Learning Adaptive Language Interfaces through Decomposition

Siddharth Karamcheti and **Dorsa Sadigh** and **Percy Liang**
Computer Science Department, Stanford University
{skaramcheti, pliang, dorsa}@cs.stanford.edu

Abstract

Our goal is to create an interactive natural language interface that efficiently and reliably learns from users to complete tasks in simulated robotics settings. We introduce a neural semantic parsing system that learns new high-level abstractions through *decomposition*: users interactively teach the system by breaking down high-level utterances describing novel behavior into low-level steps that it can understand. Unfortunately, existing methods either rely on grammars which parse sentences with limited flexibility, or neural sequence-to-sequence models that do not learn efficiently or reliably from individual examples. Our approach bridges this gap, demonstrating the flexibility of modern neural systems, as well as the one-shot reliable generalization of grammar-based methods. Our crowdsourced interactive experiments suggest that over time, users complete complex tasks more efficiently while using our system by leveraging what they just taught. At the same time, getting users to trust the system enough to be incentivized to teach high-level utterances is still an ongoing challenge. We end with a discussion of some of the obstacles we need to overcome to fully realize the potential of the interactive paradigm.

1 Introduction

As robots are deployed in collaborative applications like healthcare and household assistance (Scassellati et al., 2012; Knepper et al., 2013), there is a growing need for reliable human-robot communication. One such communication modality that is both user-friendly and versatile is natural language; to this end, we focus on robust natural language interfaces (NLIs) that can map utterances to executable behavior (Tellex et al., 2011; Artzi and Zettlemoyer, 2013; Thomason et al., 2015; Arumugam et al., 2017; Shridhar et al., 2020).

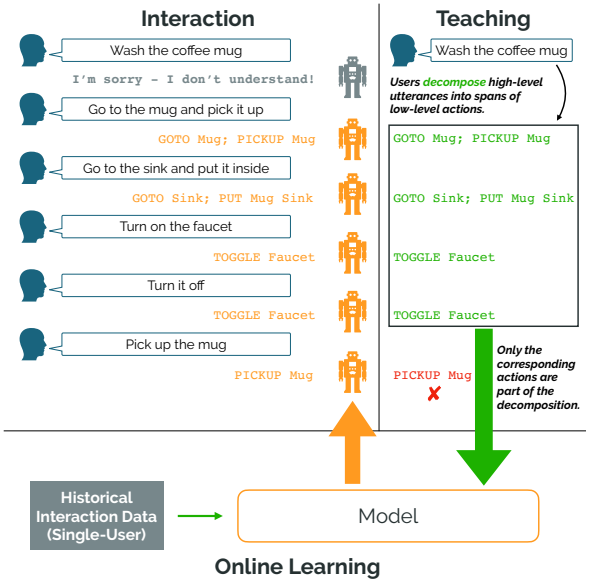

Figure 1: In our proposed framework, users interact with a simulated robot to complete tasks. Central to our approach is *learning by decomposition*: users teach the system to understand novel high-level utterances by breaking them down into utterances that the system can understand and execute. Using these decompositions, we update a semantic parser online, allowing our system to adapt to users as they complete more tasks.

Most existing work on NLIs (and AI systems more broadly) falls into a static train-then-deploy paradigm: models are first trained on large datasets of (language, action) pairs and then deployed, with the hope they will reliably generalize to new utterances. Yet, what happens when such models make mistakes or are faced with types of utterances unseen at training — for example, providing a household robot with a novel utterance like "wash the coffee mug?" Such static systems will fail with no way to recover, burdening the user to find alternate utterances to accomplish the task (or give up). Instead, we argue that NLIs need to be dynamic and adaptive, learning interactively from user feedback

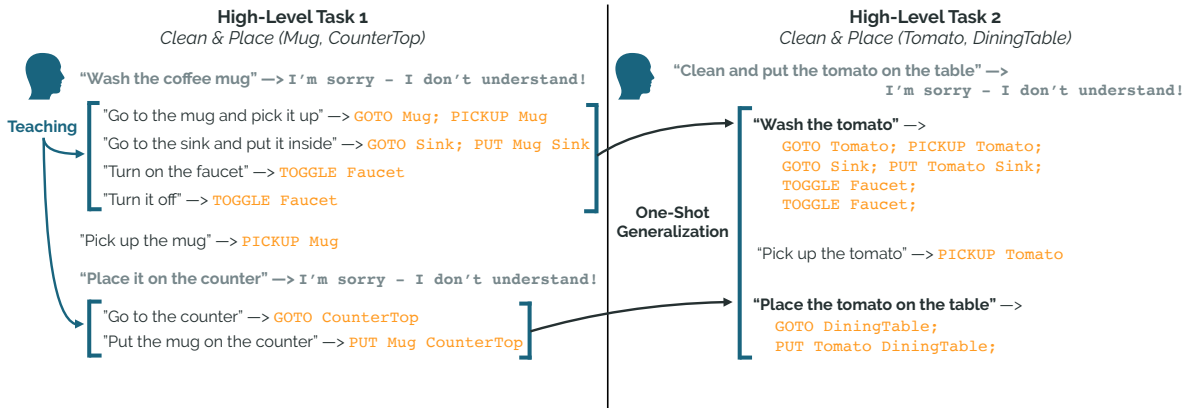

Figure 2: One-shot generalization example: When the system fails to understand an utterance (e.g. "wash the coffee mug", "place it on the counter"), the user teaches the system by *decomposing* it into other utterances the system can understand (illustrated by brackets above), which eventually get mapped to low-level actions that are executed. This induced mapping of high-level utterance to low-level actions forms an example that we use to update our semantic parser online. Because our semantic parser is capable of reliable *one-shot generalization*, users can leverage these decompositions when completing the next task.

to index and perform more complicated behaviors.

In this work, we explore building NLIs for simulated robotics that learn from real humans. Inspired by Wang et al. (2017), we leverage the idea of *learning from decomposition* to learn new abstractions. Just like how a human interactively teaches a new task to a friend by breaking it down, users interactively teach our system by simplifying utterances that the system cannot understand (e.g. "wash the coffee mug") into lower-level utterances that it can (e.g. "go to the coffee mug and pick it up", "go to the sink and put it inside", etc. — see Figure 1).

To map language to executable behavior, Wang et al. (2017) and Thomason et al. (2019) built adaptive NLIs that leverage grammar-based parsers that allow reliable *one-shot generalization* but lack *lexical flexibility*. For example, a grammar-based system that understands how to "wash the coffee mug" may not generalize to "clean the mug." Meanwhile, recent semantic parsers are based primarily on neural sequence-to-sequence models (Dong and Lapata, 2016; Jia and Liang, 2016; Guu et al., 2017). While these models excel from a *lexical flexibility* perspective, they lack the ability to perform reliable *one-shot generalization*: it is difficult to train them to generalize from individual examples (Koehn and Knowles, 2017).

In this paper we propose a new interactive NLI that is *lexically flexible* and can *reliably and efficiently perform one-shot generalization*. We introduce a novel exemplar-based neural network semantic parser that first abstracts away entities (e.g. "wash the coffee mug" → "wash the <obj>"), allowing for generalization to previously taught utterances with novel object combinations. Our parser then retrieves the corresponding "lifted" utterance and respective program (exemplar) from the training examples based on a learned metric (implemented as a neural network), giving us the lexical flexibility of sequence-to-sequence models.

We demonstrate the efficacy of our learning from decomposition framework through a set of human-in-the-loop experiments where crowdworkers use our NLI to solve a suite of simulated robotics tasks in household environments. Crucially, after completing a task, we update the semantic parser so that users can immediately reuse what they taught. We show that over time, users are able to complete complex tasks (requiring several steps) more efficiently with our exemplar-based method compared to a neural sequence-to-sequence baseline. However, for more straightforward tasks that can be completed in fewer steps, we see similar performance to the baseline. We end with an error analysis and discussion of user trust and incentives in the context of building interactive semantic parsing systems, paving the way for future work that better realizes the potential of the interactive paradigm.

2 Learning from Decomposition

User sessions are broken up into a sequence of *episodes* (individual tasks), each comprised of *two phases*: 1) *Interaction*, where the user provides utterances to the system to accomplish the task, and 2) *Teaching*, where the user teaches the system to understand novel utterances (Figures 1 and 2).

Primitive Action	Canonical Utterance
`GOTO <OBJ>`	go to <obj>
`PICKUP <OBJ>`	pick up <obj>
`OPEN <OBJ>`	open <obj>
`CLOSE <OBJ>`	close <obj>
`TOGGLE <OBJ>`	turn on/off <obj>
`PUT <OBJ> <OBJ>`	put object <obj>

Table 1: List of primitive programmatic actions and seed utterances used to initialize our semantic parser. Note that the utterances are *lifted*; they do not include references to concrete objects. This enables one-shot generalization to unseen object combinations.

2.1 Interaction

During interaction, the user attempts to complete a task by producing a sequence of user utterances $u_1, u_2, \ldots$ with the corresponding system responses $p_1, p_2, \ldots$ (including the `NOT-SURE` action) that are executed in the environment (the `NOT-SURE` action executes to an error message "I'm sorry - I don't understand!"). For example, in Figure 1, the user first says the novel utterance "wash the coffee mug," and the system returns `NOT-SURE`. The user follows up with "go to the mug and pick it up," which the system maps to the program `GOTO Mug; PICKUP Mug`. This continues until the user has completed the task. If the system or user makes a mistake and produces an undesired action, the user must continue to provide utterances, as there are no resets.

2.2 Teaching

The goal of teaching is to convert the sequence of utterance-action pairs (u_i, p_i) into a set of valid training examples for updating the system. To do this, the system presents the user with each u_i where p_i is `NOT-SURE`, and asks the user to select the corresponding contiguous sequence of actions $p_{i+1}, \ldots p_j$. To facilitate comprehension, we show users (programatically generated) human-readable representations of each action p — e.g. "go to the mug" for a program $p =$ `GOTO Mug`. For example, the user maps "wash the coffee mug" to the sequence `GOTO Mug; PICKUP Mug; ... TOGGLE Faucet` (see Figure 1 for the full decomposition). Similarly, the user maps "place it on the counter" to `GOTO CounterTop; PUT Mug CounterTop`. The resulting examples $(u_i, \hat{p}_i = p_{i+1} \ldots p_j)$ are used to update the system (details in Section 3.2.2). We update *every time* a user completes a task and teaches new examples — this allows users to access what they have taught immediately, during the following task.

2.3 Desiderata

This example illustrates two desiderata for our framework, both of which are key to *trust*: 1) the ability to identify novel types of utterances (when to output `NOT-SURE`), as well as 2) the ability to perform one-shot generalization. Knowing when to output `NOT-SURE` is key to *trust during inference*: signaling to users what the system knows, so that the simulated robot does not take undesired actions (like dropping your coffee mug on the floor). Performing one-shot generalization is key to *trust during learning*: users need to rely on the system remembering what has been taught so they can more efficiently complete future tasks. For example, when the user is completing the next task (second half of Figure 1), they should be able to rely on the system understanding "wash the tomato" and "place the tomato on the table," even though these refer to different objects than in the taught examples. Section 3 discusses how we enable one-shot generalization in further detail.

Sequence-to-sequence models fail. We found modern neural sequence-to-sequence models to be a poor fit in our setting. The biggest problem we found was their ability to handle novel utterances. Anecdotally, we found when given the novel utterance "wash the coffee mug," a neural sequence-to-sequence system trained on the seed set of utterances in Table 1 returned the program `OPEN Mug`, which does not even execute. These problems are exacerbated by the lack of training data; a single user's interaction only creates a handful of new examples, contraindicating the use of data-hungry sequence-to-sequence models (Koehn and Knowles, 2017).

3 Semantic Parsing

To address the above desiderata (identifying when to output `NOT-SURE`, and one-shot generalization), we incorporate two key insights into our approach. To identify when to output `NOT-SURE`, we look at the distances between a new utterance and the utterances in our training set, similar to the exemplar-based approach of Papernot and McDaniel (2018) — if an utterance is "close enough" to a training utterance, return the corresponding program, otherwise return `NOT-SURE`. To enable

one-shot generalization, our parser operates over *lifted* versions of utterances and programs — versions that abstract out explicit references to objects (allowing for automatic generalization to new combinations of objects unseen during training).

We now describe our semantic parser, which maps a user utterance u and environment state s to the corresponding program p that best reflects the meaning of the user's utterance. In this work, a state s consists of a set of objects where each object is defined by a fixed set of features (e.g. *visibility*, *toggle status*, etc.). We define a program p as a sequence of primitive actions, where each action consists of a template (from Table 1) with arguments corresponding to object types. We conclude with a description of how we retrain our semantic parser using the newly taught examples from the teaching phase (Section 2.2).

3.1 Model

Our semantic parser (Figure 3) takes an utterance u and first abstracts out entities (Section 3.1.1), creating *object references* and *lifted utterances*. We parse these into *object types* (Section 3.1.1) and *lifted programs* (Section 3.1.2), which are combined (Section 3.1.3) and fed to a reranker that additionally uses the state s (Section 3.1.4) to identify the program p^* to execute.

3.1.1 Entity Abstraction & Resolution

We define an entity abstractor that maps an utterance u (e.g. "wash the coffee mug") to a lifted utterance f (e.g. "wash the <obj>") and a list of object references $\mathcal{O}$ (e.g. ["coffee mug"]). The entity resolver maps each object reference $o \in \mathcal{O}$ (e.g. "coffee mug") to a grounded object type g (e.g. `Mug`) resulting in a new list $\mathcal{G}$. To do this, we exploit a set of "typical names," (e.g. `Mug` = {"coffee mug", "mug", "cup"}) that we define a priori, looking up the object type with the given name. However, if there are multiple types that share the given name (e.g. in our dataset, table is a "typical name" for `DiningTable`, `CoffeeTable`, `SideTable`), we use the current state s to disambiguate: we fetch all the matching items in s and return the physically closest one.

3.1.2 Semantic Parsing

Central to our approach is the exemplar-based semantic parser that maps a lifted utterance f to a set of lifted programs $\mathcal{Q}$. To do this, we learn a classifier p_θ that takes two lifted utterances (f, f')

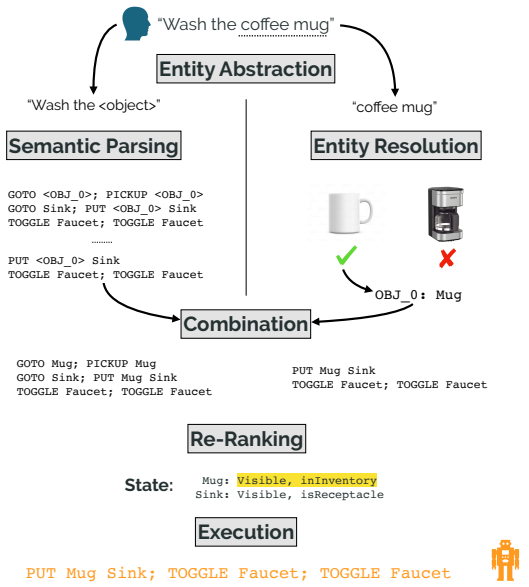

Figure 3: Semantic parsing pipeline. First, entities are extracted and the corresponding outputs — the lifted utterance and object references — are parsed into programs and grounded object types. These are combined and re-ranked to identify the program to execute.

and predicts a probability whether they have the same lifted program ($q = q'$). We take $\mathcal{Q}$ to be the programs corresponding to the highest probability f' under p_θ.

Embedding Utterances. We first embed each utterance with an embedding function ϕ, implemented as a neural network that first uses GloVe (Pennington et al., 2014) to embed the words in f followed by position encoding similar to that used in Vaswani et al. (2017) and a nonlinear transform. The resulting embeddings are summed and fed into to a two-layer MLP to create the utterance embedding $\phi(f)$. The classifier p_θ outputs $\sigma(a \text{ cos-sim}(\phi(f), \phi(f')) + b)$, where cos-sim is cosine similarity, a, b are learned scalars, and σ is the sigmoid function. We train p_θ with a binary cross-entropy objective on a training set of (lifted utterance, lifted program) pairs: $\{(f_i, f_j, [q_i = q_j]) : i, j \in [n]\}$.

Efficient Inference. We now describe how we use p_θ for inference given a new lifted utterance f'. Unfortunately, naïve application of p_θ for a new f' requires pairwise comparison with *every training example*. We streamline this by using the structure of our embedding space — as the classifier outputs the scaled cosine similarity between two utterances, we store the embeddings $\phi(f_i)$ for each training utterance (f_i, q_i) in our dataset, then use

an approximate nearest neighbors algorithm to find the the set of utterances that are "close-enough"; we use the corresponding lifted programs to form the output set $\mathcal{Q}$. We formalize what it means for an utterance to be "close-enough" in the following paragraph. We note that this procedure is similar to COSINEBERT (Mussman et al., 2020), a model used for active learning on pairwise language tasks.

Setting a Threshold. One of the desiderata of our system is returning `NOT-SURE` for utterances it is not confident about. To do this, we set a *threshold* τ such that if $\|\phi(f) - \phi(f')\|_2 \geq \tau$, return `NOT-SURE`. Note that this is equivalent to to thresholding the probability output by p_θ which is monotonic in the cosine distance as defined above. We set this threshold using a held-out validation set of (utterance, program) pairs (defined based solely on the seed examples in Table 1). For each utterance in the validation set f, we set τ such that 90% of the programs corresponding to utterances with τ are correct. Given an utterance f' at test time, we return the set of lifted programs $\mathcal{Q}$ corresponding to all lifted utterances within τ of $\phi(f')$ (all lifted utterances "close enough" to f').

Handling Compositionality. For multi-action utterances (e.g. "go to the apple **and** pick it up") we heuristically split on the keyword "and," resulting in multiple substrings. We parse each substring obtaining subsets of lifted programs, and take the cross-product of these subsets as the final set $\mathcal{Q}$. We acknowledge that this is not a perfect heuristic; in future work we hope to explore more general extensions that allow us to efficiently interpret utterances that have been composed in this way.

Implementation Details. When identifying the threshold τ, we define a hyperparameter lower bound β; this lower bound ensures that our semantic parser isn't overly conservative (returning `NOT-SURE` despite being moderately confident about the set of candidate programs). We find a value $\beta = 0.15$ works well for our experiments. We use Spotify's `annoy` library as our approximate nearest neighbors store for fast lookups.

We initialize our exemplar-based parser with seed examples (utterances mapped to programs) that cover the set of actions. Table 1 shows these actions, and a subset of the utterances used for training — our full dataset consists of only 44 examples (minor variations of the trigger words in the table). This is similar to prior work that defines a set of *canonical utterances* (Wang et al., 2015), or a *core grammar* (Wang et al., 2017). We strip stop words (*the, up, down, on, off, of, in, to, then, a, an, back, front, out, from, with, inside, outside, below, above, top*) from f prior to feeding to our parser to make our model more robust to minor lexical variation.

3.1.3 Combination

We combine each lifted program $q \in \mathcal{Q}$ with the grounded object types $\mathcal{G}$ to form a set of grounded programs $\mathcal{P} = \{p_1, \ldots, p_k\}$. In general, given a lifted program q that takes a sequence of arguments (e.g `PUT <OBJ> <OBJ>`) and a list of object types (e.g. $\mathcal{G}$ = `[Mug, DiningTable]`), we simply substitute the object types into the program, replacing each argument in the lifted program. This results in a final grounded program (e.g. p = `PUT Mug DiningTable`).

3.1.4 Reranking

The semantic parser, entity resolver, and combination step produce a set of grounded programs $\mathcal{P}$. The reranker takes the original utterance u, current state s, and this set of grounded programs $\mathcal{P}$ and chooses a single candidate $p^* \in \mathcal{P}$ to execute.

As a first step, we discard candidate programs that fail to execute in our simulator: for example, `PICKUP Mug` is discarded if the robot is already holding an object. Then we use a neural network to produce a score for each $p_i \in \mathcal{P}$. This network separately embeds the utterance, state, and each candidate program, feeding the concatenated embeddings to a two-layer MLP to produce a real-valued score for each p_i. In our work, the state s is retrieved dynamically based on the grounded objects $\mathcal{G}$ returned by the entity resolver; the state is made up of hand-coded features corresponding to attributes like *visibility*, *toggle status*, and *whether it can be picked up*, amongst others. We use a similar scheme as the semantic parser (Section 3.1.2) to encode utterances and candidate programs (embed, position encode, and sum), and a simple linear transformation to encode the bag-of-features representing the state s.

The highest-scoring candidate $p^* \in \mathcal{P}$ is executed. The reranker is trained via the process described in Section 3.2.3 *only after new examples are taught by users* during the teaching phase following each task they are asked to complete.

3.2 Retraining from User Feedback

In the following subsections, we discuss how to retrain our semantic parser and reranker to achieve the second of the two desiderata desired of our system: reliable and efficient *one-shot generalization*. As input to the retraining procedure, we take the dataset $\hat{\mathcal{D}} = (u_i, \hat{p}_i)$ of newly taught examples from the teaching phase (Section 2.2).

3.2.1 Creating Lifted Examples

Retraining the exemplar-based semantic parser requires converting our grounded dataset $\hat{\mathcal{D}}$ to pairs of lifted utterances and programs. Consider the grounded example ("Place the tomato on the table", `GOTO DiningTable; PUT Tomato DiningTable`); we want to map this to its lifted form ("Place the <obj> on the <obj>", `GOTO <OBJ> PUT <OBJ> <OBJ>`). To do this, we use the entity abstractor and resolver (from Section 3.1.1) to factor out object references.

Concretely, using the entity abstractor on the above example leaves us with $\hat{f}$ = "Place the <obj> on the <obj>", and references $\hat{\mathcal{O}}$ = ["tomato", "dining table"], which the entity resolver maps to $\hat{\mathcal{G}} =$ [`Tomato, DiningTable`]. We replace any element of $\mathcal{G}$ that occurs in the original program with the generic `<OBJ>` token to create the lifted program ($\hat{q} =$ `GOTO <OBJ>; PUT <OBJ>`). Applying this procedure to each example in $\hat{\mathcal{D}}$ gives us our lifted examples $(\hat{f}, \hat{q})$.

3.2.2 Updating the Semantic Parser

Updating the semantic parser requires optimizing the binary cross-entropy objective from Section 3.1.2 using these lifted examples $(\hat{f}, \hat{q})$. As we train our parser from pairs of examples, and there are far more negative examples (pairs with different programs) than positives, we over-sample positive examples so that batches have an equal number of positives and negatives. We train on the entire history of data for the given user, re-creating the nearest neighbors store with embeddings of each training utterance f_i. After this step, we recalibrate the nearest neighbors threshold using the procedure in Section 3.1.2.

3.2.3 Updating the Reranker

After updating the semantic parser, we re-parse each utterance in our dataset to define our retraining dataset of $(\hat{u}_i, \hat{\mathcal{P}}_i, \hat{s}_i)$ tuples. We use the program p^* that was actually executed for utterance $\hat{u}_i$ in state $\hat{s}_i$ as the "gold" label for the reranker. We train the reranker by maximizing the log-likelihood (minimizing the cross-entropy loss) of this candidate p^* amongst the others.

4 Experiments

We evaluate our approach with a set of human-in-the-loop experiments where crowdworkers are tasked with solving a series of simulated robotics tasks. Users interact with our system over 5 episodes (where each episode consists of a single task), teaching our system new examples after successfully completing each one. Each user has their own individual semantic parser and re-ranker (models are not shared across the users), with both components updating online after each teaching phase, prior to the start of the next task. Updating the two models (including rebuilding the nearest neighbors store) after each teaching phase varies depending on task complexity, but takes anywhere from 28 – 63 seconds on an Amazon EC2 T2.Medium (2 CPUs, 4 GiB RAM, no GPU) instance.

4.1 Experimental Setup

Environment and Tasks. Our experiments take place in simulated household environments, with users completing structured, everyday tasks. We create a 2D web-client inspired by the AI2-THOR Simulation Environment (Kolve et al., 2017) that removes the 3D rendering and spatial layout, but preserves the object types, attributes, and relations.

We borrow our tasks from the ALFRED Dataset (Shridhar et al., 2020) that defines 7 task types: 1) *Pick and Place*, 2) *Pick Two Objects and Place*, 3) *Look at Object in Light*, 4) *Nested Pick and Place*, 5) *Pick, Clean, and Place*, 6) *Pick, Heat, and Place*, and 7) *Pick, Cool, and Place*.

Interactive User Studies: We run our interactive user studies via Amazon Mechanical Turk (AMT). Each user is assigned one of the 7 task types, and is asked to complete 5 tasks of that type in a row. We recruited 20 workers per approach. Workers were paid \$5 with an average completion time of 23 minutes. We limit our AMT studies to workers with an approval rating $\geq 98\%$, location = US, and a total number of completed HITs > 5000.

Baseline. We compare our approach with a neural sequence-to-sequence with attention model similar to Jia and Liang (2016). To improve reliability, if the user enters an utterance that can be handled by a simple grammar that covers the core utterances

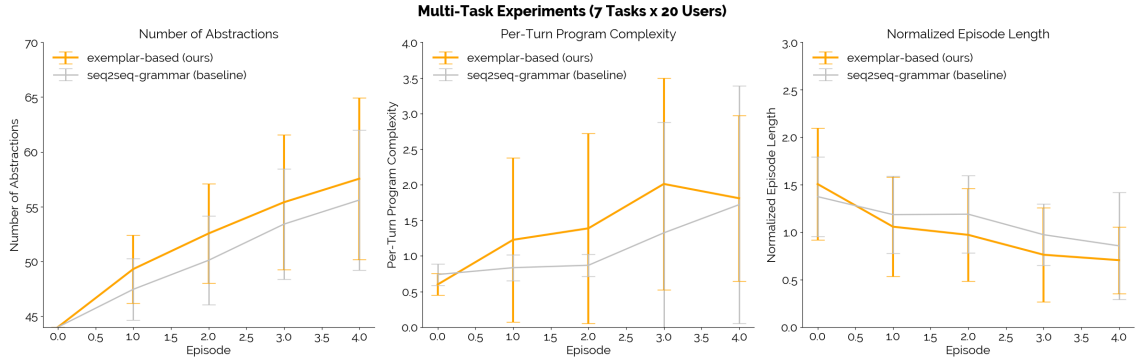

Figure 4: Complete set of results across 20 users with 7 different task types. Each user is given a single task type, and asked to complete 5 different episodes, with different combinations of environments and objects. The graph on the left shows the *number of examples taught* over 5 episodes. The graph in the middle shows the *per-turn program complexity* (number of primitives per language utterance) over time. The last graph shows the *normalized episode length* (# utterances to solve task / number of actions required).

from Table 1, we return the resulting program; otherwise, we invoke the sequence-to-sequence model. We find the inclusion of such a grammar necessary to prevent users from getting stuck. We refer to this combination of a neural sequence-to-sequence with a grammar as "*seq2seq-grammar*", whereas we refer to our system as "*exemplar-based*". We keep the learning by decomposition framework identical for both our system and the sequence-to-sequence system — in other words, we simply swap out our exemplar-based neural parser described in Section 3.1.2 for the *seq2seq-grammar* model.

Metrics. We define three evaluation metrics:

1. *Total number of examples taught*: The number of unique (utterance, program) pairs that the users teach the system across each teaching phase (as described in Section 2.2). This number starts at 44, the number of unique seed examples from Table 1. Higher is better — this metric indicates whether users are engaging with the system to teach high-level abstractions; a flat curve means that the users have finished teaching and are exploiting the examples they have previously taught.

2. *Per-turn program complexity*: the number of actions generated per utterance. For example, an utterance that generates the program `GOTO Mug; PICKUP Mug; GOTO Sink; PUT Mug Sink` has complexity of 4 — one for each primitive (`NOT-SURE` counts at 0). We expect a steep upward trend in this metric over time as users teach and reuse progressively more complex examples.

3. *Normalized episode length*: the number of language utterances the user provided divided by the number of primitive actions required to solve the task. This is the end-to-end metric we seek to optimize — values less than 1 indicate that users are able to tap into what they have taught to complete tasks in fewer steps.

4.2 Results

Full Results: 20 Users x 7 Tasks. Figure 4 presents graphs of the three metrics over the 5 episodes for each of the 20 users, split across the 7 different task. Error bars denote estimated standard deviation across all 20 users. Users of both our exemplar-based system and the sequence-to-sequence baseline teach a moderate number of new examples over time, with an upwards trend in per-turn program complexity as they complete more tasks. Finally, we see a decreasing trend in the normalized episode length, with the mean value of our system dipping slightly below a value of 1 after completing 5 instances.

Case Study: *Pick, Cool, and Place*. Figure 5, on the other hand, presents graphs of the 3 metrics across 3 users for the *Pick, Cool, and Place* task, one of the more complex tasks in our suite, requiring at least 12 primitive utterances to complete successfully (compared to tasks like *Pick and Place* that only require 4). Here we see large gaps between our system and the sequence-to-sequence baseline — not only do users of our system teach significantly more high-level examples, but they have a much-higher per-turn program complexity after 5 episodes compared to the baseline. Finally,

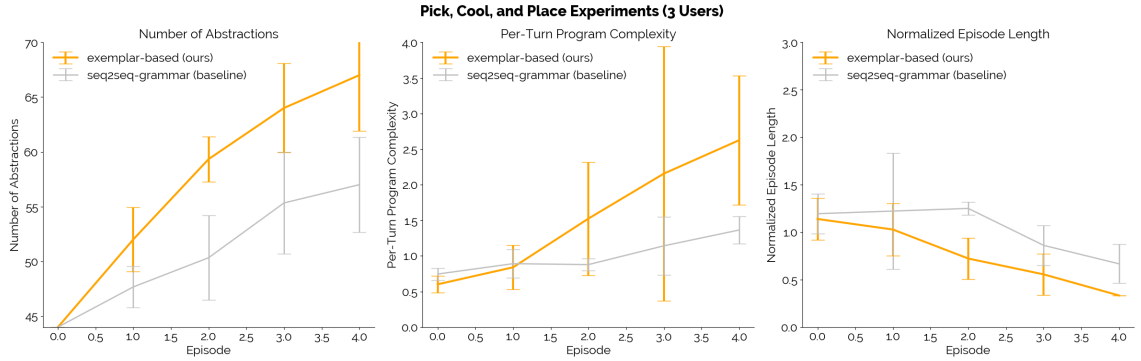

Figure 5: Results for the *Pick, Cool, and Place* task across 3 users (subset of the original 20). This task is complex, requiring at least 12 primitives to complete. Notice how the number of defined examples and per-turn program complexity are much higher for our method, and that the normalized episode length is lower.

we see that after 5 episodes, the normalized episode length is around 0.2, indicating that users are able to complete the complex task in 1/5 the steps necessary with our system.

Are users re-using high-level abstractions? The general results in Figure 4 indicate that while users are teaching the system new abstractions, they are unfortunately not re-using them effectively. The normalized episode length plot shows that both systems converge to 1, indicating that users are defaulting to the primitive actions, rather than trying to teach higher-level examples. One possible explanation for this is that for simpler tasks (e.g. *Pick and Place*), it is perhaps easier and faster to provide low-level utterances (those in Table 1), rather than teach new examples. Defaulting to low-level utterances also explains the lack of a significant gap between the sequence-to-sequence model and our model — in light of low-level utterances, the grammar does the heavy-lifting (in other words, we would not be invoking the sequence-to-sequence model at all). Indeed, across all 20 users for the *seq2seq-grammar* model, 89.9% of successfully parsed utterances (713 out of 793 total) were handled by the grammar, with only 10.1% handled by the seq2seq model (70 of 793 total).

However, this trend doesn't hold true for more complex tasks. Figure 5 shows that users are teaching and reusing a significant number of examples, completing tasks extremely efficiently. One hypothesis is to correlate task complexity with abstraction reuse (and thus, the ease by which users solve tasks), and while supported by the *Pick, Cool, and Place* results (Figure 5), we would require future experiments with a larger number of users before we can draw meaningful conclusions.

5 Related Work

We build on a long tradition of learning semantic parsers for mapping language to executable programs (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005, 2007; Liang et al., 2011), with a focus on using context and learning from interaction.

Contextual Semantic Parsing. In many settings, successfully parsing an utterance requires reasoning about both linguistic and environment context. Artzi and Zettlemoyer (2013) developed a model for parsing instructions in the SAIL Navigation dataset (MacMahon et al., 2006; Chen and Mooney, 2011) that leverages the environment context. Later, Long et al. (2016) introduced the SCONE Dataset, requiring building models that can reason over both types of context. More recently, Yu et al. (2019) introduced the large-scale Conversational Text-to-SQL (CoSQL) dataset that requires jointly reasoning over dialogue history and databases to parse user queries to SQL. We handle both linguistic context and environment context in our work, by decoupling semantic parsing from grounding; our lifted semantic parser handles linguistic context, while our entity resolver and reranker handle environment context.

Learning from Interaction. Closest to our work is Voxelurn (Wang et al., 2017), and its close predecessor SHRDLURN (Wang et al., 2016). Voxelurn defined an open-ended environment where the goal was to build arbitrary voxel structures using language instructions. We take inspiration from its teaching procedure where users decompose high-level utterances into low-level actions in the context

of a grammar-based parser. Other work uses alternative modes of interaction to teach new behaviors. Srivastava et al. (2017) used natural language explanations to teach new concepts. Relatedly, Labutov et al. (2018) introduced LIA, a programmable personal assistant that learned from user-provided condition-action rules. Furthermore, Weigelt et al. (2020) introduce an approach for teaching systems new programmatic functions from language that explicitly reasons about whether utterances contain "teaching intents," a mechanism that is similar to our procedure for returning `NOT-SURE`. Once these "teaching intents" have been identified, they are parsed into corresponding code blocks that can then be executed. Other work leverages conversations to learn new concepts, generating queries for users to respond to (Artzi and Zettlemoyer, 2011; Thomason et al., 2019). Notably, Thomason et al. (2019) used this conversational structure in a robotics setting similar to ours, but focused on learning new percepts, rather than structural abstractions. Yao et al. (2019) defined a similar conversational system for Text-to-SQL models that decides when intervention is needed, and generates a clarification question accordingly.

General Instruction Following. Other work looks at instruction following for robotics tasks outside the semantic parsing paradigm, for example by mapping language directly to sequences of actions (Anderson et al., 2018; Fried et al., 2018; Shridhar et al., 2020), mapping language to representations of reward functions (Arumugam et al., 2017; Karamcheti et al., 2017), or learning language-conditioned policies via reinforcement learning (Hermann et al., 2017; Chaplot et al., 2018).

6 Discussion & Lessons Learned

Towards More Complex Settings. Our analysis in Section 4.2 suggests that situating our system in a more complex setting might allow us to truly see the benefits of learning by decomposition. One such setting is Voxelurn (Wang et al., 2017), with its open-ended tasks that allow for the definition of multiple different high-level abstractions with compositional richness. In contrast, the tasks in this work are linear, with similar sequences of primitives used to accomplish each high-level task.

Future work should use this insight and identify environments that are more complex and open-ended, where users are naturally incentivized to teach the system new abstractions that built atop each other, to facilitate performing more complex behaviors. In robotics, this might translate to building systems for cooking, perhaps taking inspiration from Epic Kitchens (Damen et al., 2018), where the set of high-level objectives (general recipes to follow, kitchen behaviors to imitate) is much larger, but where individual subtasks (low-level abstractions like slicing a vegetable, stirring a pot) are very common and generalizable. Other settings might include open-ended building tasks, either in the real world (Knepper et al., 2013; Lee et al., 2019), or in virtual worlds like Minecraft (Johnson et al., 2016; Gray et al., 2019).

On Trusting Interactive Learning. Users have an implicit expectation that after providing just a single example — say to "wash the coffee mug" — the system will know how to "wash the tomato" or even "clean the plate" immediately. However, existing machine learning is not built with such extreme data efficiency in mind; especially for harder types of generalization (e.g. to "clean the plate"), we cannot guarantee learning this in a single step. While in this work we show reliable one-shot generalization across objects in a simplified setting, the real-world is much more complex, and different entities merit different behaviors. For example, consider generalizing from "wash the spoon" to "wash the table"; a system like ours will try to execute the program taught in the first context (going to the sink, placing the object inside, etc.) to the second, leading to complete failure.

Part of the problem is a *lack of transparency*; after teaching an example, it is hard for a user to understand what the system knows. This impacts trust, and as a result, when the system makes a mistake interpreting a high-level utterance, users back off to using utterances they are confident the system will understand (mirroring our observed results). This suggests future work in building more reliable methods for one-shot generalization and interpretability, providing users with a clear picture of what the model has learned.

Acknowledgements

This work was supported by NSF Award Grant no. 2006388 and the Future of Life Institute. S.K. is supported by the Open Philanthropy Project AI Fellowship. We thank Robin Jia, Michael Xie, John Hewitt, and Chris Potts for helpful feedback during the initial stages of this work. We finally thank the anonymous reviewers for their helpful comments.

References

P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Computer Vision and Pattern Recognition (CVPR)*.

Y. Artzi and L. Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 421–432.

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.

D. Arumugam, S. Karamcheti, N. Gopalan, L. L. S. Wong, and S. Tellex. 2017. Accurately and efficiently interpreting human-robot instructions of varying granularities. In *Robotics: Science and Systems (RSS)*.

D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. 2018. Gated-attention architectures for task-oriented language grounding. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

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.

D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray. 2018. Scaling egocentric vision: The EPIC-KITCHENS dataset. In *European Conference on Computer Vision (ECCV)*.

L. Dong and M. Lapata. 2016. Language to logical form with neural attention. In *Association for Computational Linguistics (ACL)*.

D. Fried, R. Hu, V. Cirik, A. Rohrbach, J. Andreas, L. Morency, T. Berg-Kirkpatrick, K. Saenko, D. Klein, and T. Darrell. 2018. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

J. Gray, K. Srinet, Y. Jernite, H. Yu, Z. Chen, D. Guo, S. Goyal, C. L. Zitnick, and A. Szlam. 2019. Craftassist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584*.

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)*.

K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom. 2017. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. 2016. The malmo platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

S. Karamcheti, E. C. Williams, D. Arumugam, M. Rhee, N. Gopalan, L. L. S. Wong, and S. Tellex. 2017. A tale of two draggns: A hybrid approach for interpreting action-oriented and goal-oriented instructions. In *First Workshop on Language Grounding for Robotics @ ACL*.

R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. 2013. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *International Conference on Robotics and Automation (ICRA)*, pages 855–862.

P. Koehn and R. Knowles. 2017. Six challenges for neural machine translation. In *NMT@ACL*.

E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. 2017. Ai2-thor: An interactive 3d environment for visual AI. *arXiv preprint arXiv:1712.05474*.

I. Labutov, S. Srivastava, and T. M. Mitchell. 2018. Lia: A natural language programmable personal assistant. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim. 2019. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*, pages 590–599.

R. Long, P. Pasupat, and P. Liang. 2016. Simpler context-dependent logical forms via model projections. In *Association for Computational Linguistics (ACL)*.

M. MacMahon, B. Stankiewicz, and B. Kuipers. 2006. Walk the talk: Connecting language, knowledge, and action in route instructions. In *National Conference on Artificial Intelligence*.

S. Mussman, R. Jia, and P. Liang. 2020. On the importance of adaptive data collection for extremely imbalanced pairwise tasks. In *Findings of Empirical Methods in Natural Language Processing (Findings of EMNLP)*.

N. Papernot and P. McDaniel. 2018. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*.

J. Pennington, R. Socher, and C. D. Manning. 2014. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

B. Scassellati, H. Admoni, and M. Mataric. 2012. Robots for use in autism research. *Annual review of biomedical engineering*, 14:275–294.

M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Computer Vision and Pattern Recognition (CVPR)*.

S. Srivastava, I. Labutov, and T. Mitchell. 2017. Joint concept learning and semantic parsing from natural language explanations. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1528–1537.

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)*.

J. Thomason, A. Padmakumar, J. Sinapov, N. Walker, Y. Jiang, H. Yedidsion, J. W. Hart, P. Stone, and R. J. Mooney. 2019. Improving grounded natural language understanding through human-robot dialog. In *International Conference on Robotics and Automation (ICRA)*.

J. Thomason, S. Zhang, R. J. Mooney, and P. Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

S. I. Wang, S. Ginn, P. Liang, and C. D. Manning. 2017. Naturalizing a programming language via interactive learning. In *Association for Computational Linguistics (ACL)*.

S. I. Wang, P. Liang, and C. Manning. 2016. Learning language games through interaction. In *Association for Computational Linguistics (ACL)*.

Y. Wang, J. Berant, and P. Liang. 2015. Building a semantic parser overnight. In *Association for Computational Linguistics (ACL)*.

S. Weigelt, V. Steurer, T. Hey, and W. Tichy. 2020. Programming in natural language with fuse: Synthesizing methods from spoken utterances using deep natural language understanding. In *Association for Computational Linguistics (ACL)*.

Z. Yao, Y. Su, H. Sun, and W. Yih. 2019. Model-based interactive semantic parsing: A unified framework and a text-to-SQL case study. In *Empirical Methods in Natural Language Processing (EMNLP)*.

T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. R. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. S. Lasecki, and D. R. Radev. 2019. Cosql: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Empirical Methods in Natural Language Processing (EMNLP)*.

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. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

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.