Interactive Semantic Parsing for If-Then Recipes via Hierarchical Reinforcement Learning

Ziyu Yao[⋆] **Xiujun Li**[†§] **Jianfeng Gao**[†] **Brian Sadler**[‡] **Huan Sun**[⋆]
[⋆]The Ohio State University [§]University of Washington
[†]Microsoft Research AI [‡]U.S. Army Research Lab
{yao.470,sun.397}@osu.edu {xiul,jfgao}@microsoft.com
brian.m.sadler6.civ@mail.mil

Abstract

Given a text description, most existing semantic parsers synthesize a program in one shot. However, it is quite challenging to produce a correct program solely based on the description, which in reality is often ambiguous or incomplete. In this paper, we investigate *interactive semantic parsing*, where the agent can ask the user clarification questions to resolve ambiguities via a multi-turn dialogue, on an important type of programs called "If-Then recipes." We develop a hierarchical reinforcement learning (HRL) based agent that significantly improves the parsing performance with minimal questions to the user. Results under both simulation and human evaluation show that our agent substantially outperforms non-interactive semantic parsers and rule-based agents.[1]

1 Introduction

Semantic parsing aims to map natural language to formal domain-specific meaning representations, such as knowledge base or database queries (Berant et al. 2013; Dong and Lapata 2016; Zhong, Xiong, and Socher 2017; Gao, Galley, and Li 2018), API calls (Campagna et al. 2017; Su et al. 2017) and general-purpose code snippets (Yin and Neubig 2017; Rabinovich, Stern, and Klein 2017). In this work, we focus on semantic parsing for synthesizing a simple yet important type of conditional statements called *If-Then recipes* (or *If-Then programs*), based on a natural language description (Quirk, Mooney, and Galley 2015; Beltagy and Quirk 2016; Liu et al. 2016; Yin and Neubig 2017; Chaurasia and Mooney 2017). For example, the description "Create a link note on Evernote for my liked tweets" should be parsed into an If-Then recipe with 4 components: *trigger channel* `Twitter`, *trigger function* `New liked tweet by you`, *action channel* `Evernote`, and *action function* `Create a link note`. On the one hand, If-Then recipes allow users to perform a large variety of tasks such as home security ("text me if the door is not locked"). On the other hand, developing intelligent agents that can automatically parse these recipes is an important step towards complex natural language programming (Quirk, Mooney, and Galley 2015).

[1]All source code and documentations are available at https://github.com/LittleYUYU/Interactive-Semantic-Parsing.

User: "record to evernote" **Ground-truth recipe**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`] **(Liu et al. 2016)**: [tc: `Phone Call`, tf: `Leave IFTTT any voicemail`, ac: `Evernote`, af: `Append to note`]
User: "record to evernote" **HRL-fixedOrder agent**: "Which channel triggers the action?" **User**: "twitter" **HRL-fixedOrder agent**: "Which event triggers the action?" **User**: "If I like a tweet" **HRL-fixedOrder agent**: "Which event results from the trigger?" **User**: "Create a note with link" **Agent Prediction**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`]
User: "record to evernote" **HRL agent**: "Which event triggers the action?" **User**: "If I like a tweet" **HRL agent**: "Which event results from the trigger?" **User**: "Create a note with link" **Agent Prediction**: [tc: `Twitter`, tf: `New liked tweet by you`, ac: `Evernote`, af: `Create a link note`]

Table 1: Semantic parsers on an ambiguous description: The state-of-the-art non-interactive model (Liu et al. 2016) cannot correctly parse the recipe while our two HRL-based interactive agents can. Particularly, by coordinating the subtask order, the HRL agent asks fewer questions than the HRL-fixedOrder agent (tc: trigger channel, tf: trigger function, ac: action channel: af: action function).

Most previous work translates a natural language description to an If-Then recipe *in one turn*: The user gives a recipe description and the system predicts the 4 components. However, in reality, a natural language description can be very noisy and ambiguous, and may not contain enough information. For simplicity, we refer to this problem as *description ambiguity*. In fact, in the widely used If-Then evaluation dataset (Quirk, Mooney, and Galley 2015), **80%** of the descriptions are ambiguous. As shown in Table 1, the description "record to evernote" is paired with the same ground-truth recipe as in the first example, but even humans cannot tell what the "record" refers to (i.e., trigger channel/function) and what kind of note to create on Evernote

(i.e., action function). Therefore, it is quite challenging, if not impossible, to produce a correct program in one shot merely based on an ambiguous description.

Driven by this observation, we investigate *interactive semantic parsing*, where an intelligent agent (e.g., the two HRL-based agents in Table 1) strives to improve the parsing accuracy by asking clarification questions. Two key challenges are addressed: (1) Lack of supervision on when to ask a question. To date, there is no large-scale annotated dataset on whether and when an agent should ask a question during parsing. The only feedback an agent can obtain is whether or not a synthesized program is correct. (2) How to improve the parsing accuracy with a minimal number of questions? To guarantee a good user experience, the agent should only ask "necessary" questions and learn from human interactions over time.

Previous work (Chaurasia and Mooney 2017) developed rule-based agents to interactively predict the 4 components of an If-Then recipe. These agents decide to ask a question when the prediction probability of a recipe component is lower than a predefined threshold. However, such rule-based agents are not trained in an optimization framework to simultaneously improve the parsing accuracy and reduce the number of questions.

We address these challenges via a Hierarchical Reinforcement Learning (HRL) approach. We formulate the interactive semantic parsing in the framework of *options* over Markov Decision Processes (MDPs) (Sutton, Precup, and Singh 1999), where the task of synthesizing an If-Then recipe is naturally decomposed into 4 *subtasks* or *options* (i.e., predicting trigger/action channel/function). In particular, we propose an HRL agent with a hierarchical policy: A high-level policy decides the order of the subtasks to work on, and a low-level policy for each subtask guides its completion by deciding whether to (continue to) ask a clarification question or to predict the subtask component. We train the policies to maximize the parsing accuracy and minimize the number of questions with the rewarding mechanism, where the only supervision (reward signal) is whether or not a predicted component is correct.

Compared with the approach of solving the entire task with one flat policy (Mnih et al. 2015), HRL takes advantage of the naturally defined "4-subtask" structure. Such design also allows the agent to focus on different parts of a recipe description for each subtask, as emphasized in (Liu et al. 2016), and endows each low-level policy with a reduced state-action space to simplify the learning. On the other hand, the high-level policy optimizes the subtask order by taking into account both the recipe description and user responses. As shown in Table 1, the HRL agent learns to ask about the trigger function first, to which the user response (i.e., "tweet") implies the trigger channel. This mechanism leads to fewer questions than the HRL-fixedOrder agent, which executes subtasks in the fixed order of "tc-tf-ac-af".

Experimental results under both simulation and human evaluation show that our HRL agent can obtain a significantly better accuracy while asking fewer questions than rule-based agents like (Chaurasia and Mooney 2017). In addition, we show the effectiveness of the high-level policy on reducing the number of questions. Our agent tends to predict functions before channels, which is different from most existing works that either assume independence among subtasks (Chaurasia and Mooney 2017) or predict channels before functions (Beltagy and Quirk 2016; Dong and Lapata 2016; Yin and Neubig 2017).

2 Background

If-Then recipes allow people to manage a variety of web services or physical devices and automate event-driven tasks. We focus on recipes from IFTTT.com, where a recipe has 4 components: *trigger channel*, *trigger function*, *action channel*, and *action function*. There are a variety of channels, like `GMail` and `Facebook`, representing entities such as web applications and IFTTT-provided services. Each channel has a set of functions representing events (i.e., trigger functions) or action executions (i.e., action functions). In one recipe, there could be exactly one value for trigger/action channel/function.

Following (Liu et al. 2016; Chaurasia and Mooney 2017), we decompose the task of parsing a natural language description into four subtasks, i.e., predicting trigger/action channel/function in a recipe. We have made two key observations about real-life recipe descriptions and existing semantic parsing work: (1) Around 80% recipe descriptions are ambiguous or contain incomplete information, according to the human annotations provided by Quirk, Mooney, and Galley (2015), which makes it extremely difficult to synthesize a recipe in one turn (see the prediction result of (Liu et al. 2016) in Table 1). Therefore, (Liu et al. 2016; Chaurasia and Mooney 2017; Yin and Neubig 2017) focus on the unambiguous 20% recipes for evaluation. (2) Previous work assumes either independence (Chaurasia and Mooney 2017) or heuristic dependencies among the 4 subtasks. In particular, Liu et al. (2016) assumes that functions should be predicted before channels since a channel can be derived from the function prediction, while (Beltagy and Quirk 2016; Dong and Lapata 2016; Yin and Neubig 2017) assume that channels should be predicted before functions and triggers before actions.

Given these observations, we propose an *interactive semantic parser* that can ask users for clarification to make more accurate predictions. Moreover, we abandon the inter-subtask (in)dependence assumptions used in previous work. Our agent learns to optimize the subtask order for each recipe to save questions.

3 Interactive Semantic Parser

3.1 Framework Overview

Given a recipe description, four components need to be predicted. Thus, the semantic parsing task can be naturally decomposed into four subtasks $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$, standing for predicting the trigger channel (st_1), trigger function (st_2), action channel (st_3) and action function (st_4), respectively. We aim at an agent that can decide the order of subtasks for each parsing task, and only moves to the next subtask when the current one has been completed. We formulate this as a hierarchical decision making problem based on

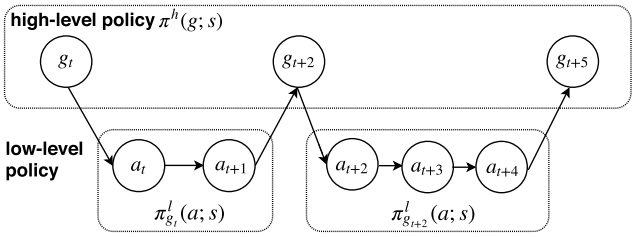

Figure 1: Hierarchical policy: The high-level policy chooses a subtask to work on while the low-level policy decides to predict the subtask or ask the user at each time step.

the framework of *options* over Markov Decision Processes (MDPs) (Sutton, Precup, and Singh 1999).

Specifically, the agent uses a hierarchical policy consisting of two levels of policies operating at different time scales. The high-level policy selects the next subtask (or *option*) to work on, which can be viewed as operating on a Semi-MDP (Sutton, Precup, and Singh 1999). The low-level policy selects primitive actions (i.e., predicting a component value or querying the user) to complete the selected subtask. As elaborated in Section 3.2, we adopt 4 low-level policies, each for one subtask.

Figure 1 shows the process. At an *eligible* time step t (i.e., *at the beginning of a parsing task or when a subtask terminates*), the high-level policy $\pi^h(g; s_t)$ receives a state s_t and selects a subtask $g_t \in \mathcal{G}$ to work on. Then the low-level policy $\pi^l_{g_t}(a; s_t)$ for this subtask chooses an action $a_t \in \mathcal{A}_{g_t}$. By taking this action, the agent receives a low-level reward $r^l_{g_t}(s_t, a_t)$. For the next N time steps (e.g., $N = 2$ in Figure 1), the agent will work on the same subtask g_t until it terminates (i.e., when either the agent has predicted the corresponding component or the agent has interacted with the user for Max_Lobal_Turn turns). The agent will receive a high-level reward $r^h(s_t, g_t)$ for this subtask completion, then select the next subtask g_{t+N} and repeat the above procedure until the entire task terminates (i.e., when either all four components are predicted or the agent has worked on Max_Global_Turn subtasks).

States. A state s tracks 9 items during the course of interactive parsing:

- The initial recipe description $\mathcal{I}$.
- The boolean indicator b_i $(i = 1, 2, 3, 4)$ showing whether subtask st_i has been predicted.
- The received user answer d_i $(i = 1, 2, 3, 4)$ for subtask st_i, respectively.

For each subtask st_i, we learn a *low-level state vector* $s^l_{st_i}$ to summarize state information of this subtask for low-level policy $\pi^l_{st_i}$ to select the next action. Similarly, a *high-level state vector* s^h is learned to present a summary of the entire state, consisting of the 4 low-level state vectors and other state information, for high-level policy π^h to choose the next subtasks. Section 3.2 details how states are represented.

Actions. The action space for the high-level policy is $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$, where each action denotes one subtask mentioned earlier. The action space of subtask g is $\mathcal{A}_g = \mathcal{V}_g \cup \{\text{AskUser}\}$, where $\mathcal{V}_g$ is the set of available component values for subtask g (e.g., all trigger channels for subtask st_1), meaning that the agent can either predict a component or ask the user a clarification question.

Rewards. At each eligible time step t, the agent selects a subtask $g_t \sim \pi^h(g; s_t)$ and receives a high-level reward $r^h(s_t, g_t)$ when the subtask terminates after N steps. The high-level reward will be used to optimize the high-level policy via RL. We define $r^h(s_t, g_t)$ as the accumulated low-level rewards from time step t to $t + N$. For intermediate time steps (during which the agent works on a selected subtask), there is no high-level reward.

$$r^h(s_t, g_t) = \begin{cases} \sum_{k=t}^{t+N} r^l_{g_t}(s_k, a_k) & \text{for eligible } t \\ 0 & \text{otherwise} \end{cases}$$

While working on subtask g_t, the agent receives a low-level reward for taking action a_t (i.e., predicting g_t or querying the user):

$$r^l_{g_t}(s_t, a_t) = \begin{cases} 1 & \text{if } a_t = \ell_{g_t} \\ -\beta & \text{if } a_t = \text{AskUser} \\ -1 & \text{otherwise} \end{cases}$$

where ℓ_{g_t} is the ground-truth label for subtask g_t and $\beta \in [0, 1)$ is the penalty for querying the user. The received reward will be used to optimize the low-level policy $\pi^l_{g_t}$ for this subtask via RL.

Essentially, the low-level reward r^l_g alleviates the reward sparsity in the long trajectory of the entire task, and stimulates the agent to predict a correct component with fewer questions. Note that during the course of RL we do not stop the parsing even if one of the predictions is incorrect in order to encourage the agent to predict as more correct components as possible. This also fits the realistic application setting where the agent does not know the ground truth at the component level, and does not receive the external reward signal until it recommends a synthesized program to the user at the end of the interactive parsing process.

Transition. Interactive semantic parsing starts with a state s_0 where $b_i = 0$ (i.e., no subtask is completed) and $d_i = \varnothing$ (i.e., no user answer). As the agent takes actions, it deterministically transits to a new state with updated b_i and d_i.

3.2 Hierarchical Policy Functions

Low-level policy function. The low-level policy $\pi^l_{st_i}(a; s)$ decides whether to ask a question or predict subtask st_i (e.g., selecting a trigger channel name for st_1). When it selects the "AskUser" action, the user will clarify the subtask with a natural language utterance as shown in Table 1. The policy then decides the next action by considering both the recipe description and the user response.

To understand a recipe description, we choose one of the state-of-the-art models, Latent Attention Model (LAM) (Liu et al. 2016). The main idea behind LAM is to first understand the latent sentence structure and then pay attention to words that are critical for a subtask. For example, given a description with a pattern "X to Y," LAM adopts a latent attention

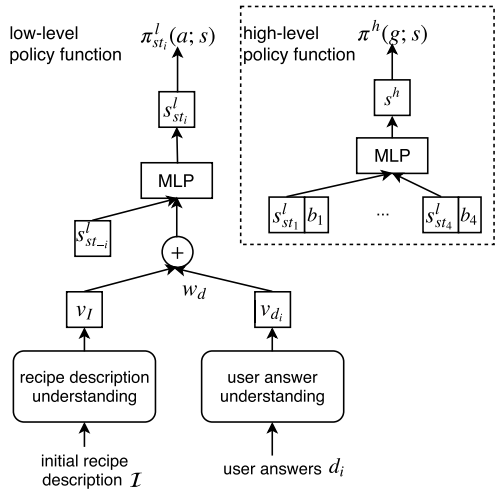

Figure 2: High- and low-level policy designs.

mechanism to first locate the keyword "to," and then focus more on "X" when predicting the trigger channel/function and on "Y" when predicting the action channel/function. This reveals that different subtasks need different policies, so that they can focus on different parts of a recipe description. Such design also allows each subtask to have a different and reduced action space. Hence, we define 4 low-level policy functions with the same model structure yet different parameters for the 4 subtasks, respectively.

To deal with user responses, one straightforward adaptation from LAM is to simply concatenate them with the initial recipe description as the input and extend the prediction space with an extra "AskUser" action. However, we observe that user answers are fundamentally different from a recipe description, e.g., user answers typically contain information related to the queried subtask (rather than all subtasks). Therefore, we propose to model these two parts separately as shown in Figure 2. In particular, we employ LAM to instantiate the "*recipe description understanding*" module to capture the meaning of the initial recipe description (i.e., v_I), and utilize a bidirectional GRU-RNN with the attention mechanism (Zhou et al. 2016) to instantiate the "*user answer understanding*" module.[2] Details can be found in the Appendix A. Since each subtask maintains a separate policy function, the learned v_I can be different for different subtasks. The agent can ask the user to clarify a subtask for multiple times,[3] and the newly received answers will be concatenated with the old ones to update d_i.

We combine the representations of recipe description (v_I) and user answers (v_{d_i}) as the semantic representation related to subtask st_i:

$$v_i = (1 - w_d)v_I + w_d v_{d_i}, \tag{1}$$

where $w_d \in [0, 1]$ is a weight controlling information from the initial recipe description versus from user answers. The semantic vector v_i, concatenated with the information of other subtasks, defines the *low-level state vector* $s^l_{st_i}$ of subtask st_i via a multi-layer perceptron model (the "MLP" module in Figure 2):

$$s^l_{st_i} = \tanh(W_{c_i}[s^l_{st_1}; ...; s^l_{st_{i-1}}; v_i; s^l_{st_{i+1}}; ...; s^l_{st_4}]).^4 \tag{2}$$

Essentially, $s^l_{st_i}$ summarizes the state information for completing subtask st_i, including the initial recipe description, user answers for st_i, and the current low-level state vectors of other subtasks (such that the completion status of other subtasks can affect the current one, as shown in Table 1). Finally, the low-level policy function $\pi^l_{st_i}(a; s)$=softmax$(W^l_{st_i} s^l_{st_i})$, takes the state vector as the input, and outputs a probability distribution over the action space of subtask st_i.

High-level policy function. The high-level policy $\pi^h(g; s)$ receives a state s and decides the next subtask g. The *high-level state vector* s^h is learned to encode the state of overall parsing task through a multi-layer perceptron model (i.e., the "MLP" module in Figure 2) using the 4 low-level state vectors $s^l_{st_i}$'s, as well as the subtasks' boolean indicators b_i's, as inputs:

$$s^h = \tanh(W_c[s^l_{st_1}; b_1; s^l_{st_2}; b_2; s^l_{st_3}; b_3; s^l_{st_4}; b_4]),$$
$$\pi^h(g; s) = \text{softmax}(W^h s^h). \tag{3}$$

Optimization. The high-level policy π^h is trained to maximize the expectation of the discounted cumulative rewards for selecting subtask g_t in state s_t:

$$\max_{\pi^h} J(\theta) = \max_{\pi^h} E_{\pi^h}[r^h(s_t, g_t) \tag{4}$$
$$+ \gamma r^h(s_{t+N_1}, g_{t+N_1}) + \gamma^2 r^h(s_{t+N_1+N_2}, g_{t+N_1+N_2})$$
$$+ ... + \gamma^\infty r^h(s_{t+\sum_{n=1}^{\infty} N_n}, g_{t+\sum_{n=1}^{\infty} N_n})|s_t, g_t, \pi^h(\theta)],$$

where θ stands for parameters in π^h, $N_n (n = 1, 2, ..., \infty)$ is the number of time steps that the agent spent on the previous subtask, and $\gamma \in [0, 1]$ is the discount factor. Similarly, we train the low-level policy $\pi^l_{g_t}$ for the selected subtask g_t to maximize its expected cumulative discounted low-level reward:

$$\max_{\pi^l_{g_t}} J_{g_t}(\phi_{g_t}) = \max_{\pi^l_{g_t}} E_{\pi^l_{g_t}}[\tag{5}$$
$$\sum_{k \geq 0} \gamma^k r^l_{g_t}(s_{t+k}, a_{t+k})|s_t, a_t, g_t, \pi^l_{g_t}(\phi_{g_t})],$$

where ϕ_{g_t} denotes the parameters in $\pi^l_{g_t}$, and γ is the same discount factor.

All policies are stochastic in that the next subtask or action is sampled according to the probability distribution which allows exploration in RL, and that the policies can be optimized using policy gradient methods. In our experiments we used the REINFORCE algorithm (Williams 1992). Details are outlined in Algorithm 1 of the Appendix.

4 Experiments

We experiment with our proposed HRL agent under both simulation and human evaluation.

[2] We also verified this separate design is much better than the straightforward adaptation from LAM mentioned earlier during model development.

[3] Using the same predefined question (see User Simulation).

[4] Bias terms are omitted for clarity.

Test Data	CI	VI		Total
		VI-1/2	VI-3/4	
Size	727	1,271	1,872	3,870
(%)	(18.79)	(32.84)	(48.37)	(100)

Table 2: Statistics of the test subsets.

4.1 Dataset

We utilize the 291,285 <recipe, description> pairs collected by Ur et al. (2016) for training and the 3,870 pairs from Quirk, Mooney, and Galley (2015) for testing.[5] 20% of the training data are randomly sampled as a validation set. All recipes were created by real users on IFTTT.com. In total, the datasets involve 251 trigger channels, 876 trigger functions, 218 action channels and 458 action functions. For each description in the test set, Quirk, Mooney, and Galley (2015) collected five recipe annotations from Amazon Mechanical Turkers. For each subtask, if at least three annotators make the same annotation as the ground truth, we consider this recipe description as *clear* for this subtask; otherwise, it is labeled as *vague*. In this way, we split the entire test set into three subsets as shown in Table 2: (1) *CI*: 727 recipes whose descriptions are clear for all 4 subtasks; (2) *VI-1/2*: 1,271 recipes containing 1 or 2 vague subtasks; (3) *VI-3/4*: 1,872 recipes containing 3 or 4 vague subtasks.

4.2 Methods Comparison

- **LAM**: The Latent Attention Model (Liu et al. 2016),[6] one of the state-of-the-art models for synthesizing If-Then recipes. We do not consider the model ensemble in (Liu et al. 2016), as it can be applied to all other methods as well. Our reproduced LAM obtains a performance close to the reported one without ensemble.
- **LAM-rule Agent**: A rule-based agent built on LAM, similar to (Chaurasia and Mooney 2017). Specifically, the agent makes a prediction on a subtask with a certain probability. If the probability is lower than a threshold,[7] the user is asked a question. The user answer is concatenated with the initial recipe description for making a new prediction. This procedure repeats until the prediction probability is greater than the threshold or the agent has run for Max_Local_Turn turns on the subtask.
- **LAM-sup Agent**: An agent based on LAM, but with the user answer understanding module in Figure 2 and an extra "AskUser" action for each subtask. It is trained via a supervised learning strategy and thus is named LAM-sup. We collected the training data for each subtask st_i based on the LAM-rule agent: If LAM-rule completes the subtask without interactions with humans, we add a tuple $<\mathcal{I}, \varnothing, \ell_{st_i}>$ to the training set, where $\mathcal{I}$ is the recipe description and ℓ_{st_i} is the ground-truth label for subtask st_i; otherwise, we add two tuples $<\mathcal{I}, \varnothing,$ "AskUser"$>$ and $<\mathcal{I}, d_i, \ell_{st_i}>$, where d_i is the received user answer. We train the agent by minimizing the cross entropy loss. During testing, for each recipe description, the agent starts with no user answer; for each subtask, if it predicts the "AskUser" label, the received user answer will be concatenated with previous ones to make a new prediction until a non-AskUser label is selected.
- **HRL Agent**: Our agent with a two-level hierarchical policy.
- **HRL-fixedOrder Agent**: A variant of our HRL agent with a fixed subtask order of "st_1-st_2-st_3-st_4" and no high-level policy learning.

Evaluation metrics. We compare each method on three metrics: (1) *C+F Accuracy*: A recipe is considered parsed correctly only when all its 4 components (i.e., Channel+Function) are accurately predicted, as adopted in (Quirk, Mooney, and Galley 2015; Liu et al. 2016). (2) *Overall Accuracy*: We measure the average correctness of predicting 4 components of a recipe, e.g., the overall accuracy for predicting 3 components correctly and 1 incorrectly is 0.75. (3) *#Asks*: The averaged number of questions for completing an entire task. Generally, the *C+F Accuracy* is more challenging as it requires no mistake on any subtask. On the other hand, *#Asks* can reveal if an agent asks redundant questions. In our experiments, we consider *C+F Accuracy* and *#Asks* as two primary metrics.

Implementation details. The word vector dimension is set at 50, the weight factor w_d is 0.5, and the discount factor γ is 0.99. The max turn Max_Local_Turn is set at 5 and Max_Global_Turn at 4, which allows four subtasks at most. β is a trade-off between parsing accuracy and the number of questions: A larger β trains an agent to ask fewer questions but with less accuracy, while a lower β leads to more questions and likely more accurate parses. With the validation set, we experimented with β=$\{0.3, 0.4, 0.5\}$, and observed that when $\beta = 0.3$, the number of questions raised by the HRL-based agents is still reasonable compared with LAM-rule/sup, and its parsing accuracy is much higher. More details are shown in Appendix C.

4.3 User Simulation

In our work, for each subtask, agent questions are predefined based on templates,[8] and a user answer is a natural language description about the queried subtask. Given that it is too costly to involve humans in the training process, we introduce a user simulator to provide answers to agent questions.

For a trigger/action function, we adopt several strategies to simulate user answers, including revising its official description on IFTTT.com and replacing words and phrases in the function description with their paraphrases according to the PPDB paraphrase database (Pavlick et al. 2015).

[5] Quirk, Mooney, and Galley (2015) only released the urls of recipes in their test set, among which the unavailable ones have been removed from our test set. Each recipe is associated with a unique ID. We ensure no overlapping recipes between training and testing set by examining their IDs.

[6] Unlike (Liu et al. 2016), which consolidates channel and function names (e.g., "`Twitter.New liked tweet by you`") and builds 2 classifiers for trigger and action respectively, we develop 4 classifiers so that an agent can inquire channel or function separately.

[7] We set it at 0.85 based on validation set.

[8] Automatically generating recipe-specific questions is non-trivial, which we leave for future work.

Model	Simulation Eval									Human Eval	
	All			CI		VI-1/2		VI-3/4		VI-3/4	
	C+F Acc	Overall Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks	C+F Acc	#Asks
LAM	0.374	0.640	0	0.801	0	0.436	0	0.166	0	0.206	0
LAM-rule	0.761	0.926	3.891	0.897	1.433	0.743	2.826	0.721	5.568	0.518	2.781
LAM-sup	0.809	0.940	**2.028**	0.894	**0.684**	0.803	**1.482**	0.780	2.921	0.433	2.614
HRL-fixedOrder	0.881	0.966	2.272	**0.950**	1.522	0.855	1.958	0.871	2.777	0.581	2.306*
HRL	**0.894***	**0.968**	2.069*	0.949	1.226*	**0.888***	1.748*	**0.878***	**2.615***	**0.634***	**2.221***

Table 3: Model evaluation on the test set. For Simulation Eval, each number is averaged over 10 runs. For Human Eval, the LAM result is calculated on the sampled 496 recipes. * denotes significant difference in mean between HRL-fixedOrder vs. HRL in Simulation Eval and between HRL-based agents vs. {LAM-rule, LAM-sup} agents in Human Eval ($p < 0.05$).

In addition, we extract user descriptions of a function from our training set, based on six manually defined templates. For example, for a recipe description following pattern "If X then Y," X will be considered as an answer to questions about the ground-truth trigger function and Y as an answer to those about the ground-truth action function. More details regarding the strategies are in Appendix and source code will also be released. For each function, we collected around 20 simulated user answers on average. In our simulations, for each question we randomly select one from this set of possible answers as a response.

For trigger/action channels, when an agent asks a question, the user simulator will simply provide the channel name (e.g., `GMail`), since it is straightforward and natural for real users as well.

4.4 Simulation Evaluation

Table 3 shows results on the test set in the simulation environment, where our user simulator provides an answer when requested. By enabling the user to clarify, all agents obtain much better accuracy compared with the original non-interactive LAM model. In particular, HRL-based agents outperform others by **7%** $\sim$ **13%** on C+F accuracy and **2%** $\sim$ **4%** in terms of Overall accuracy. For vague recipes in VI-1/2 and VI-3/4 subsets, which make up more than 80% of the entire test set, the advantage of HRL-based agents is more prominent. For example, on VI-3/4, HRL-based agents obtain 9% $\sim$ 15% better C+F accuracy than LAM-rule/sup, yet with fewer questions, indicating that they are much more able to handle ambiguous recipe descriptions.

Compared with HRL-based agents, the LAM-rule agent usually asks the most questions, partly because it relies on heuristic thresholding to make decisions. On VI-3/4, it asks twice the number of questions but parses with 15% less accuracy than HRL-based agents. On the other hand, the LAM-sup agent always asks the least questions, especially when the recipe description is relatively clear (i.e., CI and VI-1/2). However, it may simply miss many necessary questions, leading to at least 5% accuracy loss.

Finally, we evaluate the high-level policy by comparing HRL with HRL-fixedOrder. The significance test shows that HRL requires fewer questions to obtain a similar or better accuracy. Interestingly, we observe that, under the interactive environment, HRL tends to predict the function before the channel, which is different from the inter-task (in)dependence assumptions in previous work. This is mainly because users' descriptions of a function can be more specific and may contain information about its channel. The HRL agent is thus trained to utilize this intuition for asking fewer questions.

4.5 Human Evaluation

We further conduct human evaluation to test the four interactive agents on the most challenging VI-3/4 subset. Two students familiar with IFTTT were instructed to participate in the test. For each session, a recipe from VI-3/4 and one agent from {LAM-rule, LAM-sup, HRL-fixedOrder, HRL} were randomly picked. The participants were presented the description and the ground-truth program components of the recipe, and were instructed to answer clarification questions prompted by the agent with a natural language sentence. To help the participants better understand the recipe, we also showed them the official explanation of each program component. However, we always encouraged them to describe a component in their own words when being asked. For a better user experience and an easier comparison, we limited each agent to ask at most one question for each subtask. In total, we collected 496 conversations between real humans and the four agents. Examples are shown in Appendix E.

We compare each agent primarily on *C+F Accuracy* and *#Asks*. As shown in Table 3, all agents perform much better than the non-interactive LAM model. Particularly, the HRL agent outperforms the LAM model by >40% accuracy, with an average of $\sim$2.2 questions on VI-3/4 (which is a reasonable number of questions as each task contains at least 3 vague subtasks). We also observe that the two HRL-based agents obtain 6% $\sim$ 20% better parsing accuracy with even fewer questions than the LAM-rule/sup agents. Moreover, in comparison with the HRL-fixedOrder agent, the HRL agent can synthesize programs with a much better accuracy but fewer questions, showing the benefit of optimizing subtask order at the high level. However, there is still large space to improve compared with simulation results in Table 3, mainly because agents are trained with the user simulator while the language used by real users for answers can be very different (e.g., having the Out-Of-Vocabulary issue, misspellings,

less or non- relevant information). How to simulate user responses as close as possible to real ones for training is a non-trivial task, which we leave for future work.

4.6 Discussion

Here we further discuss our framework and future work on the following aspects:

Error analysis. Due to the discrepancy between real users and simulated users, several major factors affect the performance of the HRL agent, including user typos (e.g., "emial" for "email") and unseen expressions (e.g., "i tweeted something" to describe the function `New tweet by you`). To improve the robustness of the HRL agent, possible solutions as future work can be modeling user noises in simulation (Li et al. 2016b), or crowdsourcing more diverse component descriptions as user answers for training.

Training with real users in the loop. Theoretically, our agents can be trained with real users. However, it is too costly to be practical because the agent can require many interactions during the training phase. An alternative way is to train the agent in simulation and fine-tune it with real users, or to build a *world model* that mimics real user behaviors during the human-in-the-loop training (Peng et al. 2018). Both approaches need significant efforts to be carefully designed, which we leave as future work.

Generalizability to other semantic parsing tasks. The proposed HRL framework can be easily generalized to resolve ambiguities in other semantic parsing tasks where subtasks can be pre-defined. For example, in the knowledge-graph-based question answering task (Berant et al. 2013; Yih et al. 2015), the subtasks include identifying entities, predicting relations, and associating constraints. To train the HRL agent, one can build a user simulator by paraphrasing the ground-truth entities or relations, similar to Section 4.3. HRL can be very promising for these tasks, as it enables temporal abstractions over the state and action space (leading to a smaller search space) and can model the dependencies between subtasks, as shown in Table 1. We will explore these applications in the future.

5 Related Work

In addition to the aforementioned work on If-Then program synthesis, Dong, Quirk, and Lapata (2018) investigated how to measure a semantic parser's confidence in its predictions, but did not further resolve uncertainties. Others include:

Interactive Systems for Resolving Ambiguities. Resolving ambiguities via interactions with humans has been explored in Natural Language Understanding in dialog systems (Thomason et al. 2015; Dhingra et al. 2017), Question Answering (Guo et al. 2016; Li et al. 2017), CCG parsing (He et al. 2016) and parsing for SQL and web APIs (Li and Jagadish 2014; Gur et al. 2018; Su et al. 2018). Guo et al. (2016) built an agent to ask relevant questions until it has enough information to correctly answer user's question, but expected the user to respond with an oracle value. He et al. (2016) investigated generating multi-choice questions for humans to resolve uncertainties in parsing sentences. They determined the necessity of a question by a heuristic threshold. In contrast, we allow users to respond with natural language utterances, and our HRL-based agents can learn when to ask through the reward mechanism. Recently, (Li et al. 2016a; Azaria, Krishnamurthy, and Mitchell 2016; Iyer et al. 2017) explored human feedback on the final results as training supervision. Different from theirs, we include humans during the parsing process for them to provide necessary information in natural language, and define rewards as the only feedback.

Hierarchical Reinforcement Learning (HRL). To solve a complex task, HRL decomposes the task into several easier subtasks and solve them sequentially via MDPs (Parr and Russell 1998; Sutton, Precup, and Singh 1999; Dietterich 2000). Recently, HRL-based approaches are applied to tasks like game playing (Kulkarni et al. 2016; Tessler et al. 2017), travel planning (Peng et al. 2017), and visual question answering and captioning (Wang et al. 2017; Gordon et al. 2017; Zhang, Zhao, and Yu 2018). Inspired by these work, given that our semantic parsing task can be naturally decomposed into 4 subtasks, we learn a two-level policy where a high-level policy decides the subtask order while a low-level policy accomplishes each subtask by asking humans clarifying questions if necessary.

6 Conclusion

In this paper we explored using HRL for *interactive semantic parsing*, where an agent asks clarification questions when the initially given natural language description is ambiguous and accomplishes subtasks in an optimized order. On the If-Then recipe synthesis task, in both simulation and human evaluation settings, we have shown that our HRL agent can substantially outperform various interactive baselines in that it produces more accurate recipes but asks the user fewer questions in general. As future work, we will generalize our HRL framework to other semantic parsing tasks such as knowledge based question answering, explore better training strategies such as modeling real user noises in simulation, as well as further reduce the user interaction turns.

Acknowledgments

We would like to thank Chris Brockett, Michel Galley and Yu Su for their insightful comments on the work. This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, Fujitsu gift grant, and Ohio Supercomputer Center (Center 1987). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Azaria, A.; Krishnamurthy, J.; and Mitchell, T. M. 2016. Instructable intelligent personal agent. In *AAAI*, 2681–2689.

Beltagy, I., and Quirk, C. 2016. Improved semantic parsers for if-then statements. In *ACL*, volume 1, 726–736.

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 1533–1544.

Campagna, G.; Ramesh, R.; Xu, S.; Fischer, M.; and Lam, M. S. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *WWW*, 341–350.

Center, O. S. 1987. Ohio supercomputer center. `http://osc.edu/ark:/19495/f5s1ph73`.

Chaurasia, S., and Mooney, R. J. 2017. Dialog for language to code. In *IJCNLP*, volume 2, 175–180.

Dhingra, B.; Li, L.; Li, X.; Gao, J.; Chen, Y.-N.; Ahmed, F.; and Deng, L. 2017. Towards end-to-end reinforcement learning of dialogue agents for information access. In *ACL*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research* 13:227–303.

Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *ACL*, volume 1, 33–43.

Dong, L.; Quirk, C.; and Lapata, M. 2018. Confidence modeling for neural semantic parsing. In *ACL*.

Gao, J.; Galley, M.; and Li, L. 2018. Neural approaches to conversational ai. *arXiv preprint arXiv:1809.08267*.

Gordon, D.; Kembhavi, A.; Rastegari, M.; Redmon, J.; Fox, D.; and Farhadi, A. 2017. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*.

Guo, X.; Klinger, T.; Rosenbaum, C.; Bigus, J. P.; Campbell, M.; Kawas, B.; Talamadupula, K.; Tesauro, G.; and Singh, S. 2016. Learning to query, reason, and answer questions on ambiguous texts.

Gur, I.; Yavuz, S.; Su, Y.; and Yan, X. 2018. Dialsql: Dialogue based structured query generation. In *ACL*, 1339–1349.

He, L.; Michael, J.; Lewis, M.; and Zettlemoyer, L. 2016. Human-in-the-loop parsing. In *EMNLP*, 2337–2342.

Iyer, S.; Konstas, I.; Cheung, A.; Krishnamurthy, J.; and Zettlemoyer, L. 2017. Learning a neural semantic parser from user feedback. In *ACL*, volume 1, 963–973.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS*.

Li, F., and Jagadish, H. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*.

Li, J.; Miller, A. H.; Chopra, S.; Ranzato, M.; and Weston, J. 2016a. Dialogue learning with human-in-the-loop. *arXiv preprint arXiv:1611.09823*.

Li, X.; Lipton, Z. C.; Dhingra, B.; Li, L.; Gao, J.; and Chen, Y.-N. 2016b. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*.

Li, H.; Min, M. R.; Ge, Y.; and Kadav, A. 2017. A context-aware attention network for interactive question answering. In *SIGKDD*, 927–935. ACM.

Liu, C.; Chen, X.; Shin, E. C.; Chen, M.; and Song, D. 2016. Latent attention for if-then program synthesis. In *NIPS*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *NIPS*, 1043–1049.

Pavlick, E.; Rastogi, P.; Ganitkevitch, J.; Van Durme, B.; and Callison-Burch, C. 2015. Ppdb 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification. In *ACL-IJCNLP*, 425–430.

Peng, B.; Li, X.; Li, L.; Gao, J.; Celikyilmaz, A.; Lee, S.; and Wong, K.-F. 2017. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *EMNLP*, 2221–2230.

Peng, B.; Li, X.; Gao, J.; Liu, J.; and Wong, K.-F. 2018. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. In *ACL*, volume 1, 2182–2192.

Quirk, C.; Mooney, R. J.; and Galley, M. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In *ACL*, 878–888.

Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract syntax networks for code generation and semantic parsing. In *ACL*, volume 1, 1139–1149.

Su, Y.; Awadallah, A. H.; Khabsa, M.; Pantel, P.; Gamon, M.; and Encarnacion, M. 2017. Building natural language interfaces to web apis. In *CIKM*, 177–186. ACM.

Su, Y.; Awadallah, A. H.; Wang, M.; and White, R. W. 2018. Natural language interfaces with fine-grained user interaction: A case study on web apis. In *SIGIR*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, 6.

Thomason, J.; Zhang, S.; Mooney, R. J.; and Stone, P. 2015. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, 1923–1929.

Ur, B.; Pak Yong Ho, M.; Brawner, S.; Lee, J.; Mennicken, S.; Picard, N.; Schulze, D.; and Littman, M. L. 2016. Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In *CHI*, 3227–3231. ACM.

Wang, X.; Chen, W.; Wu, J.; Wang, Y.-F.; and Wang, W. Y. 2017. Video captioning via hierarchical reinforcement learning. *arXiv preprint arXiv:1711.11135*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer. 5–32.

Yih, S. W.-t.; Chang, M.-W.; He, X.; and Gao, J. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base.

Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. *arXiv:1704.01696*.

Zhang, J.; Zhao, T.; and Yu, Z. 2018. Multimodal hierarchical reinforcement learning policy for task-oriented visual dialog. *arXiv preprint arXiv:1805.03257*.

Zhong, V.; Xiong, C.; and Socher, R. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR* abs/1709.00103.

Zhou, P.; Shi, W.; Tian, J.; Qi, Z.; Li, B.; Hao, H.; and Xu, B. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *ACL*, 207–212.

A User Answer Understanding Module

We define the user answer understanding module in Figure 2 based on a bidirectional Recurrent Neural Network with attention. Specifically, for the j-th word in the user answer d_i, we concatenate its forward and backward hidden states (i.e., $h_{d_{i,j}} = [\overrightarrow{h}_{d_{i,j}}; \overleftarrow{h}_{d_{i,j}}]$) as its semantic representation. Attention weights w_{att_j} over all words are computed with a trainable context vector c, i.e., $w_{att_j} = \text{softmax}(c^T h_{d_{i,j}})$, which will help the agent identify important words in d_i. User answer d_i is then represented as $v_{d_i} = \sum_j w_{\text{att}_j} h_{d_{i,j}}$.

B Training Algorithm for HRL

In our work, policy functions are all updated via Stochastic Gradient Ascent. Given the objective Eq (4), we deduce the updates for the high-level policy function below:

$$\nabla_\theta J(\theta) = E_{\pi^h}[\nabla_\theta \log \pi^h(g; s) u_t], \tag{6}$$

where $u_t = \sum_{\kappa=0}^{\infty} \gamma^\kappa r^h(s_{t+\sum_{n=1}^{\kappa} N_n}, g_{t+\sum_{n=1}^{\kappa} N_n})$ is the summation item within the two brackets in Eq (4).

Similarly, at low level, for the ongoing subtask $g_t \in \mathcal{G} = \{st_1, st_2, st_3, st_4\}$ at time step t, the updates of its policy function $\pi^l_{g_t}$ are given by:

$$\nabla_{\phi_{g_t}} J_{g_t}(\phi_{g_t}) = E_{\pi^l_{g_t}}[\nabla_{\phi_{g_t}} \log \pi^l_{g_t}(a; s) u_{g_t,t}], \tag{7}$$

where $u_{g_t,t} = \sum_{k \geq 0} \gamma^k r^l_{g_{t+k}}(s_{t+k}, a_{t+k})$.

Algorithm 1 details the entire training procedure. Line 1-2 initialize each policy network. Particularly, each low-level policy network is pre-trained via supervised learning (Section 4). We train the agent on M episodes (i.e., recipes). At the beginning of each episode, the state vector $s^l_{st_i}$ is initialized by calculating Eq (1-2) with $s^l_{st_{-i}} = \overrightarrow{0}$ (Line 7). Gradients of parameters θ and ϕ_{st_i} are accumulated during every episode. To improve efficiency, we perform gradient update for every 64 episodes (Line 31-34).

C Implementation Details

To accelerate the training of HRL-based agents, we adopt a similar skill as (Gordon et al. 2017; Wang et al. 2017) to initialize their low-level policy functions with the well-trained LAM-sup model. Particularly for HRL, we apply an "action mask" at the high level, so that the completed subtask will not be repeatedly chosen as the next subtask. However, the agent does not rely on this mask after being fully trained. The validation check is performed every 2K iterations on a randomly sampled 1K validation recipes. The best parameter setting is the one that obtains the most high-level reward in validation.

D User Simulator

For a trigger/action function, we design the following strategies to simulate user answers:

1. Revised function description: Each function has an official description on IFTTT.com, e.g., for function `Create a link note`, its description is "*This action will create a new link note from a source URL in the notebook you specify*." We revised the function description to make a simulated user answer, e.g., "*Create a new link note from a source URL in the notebook I specify*."
2. Paraphrases of the revised function description: Specifically, we replaced words and phrases with their paraphrases, which were collected and marked as semantically "Equivalence" by the PPDB paraphrase database (Pavlick et al. 2015).
3. Extractions from a large collection of real-user recipe descriptions in our training set (Ur et al. 2016): The extractions are based on six manually defined templates and will embrace language variation in simulated user answers. For example, for a recipe description "If X then Y," X will be considered as an answer to questions about the ground-truth trigger function and Y as an answer to those about the ground-truth action function.

E User Evaluation Examples

We present three user evaluation examples in Table 4.

Example 1
User: "pinterest post to wordpress" **Ground-truth recipe**: [tc: `Feed`, tf: `New feed item`, ac: `WordPress`, af: `Create a photo post`] **HRL agent**: "Which event triggers the action?" **User**: "when there's a new pinterest post' **HRL agent**: "Which event results from the trigger?" **User**: "create a post with photo" **Agent Prediction**: [tc: `Feed`, tf: `New feed item`, ac: `WordPress`, af: `Create a photo post`]
Example 2
User: "foto do instagram" **Ground-truth recipe**: [tc: `Instagram`, tf: `Any new photo by you`, ac: `Email`, af: `Send me an email`] **HRL agent**: "Which event triggers the action?" **User**: "I share new ins photo" **HRL agent**: "Which channel should act per your request?" **User**: "email" **Agent Prediction**: [tc: `Instagram`, tf: `Any new photo by you`, ac: `Email`, af: `Send me an email`]
Example 3
User: "send fiverr orders to wunderlist" **Ground-truth recipe**: [tc: `Fiverr`, tf: `New order received`, ac: `Gmail`, af: `Send an email`] **HRL agent**: "Which event triggers the action?" **User**: "a new order is recievied" **Agent Prediction**: [tc: `Fiverr`, tf: `New email in inbox from search`, ac: `Gmail`, af: `Send an email`]

Table 4: Two examples (Example 1-2) from user evaluation where the HRL agent correctly interpreted the user instructions and two (Example 3) where it did not. The agent failed when there is a typo in the word "recievied".

Algorithm 1 Learning algorithm for HRL

Initialize parameters θ of the high-level policy network randomly.
Initialize parameters ϕ_{st_i} $(i = 1, 2, 3, 4)$ of each low-level policy network with supervised pre-training.
Initialize gradients: $d\theta \leftarrow 0$, $d\phi_{st_i} \leftarrow 0$.
for $\#episode = 1$ to M **do**
    Reset the user simulator and get a recipe description $\mathcal{I}$.
    Initialize $b_i \leftarrow 0, d_i \leftarrow \emptyset, \forall i = 1, 2, 3, 4$. Observe $s_0 = \{\mathcal{I}, b_i, d_i\}$.
    Calculate $s^l_{st_i}, \forall i = 1, 2, 3, 4$, according to Eq (1-2), with $s^l_{st_{-i}} = \overrightarrow{0}$.
    $global\_turn \leftarrow 1$.
    $t \leftarrow 0$.
    while s_t is not terminal **and** $global\_turn \leq Max\_Global\_Turn$ **do**
        Sample a subtask $g_t \sim \pi^h(g; s_t)$ according to Eq (3). We denote g_t as st_{i_t}, i.e., the i_t-th subtask in the subtask set $\mathcal{G} = \{st_1, st_2, st_3, st_4\}$.
        $t_{start} \leftarrow t$.
        $local\_turn \leftarrow 1$, $a_t \leftarrow \varnothing$.
        while ($a_t == \varnothing$ **or** $a_t ==$ "AskUser") **and** $local\_turn \leq Max\_Local\_Turn$ **do**
            Sample a primitive action $a_t \sim \pi^l_{st_{i_t}}(a; s_t)$.
            Execute and receive a low-level reward $r^l_{st_{i_t}}(s_t, a_t)$.
            $d_{i_t} \leftarrow d_{i_t} \cup$ retrieved simulated user answer.
            Observe new state s_{t+1} with new d_{i_t}.
            Update $s^l_{st_{i_t}}$ according to Eq (1-2).
            $g_{t+1} \leftarrow g_t$.
            $t \leftarrow t + 1$.
            $local\_turn \leftarrow local\_turn + 1$.
        end while
        $d\phi_{st_{i_t}} \leftarrow d\phi_{st_{i_t}} +$ accumulated gradient according to Eq (7).
        Receive a high-level reward $r^h(s_{t_{start}}, g_{t_{start}})$.
        $b_{i_t} \leftarrow 1$.
        $t \leftarrow t + 1$. Observe state s_t.
        $global\_turn \leftarrow global\_turn + 1$.
    end while
    $d\theta \leftarrow d\theta +$ accumulated gradient according to Eq (6).
    if $\#episode \ \% \ 64 \ == 0$ **then**
        Perform update of θ and ϕ_{st_i} $(i = 1, 2, 3, 4)$ by gradient ascent using $d\theta$ and $d\phi_{st_i}$.
        $d\theta \leftarrow 0$, $d\phi_{st_i} \leftarrow 0$.
    end if
end for