy further by 7.6%. This suggests that it is important to have a large synthetic training data set.

5.2 Grounding Evaluation

With an effective semantic parser to ThingTalk, we next measure the grounding accuracy: the percent of correctly identified element_ids from the 252 natural language commands referring to web elements in the RUSS test set. As shown in Table 4, RUSS achieves an accuracy of 63.6%. 81.4% of the instructions are parsed correctly, and 77.9% of the correct parses are grounded accurately. Had the semantic parser been correct 100% of the time, the Grounding Model would achieve an accuracy of 73.0%. The semantic parser is more likely to correctly parse simple instructions such as "click sign in", which are also generally easier for the Grounding Model, explaining the delta between 77.9% and 73.0%.

We create an *End-to-end Baseline* model to compare against the 2-step approach of RUSS. Here, we represent web elements using RUSS's feature elements as before. However, we do not parse the natural language sentences into their input features in RUSS, but is left intact as input to Sentence-Bert to compute its embedding. Like Section 4.3, the element sharing the closest embedding with the input sentence is returned. This end-to-end baseline model performs with 12.6% less accuracy than RUSS, illustrating the benefits of using a semantic parser.

To compare our grounding model with state-of-the-art results, we also replicate the best performing embedding model from (Pasupat et al., 2018), which we reference as PhraseNode. The webpage features used as inputs in PhraseNode are a subset of our representation. PhraseNode achieves an accuracy of 46.5%, which is 4.6% worse than our Baseline and 17.2% lower than RUSS. We show that the combination of a high-performance semantic parser and a well-tuned grounding model can outperform the best end-to-end neural models for grounding on the web.

5.3 Analysis

The entire one-time process for training RUSS takes approximately 7 hours on an NVIDIA Tesla V100. RUSS can perform a new task on-the-job by running the instructions through the semantic parser in less than 1 minute.

We analyze how well RUSS and PhraseNode perform for sentences in the Evaluation Set requiring different types of reasoning (Table 5). Russ outperforms the state-of-the-art PhraseNode (Pasupat et al., 2018) for all the reasoning types. It performs well on grounding tasks that involve type, input, and relational reasoning. Evaluation of the spatial reasoning instructions revealed that many referenced image features (e.g. “click the hamburger menu icon”), which is not supported by RUSS. The results show that ThingTalk is simple enough to be generated by a neural language model, while comprehensive enough to express the wide range of open-domain natural language instructions for web tasks.

Unlike end-to-end models that struggle with long, complex instructions, we find that RUSS ben-

# 1	Redeem Amazon Gift Card
# 2	Get Pinterest Ad Account Number
# 3	Log out of all Spotify accounts
# 4	Create new Walmart account
# 5	Send Google feedback

Table 6: Tasks in RUSS User Study

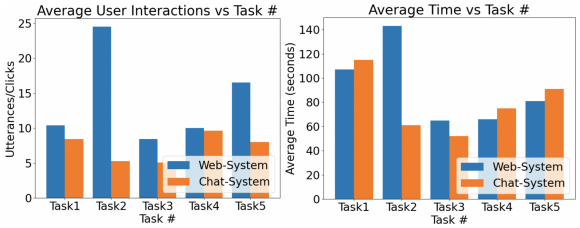

Figure 7: Average number of user interactions via utterance or click (left); average time taken to complete tasks in seconds (left)

efits from added reasoning in instructions that constrains the potential set of element candidates (e.g. "the element must be an input"). Webpages commonly have thousands of elements and the probability of matching the right element increases with constraints.

Of the 741 instructions in the RUSS dataset, 6 contain attributes that are not well expressed in ThingTalk. For example, "select the user's birth month in the month drop down" is not parsed correctly because ThingTalk does not have a notion of selecting an element in a menu. This feature will be added in the future.

Another source of errors lies in how webpages are constructed. Important attributes needed for grounding can be hidden behind classes. For example, an element may be labeled as "Click here", but the text is not present in the DOM text attribute and instead obscured behind a site-specific class name such as "next-page-button". Grounding techniques on visual data can be helpful in resolving this class of problems.

5.4 User Study

The goal of our user study is to evaluate the end-to-end feasibility of RUSS on open-domain instructions from real customer service websites, and evaluate how users respond to RUSS. This is a small-scale study with promising early results, but can benefit from further user studies on larger populations.

We recruited 12 participants who were asked to complete 5 customer-support tasks (Table 6), chosen from popular websites: Amazon, Spotify, pinterest, Google, and Walmart, with both RUSS and the browser. For all tasks, users were given a fake persona (a set of credentials such as email, password, gift card code, etc) to use when interacting with the agent. The study was approved by our IRB and participants were compensated.

The participants in our study ranged from ages 21 to 68 years old, with an average age of 36 years old, a 50/50 male/female ratio, and varied technical sophistication. To reduce learning effects, we used Latin Square Balancing (Bradley, 1958) to ensure that both the web and RUSS trials of each site were performed first half the time. We record users' time to perform each task, number of turns (in RUSS) or clicks (on the web) required to achieve each task, and gave each participant an exit survey containing qualitative assessments.

Participants were able to complete **85%** of the tasks on their own on the web and **98%** of tasks with the help of RUSS. Those who did not finish their task either gave up or failed to complete the task within 5 minutes. The time it took users to accomplish each task was similar for the Web and RUSS (Fig. 7), though RUSS was significantly faster for Task 2, a more complex task users said they were unfamiliar with. This seems to indicate that RUSS is more favorable for unfamiliar, complex tasks.

After trying the 5 tasks, **69**% of users reported they prefer RUSS over navigating online help pages. Reasons cited include ease of use, efficiency, and speed, even though the times of completion were similar. Participants were generally pleased with their RUSS experience, and only one person said that they were unlikely to use RUSS again (Fig. 8). However, many users did report that they wished RUSS was as visually stimulating as the browser. Other users noted that they felt more familiar and comfortable with the browser.

As a final discussion, it is worth noting that while the user study results are extremely promising, this is a small scale study. RUSS's runtime needs stronger error handling for out-of-context conversation. Currently, RUSS gives the user 3 tries to return an expected response before terminating. RUSS also times out if a webpage takes more than >60 seconds to load in Puppeteer. We saw instances of both of these situations in the RUSS user study in the few cases the user failed to complete a task.

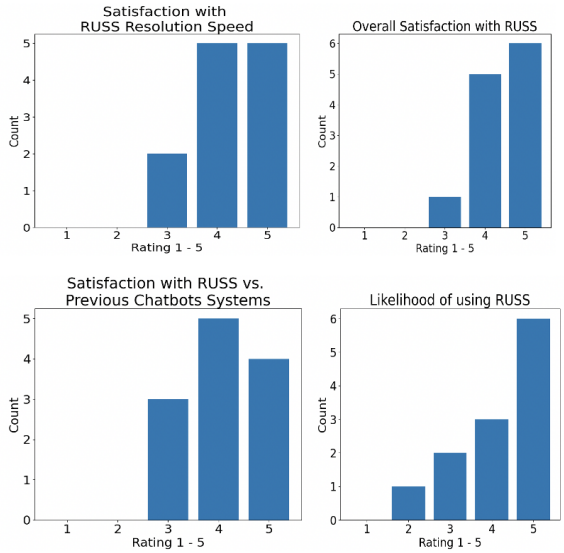

Figure 8: Qualitative results from user studies. On a scale 1-5 for satisfaction, 1 = not satisfied at all and 5 = exceeded expectations. For likelihood, 1 = will never use again and 5 = will definitely use again.

6 Conclusion

RUSS demonstrates how a semantic parser and grounding model can be used to perform unseen web tasks from natural language instructions. By achieving 76.7% accuracy on the RUSS Evaluation Dataset, we show how a modular semantic parsing approach can outperform end-to-end neural models on this task, and demonstrate how humans interact with RUSS-like systems in the user study. Like many datasets in NLP, we believe extensive research is still required to go from RUSS's 76.6% overall accuracy on the Evaluation Dataset to 100%. As seen in Table 4, prior models like PhraseNode achieve only 46.5% grounding accuracy, which points to additional work necessary in grounding natural language on the web.

The RUSS Evaluation dataset introduces a set of real instructions for grounding language to executable actions on the web to evaluate future research in this direction, including training semantic parsers to new targets using real-world instructions and neural models for grounding formal language representations on the web. Our work provides the task, technical foundation, and user research for developing open-domain web agents like RUSS.

7 Ethical Considerations

The user study conducted in this paper was submitted to the Institutional Review Board and received IRB Exempt status. All participants were read an IRB consent form prior to the user study, which detailed the study details. No deception was involved in the study: all participants knew they were evaluating an AI agent in the conversation portion of the user study and were not led to believe otherwise. They study took about 20 minutes. All participants were compensated with $10.

The webpages scraped for the RUSS dataset are all public domain webpages. No individual personal identifying information was used to obtain the webpages. On websites that required accounts to access pages, we created fake user accounts with non-identifying usernames / passwords / emails to navigate the websites in order to limit any privacy risks that may be involved.

In the future, we see web agents like RUSS helping improve accessibility by helping individuals who are visually impaired, less technologically advance, or otherwise preoccupied receive equitable access to information. Before systems like RUSS are put to practice at scale, the authors believe more research must be done in understanding user behavior with web agents to safeguard against downstream consequences of system errors and to better understand how information can be effectively delivered by AI agents that operate in potentially high-stakes transactions such as health or finance. Our user study is the first step in this direction.

8 Acknowledgments

We thank Silei Xu for helpful discussions on constructing the Synthetic dataset, and Richard Socher for feedback and review of the final publication.

This work is supported in part by the National Science Foundation under Grant No. 1900638 and the Alfred P. Sloan Foundation under Grant No. G-2020-13938.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views, policies, or endorsements of outside organizations.

References

James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A collaborative task

learning agent. In *2007 AAAI Conference on Artificial Intelligence (AAAI)*, pages 1514–1519. AAAI.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. 2015. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

James V. Bradley. 1958. Complete counterbalancing of immediate sequential effects in a latin square design. pages 525–528.

Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S. Lam. 2019. Genie: A generator of natural language semantic parsers for virtual assistant commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 394–410, New York, NY, USA. ACM.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011)*, pages 859–865.

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 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Michael H Fischer, Giovanni Campagna, Euirim Choi, and Monica S Lam. 2020. Multi-modal end-user programming of web-based virtual assistant skills. *arXiv preprint arXiv:2008.13510*.

Larry Heck, Dilek Hakkani-Tür, Madhu Chinthakunta, Gokhan Tur, Rukmini Iyer, Partha Parthasarathy, Lisa Stifelman, Elizabeth Shriberg, and Ashley Fidler. 2013. Multi-modal conversational search and browse. In *First Workshop on Speech, Language and Audio in Multimedia*.

Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. 2018. Grounding visual explanations. In *European Conference on Computer Vision*, pages 269–286. Springer.

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

Nobuhiro Ito, Yuya Suzuki, and Akiko Aizawa. 2020. From natural language instructions to complex processes: Issues in chaining trigger action rules. *arXiv preprint arXiv:2001.02462*.

Siddharth Karamcheti, Dorsa Sadigh, and Percy Liang. 2020. Learning adaptive language interfaces through decomposition. *arXiv preprint arXiv:2010.05190*.

Hyounghun Kim, Abhay Zala, Graham Burri, Hao Tan, and Mohit Bansal. 2020. Arramon: A joint navigation-assembly instruction interpretation task in dynamic environments. *arXiv preprint arXiv:2011.07660*.

Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. Coscripter: Automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, page 1719–1728, New York, NY, USA. Association for Computing Machinery.

Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. Sugilite: Creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, page 6038–6049, New York, NY, USA. Association for Computing Machinery.

Toby Jia-Jun Li and Oriana Riva. 2018. Kite: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, page 96–109, New York, NY, USA. Association for Computing Machinery.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations*.

Ramesh Manuvinakurike, Jacqueline Brixey, Trung Bui, Walter Chang, Doo Soon Kim, Ron Artstein, and Kallirroi Georgila. 2018. Edit me: A corpus and a framework for understanding natural language image editing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Piotr Mirowski, Matt Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Andrew Zisserman, Raia Hadsell, et al. 2018. Learning to navigate in cities without a map. In *Advances in Neural Information Processing Systems*, pages 2419–2430.

Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*.

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, Brussels, Belgium. Association for Computational Linguistics.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *ICLR*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3973–3983.

Ritam Jyoti Sarmah, Yunpeng Ding, Di Wang, Cheuk Yin Phipson Lee, Toby Jia-Jun Li, and Xiang 'Anthony' Chen. 2020. Geno: A developer tool for authoring multimodal interaction on existing web applications. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 1169–1181, New York, NY, USA. Association for Computing Machinery.

Zhanna Sarsenbayeva. 2018. Situational impairments during mobile interaction. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, pages 498–503.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*.

Statista Research Department. 2019. Share of customers in the united states who have contacted customer service for any reason in the past month from 2015 to 2018.

Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov, Nick Walker, Yuqian Jiang, Harel Yedidsion, Justin Hart, Peter Stone, and Raymond J Mooney. 2019. Improving grounded natural language understanding through human-robot dialog. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6934–6941. IEEE.

Jesse Thomason, Jivko Sinapov, Maxwell Svetlik, Peter Stone, and Raymond J Mooney. 2016. Learning multi-modal grounded linguistic semantics by playing "i spy". In *IJCAI*, pages 3477–3483.

Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. 2015. Sequence to sequence – video to text. In *Proceedings of the 2015 International Conference on Computer Vision (ICCV-15)*, Santiago, Chile.

Silei Xu, Giovanni Campagna, Jian Li, and Monica S Lam. 2020. Schema2QA: High-quality and low-cost Q&A agents for the structured web. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1685–1694.