The Thesis Committee for Shobhit Chaurasia

certifies that this is the approved version of the following thesis:

Dialog for Natural Language to Code

Approved by
Supervising Committee:

Raymond J. Mooney, Supervisor

Milos Gligoric

Dialog for Natural Language to Code

by

Shobhit Chaurasia, B.Tech

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

May 2017

Acknowledgments

First and foremost, I would like to thank my adviser, Ray Mooney, for his constant intellectual support throughout my thesis research. I would also like to thank Milos Gligoric, who served on my advising committee, for his support, especially for bringing in Software Engineering ideas to my research. Discussions with both of them got me interested in the exciting intersection of the fields of Natural Language Processing and Software Engineering.

I am grateful to my lab-mate Aishwarya Padmakumar for her tireless efforts since the conception of this thesis. Some of my research would not have been possible without her ideas and insights.

I am indebted to my family for creating an environment conducive of learning since my childhood, something which shaped my interest in academia and motivated me to pursue graduate studies, and for their emotional support throughout my time away from home.

Special thanks to my utilitarian brother, Gaurav Chaurasia, with whom I have the most useful relationship I could imagine. Without his invaluable help, I would not have gotten a chance to explore my passion for research in the first place.

Finally, I would like to thank my partner, Mahika Bhasin, and my sister, Ankita Chaurasia, without whom, admittedly, I could have done more work in my research, but it would not have been fun.

This work was supported by a gift from Microsoft Research to Ray Mooney.

Shobhit Chaurasia

The University of Texas at Austin

May 2017

Dialog for Natural Language to Code

Shobhit Chaurasia, MSCompSci
The University of Texas at Austin, 2017

Supervisor: Raymond J. Mooney

Generating computer code from natural language descriptions has been a long-standing problem in computational linguistics. Prior work in this domain has restricted itself to generating code in one shot from a single description. To overcome this limitation, we propose a system that can engage users in a dialog to clarify their intent until it is confident that it has all the information to produce correct and complete code. Further, we demonstrate how the dialog conversations can be leveraged for continuous improvement of the dialog system. To evaluate the efficacy of dialog in code generation, we focus on synthesizing conditional statements in the form of IFTTT recipes. IFTTT (if-this-then-that) is a web-service that provides event-driven automation, enabling control of smart devices and web-applications based on user-defined events.

Contents

Acknowledgments **iv**

Abstract **vi**

List of Tables **x**

List of Figures **xii**

Chapter 1 Introduction and Related Work **1**

1.1 Chapter Overview 1

1.2 Code Generation 1

1.3 Dialog Systems 4

1.4 Thesis Outline 5

Chapter 2 Task Overview **7**

2.1 Chapter Overview 7

2.2 IFTTT Domain 7

2.2.1 Channels and Functions 8

2.2.2 Fields and Ingredients 9

2.2.3 IFTTT Dataset 11

2.3 Problem Statement 12

Chapter 3 Dialog System **14**

3.1 Chapter Overview 14

3.2 Dialog Manager 16

3.2.1 Belief State 17

3.2.2 Static Dialog Policy 17

3.3 Natural Language Understanding 21

3.3.1 Neural Model 24

3.3.2 Input Preprocessing 27

3.3.3 Training Details 27

3.4 Natural Language Generation 28

3.5 Fields in the Dialog Setting 28

3.6 Retraining NLU using Dialog 30

Chapter 4 Experiments **33**

4.1 Chapter Overview 33

4.2 Experimental Setup 34

4.3 Dialog Experiments 35

4.3.1 Constrained User-Initiative 35

4.3.2 Free User-Initiative 35

4.3.3 Results 37

4.4 Parser Retraining 39

4.4.1 Results 40

Chapter 5 Conclusion and Future Work **41**

Appendix A **43**

Bibliography **45**

Vita **54**

List of Tables

2.1 Number of recipes in train, validation, and test sets of the original IFTTT dataset and the one we were able to collect for this work. 11

3.1 Example system utterances for confirmation requests leveraging API documentation. 29

3.2 Example system utterances for requesting information about specific slots. For functions, the utterance uses context from the belief state in the form of value assigned to the associated channel. 29

3.3 Training data for the four models extracted from the conversation in Figure 3.1. 32

4.1 Accuracy of recipe synthesis (Channel + Function) measured on the gold subset. 37

4.2 Accuracy of recipe synthesis (Channel + Function) measured on the gold subset before and after retraining the semantic parser. 37

4.3 Dialog length statistics measured in terms of number of user utterances. 38

List of Figures

2.1 A sample recipe represented as an Abstract Syntax Tree. . . . 10

3.1 Sample dialog between the system and a user over a recipe with trigger channel `flickr`, trigger function `new_public_photo_tagged`, action channel `email`, and action function `send_me_an_email`. 15

3.2 Average dialog length for different values of α and β against a simulated user on IFTTT validation set. Average dialog length is measured in terms of number of simulated user's utterances. 22

3.3 Rate of dialog success for different values of α and β against a simulated user on IFTTT validation set. 23

3.4 Network architecture. 24

4.1 The dialog interface showing a sample conversation over a recipe with trigger `any_event_starts` on `google_calendar` and action `send_a_notification` on `google_glass`. The recipe description along with details of trigger and action were presented to the user alongside the dialog interface. 36

Chapter 1

Introduction and Related Work

1.1 Chapter Overview

This chapter provides an introduction to the work presented in this thesis. It describes the two main relevant areas, code generation through language and dialog systems, and lists some of the related work in these areas. Finally, it gives a brief overview of the goal of this work and describes where it fits in the intersection of these two areas and how it relates to and departs from the previous work.

1.2 Code Generation

Building natural language interface to programmatic tasks has long been a goal of computational linguistics. Such an interface could break the barrier for novice users to write simple programs to automate repetitive tasks without requiring an exposure to programming languages. For example, it can aid them in programming their digital devices, connect web-services, and ease spreadsheet calculations. It can also speed up programming by professional programmers by taking the burden off them with respect to syntactic details, allowing them to focus on semantic and algorithmic details.

This has been explored in a plethora of Domain Specific Languages (DSLs) (Zelle and Mooney, 1996; Berant et al., 2013; Yin et al., 2016; Yaghmazadeh et al., 2017; Manshadi et al., 2013; Kate et al., 2005; She et al., 2014; Gulwani and Marron, 2014; Quirk et al., 2015; Dong and Lapata, 2016; Beltagy and Quirk, 2016; Liu et al., 2016) such as SQL and regular expressions, as well as general-purpose programming languages (Lei et al., 2013; Ling et al., 2016; Yin and Neubig, 2017) such as Python and C++.

In the realm of general-purpose programming languages, Lei et al. (2013) proposed a method to generate code to parse inputs to a program that have a fixed pattern or structure. The pattern is described in natural language, such as "The first line of the input consists of two integers, n and m. n lines follow, each containing a string." Ling et al. (2016) introduced a neural architecture to generate code for Trading Card Games. Their network takes as input a mix of natural language description of a card's effect when it is played – such as "Draw cards until you have as many in hand as your opponent" – and structured specification for that card – such as its name and the cost of playing that card – and produces a computer code to emulate the card. Yin and Neubig (2017) extended this by adding a probabilistic grammar model, moving from Ling et al. (2016)'s approach of treating code generation as a free-form language generation task to one informed by the prior knowledge of the syntax of the target language.

Efforts directed towards DSLs include generating database queries from natural language specification (Zelle and Mooney, 1996; Berant et al., 2013; Yin et al., 2016; Yaghmazadeh et al., 2017), building regular expressions (Manshadi et al., 2013), commanding a robot (Kate et al., 2005; She et al., 2014), programming on spreadsheets (Gulwani and Marron, 2014). Another domain that has seen recent interest is that of event-driven automation through IFTTT[1] (if-this-then-that) (Quirk et al., 2015; Dong and Lapata, 2016; Beltagy and Quirk, 2016; Liu et al., 2016). On IFTTT, users can create scripts to automate tasks connecting web-services and smart devices. These scripts run when a user-defined event occurs (for example, when someone is tagged in a photograph on Facebook), and they perform a user-specified action (such as saving the photograph on Dropbox).

Code synthesis through other mediums such as examples of program inputs along with desired outputs has also been explored (YWM, 2001; Gulwani, 2012; Raza et al., 2014). So and Oh (2017) integrated program synthesis with static analysis to complete partially written imperative programs through input-output examples. Wu and Knoblock (2015) investigated an iterative approach to program synthesis through examples. Departing from the traditional approach of generating programs from scratch through a set of examples, they

[1]www.ifttt.com

refined previously generated subprograms with additional inputs provided by the user in an iterative fashion. Manshadi et al. (2013) augmented natural language descriptions with examples to aid generation of regular expressions. Raza et al. (2015) proposed a domain-agnostic program synthesis framework that synthesizes programs based on compositional inputs in the form of descriptions and examples.

The existing work in all these domains assumes that a working program can be generated in one shot from a single natural language description, possibly augmented with auxiliary inputs like structured specifications and examples. However, in many cases, users omit important details that prevents the generation of fully executable code from their initial description.

1.3 Dialog Systems

Another line of research that has recently garnered increasing attention is that of dialog systems (Singh et al., 2002; Young et al., 2013). Dialog systems have been employed for goal-directed tasks such as providing technical support (Lowe et al., 2015) and travel information and booking (Williams et al., 2013), as well as in non-goal oriented domains such as social-media chat-bots (Ritter et al., 2011; Shang et al., 2015). The rapid progress in dialog systems has been mirrored in a surge in interest in conversational agents and personal assistants such as Microsoft's Cortana, Apple's Siri, and Amazon's Alexa. Kenny et al. (2008) demonstrated the use of virtual assistants in health-care, Harvey et al. (2015) evaluated their utility in aiding education, and Gordon and Breazeal

(2015) designed a conversational interface to in-car entertainment systems. Closer to the domain of web-services — which is the central theme of this work — Azaria et al. (2016) proposed a system that can learn to perform basic action sequences, such as sending an email to a contact, by asking the user to demonstrate how to execute the action through a sequence of steps using natural language.

1.4 Thesis Outline

In this work, we combine these two lines of research and propose a system that engages the user in a dialog, asking questions to elicit additional information until the system is confident that it fully understands the user's intent and has all of the details to produce correct and complete code. An added advantage of the dialog setting is the possibility of continuous improvement of the underlying semantic parser through conversations (Artzi and Zettlemoyer, 2013; Thomason et al., 2015; Azaria et al., 2016; Weston, 2016), which could further increase success rates for code generation and result in shorter dialogs. We focus on a restrictive, yet important class of programs that deal with conditions, i.e., `if-then` statements. To this end, we use the IFTTT dataset released by Quirk et al. (2015). To the best of our knowledge, this is the first attempt to use dialog for code generation from language.

To evaluate the efficacy of dialog in code generation, we conducted experiments in which users engaged in a dialog with the system to clarify their intent and answer its questions. Our dialog approach achieved $10 - 15$

percentage points higher accuracy on code generation as compared to the state-of-the-art single-shot approach. We also demonstrate how data extracted from conversations can be used for improving the system. In our experiments, we were able to achieve a marginal improvement of 1.1 percentage points in success rates for code generation, although with slightly longer dialogs.

The rest of the thesis is structures as follows. Chapter 2 provides the background and sets the context for the work presented in this thesis. It goes over the IFTTT domain in detail and formally presents the problem statement. Chapter 3 explains the proposed dialog system in detail. Experiments conducted to evaluate the efficacy of dialog in code generation and the possibility of continuous improvement through conversations are described in Chapter 4. It also presents and analyzes our observations. Finally, Chapter 5 concludes this thesis and lists some ideas for future work.

Chapter 2

Task Overview

2.1 Chapter Overview

This chapter sets the context for the work presented in this thesis. It begins with an overview of the IFTTT domain that is targeted in this work. The IFTTT domain is used as a proxy for conditional statements in computer code. This is followed by a description of the IFTTT dataset released by Quirk et al. (2015). Finally, it presents the problem statement formally and motivates why the proposed system — a dialog setting for synthesizing computer code in the IFTTT domain — is a reasonable approach to solving that problem.

2.2 IFTTT Domain

IFTTT is a web-service that allows users to automate simple tasks by creating short scripts, called *recipes*, through a GUI. They enable users to connect web-services (such as Facebook, Dropbox, Weather Underground) and smart devices (such as Phillips Hue lights, Wemo motion sensors). The type of tasks that can be automated through IFTTT recipes is varied, ranging from health monitoring ("Text me my daily exercise summary.") to social-media management ("Post my Facebook photos on Twitter.") to home automation ("Blink my lights when Texas Longhorns score.") and much more.

2.2.1 Channels and Functions

A recipe consists of a *trigger* — an event which fires the recipe — and an *action* — the task to be performed when the recipe is fired. Among other things, a trigger is characterized by a *trigger channel*, which is the source of the event, and a *trigger function*, which is the nature of the event; an action is characterized by an *action channel*, which is the destination of the task to be performed, and an *action function*, which is the nature of that task.

Users can share their recipes publicly with short descriptions of their functionalities. Others can use the published recipes based on their descriptions. For example, consider a recipe with the following description:

"Text me when I am tagged in a picture on Facebook."

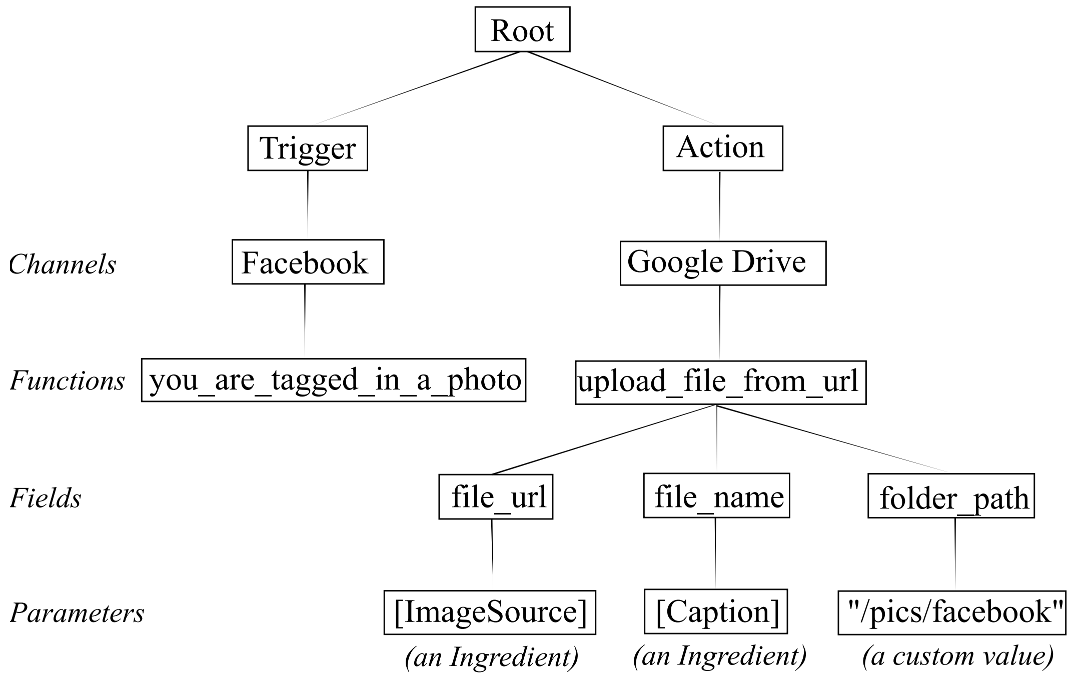
This recipe might have `you_are_tagged_in_a_photo` as the trigger function associated with the `facebook` trigger channel and `send_me_an_sms` as the action function associated with the `sms` action channel.

2.2.2 Fields and Ingredients

In addition to channels and functions, recipes might also have *fields*, which are arguments to functions. Most of the functions have fields that can take custom values. For example, the action function `send_me_an_sms` in the recipe above has a field `message` which specifies the content of the text message to be sent when the recipe fires. These values can either be fully specified by the users or contain parameters called *ingredients*, which are properties of the event that fired the recipe that can be utilized by the action function. For example, the trigger function `you_are_tagged_in_a_photo` exposes an ingredient `photo_url` which captures the URL of the photo that caused the recipe to fire, and which can be used as part of the `message` field.

A small subset of functions have fields which can take values only from a predefined list. For example, `tomorrow's_forecast_calls_for` trigger function — for monitoring changes in tomorrow's forecasted weather conditions — from `weather` channel has a field `condition` which can only take values from a fixed set containing `rain`, `snow`, and `sunny`, among others.

Figure 2.1 illustrates how channels, functions, arguments, and parameters interact with each other and how a recipe can be represented as an Abstract Syntax Tree (Quirk et al., 2015). For reasons discussed in Section 2.3, the experiments on recipe synthesis in this work focus only on channels and functions.



"Download Facebook images I am tagged in to Dropbox"

Figure 2.1: A sample recipe represented as an Abstract Syntax Tree.

Train		**Validation**		**Test**	
Original	Ours	Original	Ours	Original	Ours
77,495	66,588	5,171	3,992	4,294	3,685

Table 2.1: Number of recipes in train, validation, and test sets of the original IFTTT dataset and the one we were able to collect for this work.

2.2.3 IFTTT Dataset

Quirk et al. (2015) released the IFTTT dataset containing over 114k recipes. The released dataset contains recipe URLS; recipe details and their descriptions can be obtained by crawling the IFTTT website. Since the dataset was released, many recipes have been taken down by their authors. Statistics of the dataset are summarized in Table. 2.1.

In addition to recipe details, the test set also contains labels for channels and functions assigned by humans when presented with the recipe descriptions. A small subset of the test set, namely "gold" subset, was identified by Quirk et al. (2015) on which at least three humans agreed with the true labels. The gold subset consisted of 758 recipes, out of which only 550 are now available. This subset consists of recipes which most of the humans could reconstruct from their descriptions, and an automated system built to accomplish the same goal should be able to achieve high rate of success on synthesizing recipes at least in this subset. All prior work using the IFTTT dataset (Quirk et al., 2015; Dong and Lapata, 2016; Beltagy and Quirk, 2016; Liu et al., 2016) have evaluated their systems at least on the gold subset.

2.3 Problem Statement

Our goal is to synthesize IFTTT recipes from their natural language descriptions. However, unlike previous work in this domain (Quirk et al., 2015; Dong and Lapata, 2016; Beltagy and Quirk, 2016; Liu et al., 2016), which restrict the system to synthesizing a recipe from a single description, we seek to enable the system to interact with users by engaging them in a dialog to clarify their intent when the system's confidence in its inference is low. This is particularly crucial when there are multiple channels or functions achieving similar goals — the action in the example recipe above could also be performed by `send_an_sms` action function of `android_messages` channel — or when recipe descriptions are vague — the example recipe above could be summarized by a user as "Let me know when I am tagged in a picture on Facebook," in which case, it is unclear if the user wants to receive a text message, an email, or a notification.

Automating recipe synthesis through their descriptions equates to invoking methods and procedures based on conditions, an indispensable part of computer programming, through natural language commands. The level of abstraction that recipes provide lends this task to a convenient dialog setting: The procedures to be invoked — "functions" in IFTTT parlance — are described in terms of their high-level functionality, along with arguments necessary for their invocation, without the need to prompt users, who are likely to be non-programmers, about execution details, something that could be overwhelming and almost akin to writing code.

By switching from a single-shot setting to a dialog setting allows the system to elicit additional information to produce complete, correct code, thereby potentially increasing success rates for recipe synthesis. An added advantage of the dialog setting is the possibility of continuous improvement of the underlying semantic parser through conversations (Artzi and Zettlemoyer, 2013; Thomason et al., 2015; Azaria et al., 2016; Weston, 2016). Improvements in semantic parser could boost the single-shot accuracy, further increase success rates for code generation in the dialog setting, and result in shorter dialogs.

The dialog system we propose restricts itself to inferring channels and functions. This is partly because, as argued by Beltagy and Quirk (2016) and Liu et al. (2016), fields are not central to synthesizing a recipe, and more often than not, they take custom values specified by users (such as path of a `dropbox` folder where images should be saved, or an email id from which received emails should be posted on `facebook`) which cannot be learned. The main reason, however, is that field information is now available only for a handful of recipes from the IFTTT dataset. Since the work by Quirk et al. (2015) and Liu et al. (2016), it seems, IFTTT has made changes to its website because of which field information can only be collected for a tiny subset of recipes in an automated way. However, as argued in Section 3.5, the inclusion of fields, either though explicit specification by the user or through automatic inference, is straightforward in the proposed dialog system.

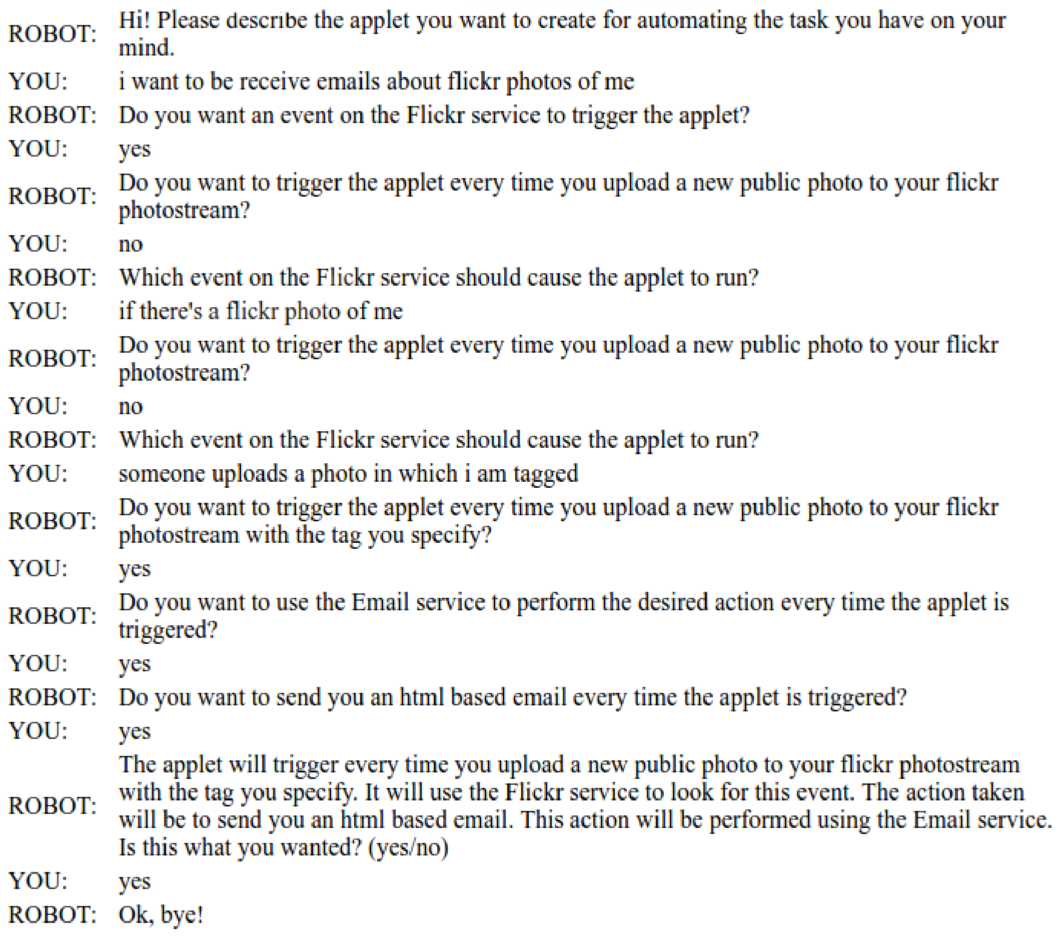
Chapter 3

Dialog System

3.1 Chapter Overview

We propose a text-based dialog system with which users can converse using natural language to create recipes. A sample conversation between the system and a user is shown in Figure 3.1. The dialog system consists of three components, namely Dialog Manager, Natural Language Understanding (NLU), and Natural Language Generation (Jurafsky, 2000).

This chapter describes the proposed dialog system in detail, going over its main three components, with descriptions of the state- and action-space of the dialog agent and the policy it follows. Further, this chapter argues how the inclusion of fields, which are not taken into account in this work, in the dialog setting is straightforward. Finally, it explains how we use the conversations to continuously improve the dialog system.



ROBOT: Hi! Please describe the applet you want to create for automating the task you have on your mind.
YOU: i want to be receive emails about flickr photos of me
ROBOT: Do you want an event on the Flickr service to trigger the applet?
YOU: yes
ROBOT: Do you want to trigger the applet every time you upload a new public photo to your flickr photostream?
YOU: no
ROBOT: Which event on the Flickr service should cause the applet to run?
YOU: if there's a flickr photo of me
ROBOT: Do you want to trigger the applet every time you upload a new public photo to your flickr photostream?
YOU: no
ROBOT: Which event on the Flickr service should cause the applet to run?
YOU: someone uploads a photo in which i am tagged
ROBOT: Do you want to trigger the applet every time you upload a new public photo to your flickr photostream with the tag you specify?
YOU: yes
ROBOT: Do you want to use the Email service to perform the desired action every time the applet is triggered?
YOU: yes
ROBOT: Do you want to send you an html based email every time the applet is triggered?
YOU: yes
ROBOT: The applet will trigger every time you upload a new public photo to your flickr photostream with the tag you specify. It will use the Flickr service to look for this event. The action taken will be to send you an html based email. This action will be performed using the Email service. Is this what you wanted? (yes/no)
YOU: yes
ROBOT: Ok, bye!

Figure 3.1: Sample dialog between the system and a user over a recipe with trigger channel `flickr`, trigger function `new_public_photo_tagged`, action channel `email`, and action function `send_me_an_email`.

3.2 Dialog Manager

The dialog manager is responsible for tracking the *state* of the system and determining its *actions* based on its state and the environment with which the system is interacting. The state consists of all pieces of information that is relevant to the dialog system. The actions could modify the environment, which can in turn alter the state of the system.

In this work, the aim of the dialog system is to determine values of channels and functions for the recipe that the user wants to create. We cast this problem as a slot-filling task: Each piece of information needed to synthesize a recipe is a slot that the system seeks to fill through its interaction with the user. This is related to the idea of Sketching in program synthesis (Solar-Lezama, 2008), where a partial program is sketched at a high-level, leaving *holes* for low-level details. The roles, however, are reversed: Slots, unlike holes, capture high-level functionality through methods, procedures, or conditions (triggers and actions in IFTTT) to be filled by the user, leaving the their low-level implementation to a programmer.

The system engages in a dialog with the user until all slots are filled. It maintains a belief state — its current estimates for all the slots — and follows a hand-coded policy — a strategy of how to drive the conversation — to update its belief state until it is confident that the belief state is same as the user goal. The strategy is similar to the one used in Thomason et al. (2015), in which there are three slots, `action`, `patient`, and `recipient`, for their dialog system to fill for navigation- and delivery-related tasks.

3.2.1 Belief State

The belief state consists of four slots: `trigger_channel`, `action_channel`, `trigger_function`, and `action_function`. As evident from Figure 2.1, slots in the IFTTT domain naturally form a hierarchy: channels are above functions and functions are above fields. Although triggers and actions are, in a loose sense, at the same level in the hierarchy[1], it is more natural to specify triggers before actions, thereby inducing a complete hierarchy over slots. This hierarchy is exploited in specifying a policy for the dialog system (see Section 3.2.2).

The system maintains a probability distribution over all possible values for each slot. After each user utterance, the probability distribution for one or more slots is updated based on the parse of the user utterance returned by the utterance parser (see Section 3.3). The system follows a hand-coded dialog policy over the discrete state-space obtained from the belief state by assigning the values with highest probability — i.e., candidates with highest confidence — to each slot.

3.2.2 Static Dialog Policy

At each step in the conversation, the system needs to decide which slot to consider next and which action to take for that slot. Actions in our system include an opening greeting, asking the user to describe the recipe; informa-

[1]Technically, the presence of ingredients — properties associated with trigger functions that can be utilized by action functions — puts triggers above actions in the hierarchy.

tion request for a slot; confirmation request for a slot; and request to reword previous user utterance.

The system follows a static, hand-coded policy. The dialog opens with an open-ended user utterance in which the user is expected to describe the recipe. Such a free-form user utterance is called user-initiative, because it is the user who is driving the conversation through that utterance. The parse of the user-initiative is used to update all of the slots in the belief state. The system moves down the slot-hierarchy, one slot at a time, and picks the next action based on the confidences of the top candidates for each slot. If the confidence in the top candidate for a slot is above α, the parse is deemed confident, and the candidate is assigned to that slot; the system then proceeds to the next slot in the hierarchy, if available. If the confidence is below β, the parse is rejected, and the system explicitly seeks information for that slot. Such a slot-specific utterance from the system is called system-initiative, because it is the dialog system which is driving the conversation through that utterance. If the confidence is between the α and β, the system seeks a confirmation of the candidate value for that slot; if the user affirms, the candidate is assigned to the slot, otherwise the system requests information for that slot.

In our system, only the top candidate for each slot is taken into consideration. Hence, confirmation is sought only for the top candidate for each slot, if at all. Alternatively, we could move down the list of top-k candidate values and seek confirmation for each until the user affirms. We chose $k = 1$ to avoid overwhelming the user with confirmation requests and encourage them to word their intentions in a more comprehensible manner so that the top candidates are more likely to be the correct ones.

Value of α and β present a trade-off between dialog success, dialog length, and overall user-engagement and user-satisfaction with the dialog system. α directly affects dialog length and dialog success. Higher values of α increase the rate of dialog success because it becomes increasingly unlikely for the system to be stuck in an impasse when it mistakenly deems an incorrect parse correct without confirmation. At the same time, higher values of α can lead to longer dialogs because the system might seek confirmation more often before assigning a candidate value to a slot. β, on the other hand, affects user-engagement and user-satisfaction during the conversation, while indirectly affecting dialog success and dialog length. Higher values of β could prevent the system from even seeking confirmation for a candidate value which might actually be correct but whose confidence isn't high enough, instead forcing the user to reword their utterance, potentially leading to longer, less engaging dialogs. Lower values of β could overwhelm the user with confirmation requests even for candidates with low confidence, not only leading to longer dialogs, but also increasing user's frustration with the system, which could result in premature dialog termination by the user.

Setting values of α and β

To set values of α and β, we analyzed the performance of the dialog system on the IFTTT validation set. Since conversations with humans are expensive and time-consuming, we created a simulated user that interacted with the dialog system with different values of α and β and measured dialog success rates and lengths of dialogs. The conversations were over the recipes in the IFTTT validation set.

In addition to recipe descriptions, the simulated user has access to the true labels for channels and functions associated with each recipe which are used to respond to system utterances. Its utterances are generated by combining template responses with the IFTTT API documentation for triggers and functions. For example, for a recipe with true action function `blink_lights` of the `hue` action channel, the simulated user's response to a system-initiative for the `action_function` slot would be: "briefly turn your hue lights off then back on," a phrase from the function's API documentation.

The simulated user is similar to humans in that both have the knowledge of channels and functions for the recipe over which they are conversing with the system: simulated user by virtue of being given access to the true labels, and humans by virtue of the goal of their interaction with the system, i.e. their own understanding of the recipe that they want the system to create. However, the simulated user is not a perfect simulation of real users since it always describes a particular channel or function using the same words, it cannot reword its utterances when the system fails to parse them — because they are derived from the API documentation which has a single description for each channel and function — and its utterances aren't as noisy as those of humans — humans often use telegraphic language and misspell words which could be harder for the system to parse.

Despite these limitations, the simulated user provides a straightforward way to grid-search over different values of α and β in an automated manner. Results of this grid-search are summarized in Figure 3.2 and 3.3. In line with our intuition, we observed that a high value of α with a low value of β boosts rate of dialog success but leads to longer dialogs, while the opposite is true for a low value of α with a high value of β. Based on these observations, the values of $\alpha = 0.85$ and $\beta = 0.25$ were chosen for all experiments in this work.

3.3 Natural Language Understanding

The Natural Language Understanding (NLU) component of the dialog system is responsible for parsing user utterances. The parse returned by this component is used by the Dialog Manager to update the system's belief state, which in turn determines its actions. We use the model proposed by Liu et al. (2016), an LSTM-based classifier enhanced with a hierarchical, two-level attention mechanism (Bahdanau et al., 2014). The model is described in detail in the following section. In our system, the semantic parser is a set of four such models, one for each slot. User initiatives are parsed by all four models, while user responses to system-initiatives are parsed by the model corresponding to the slot under consideration.

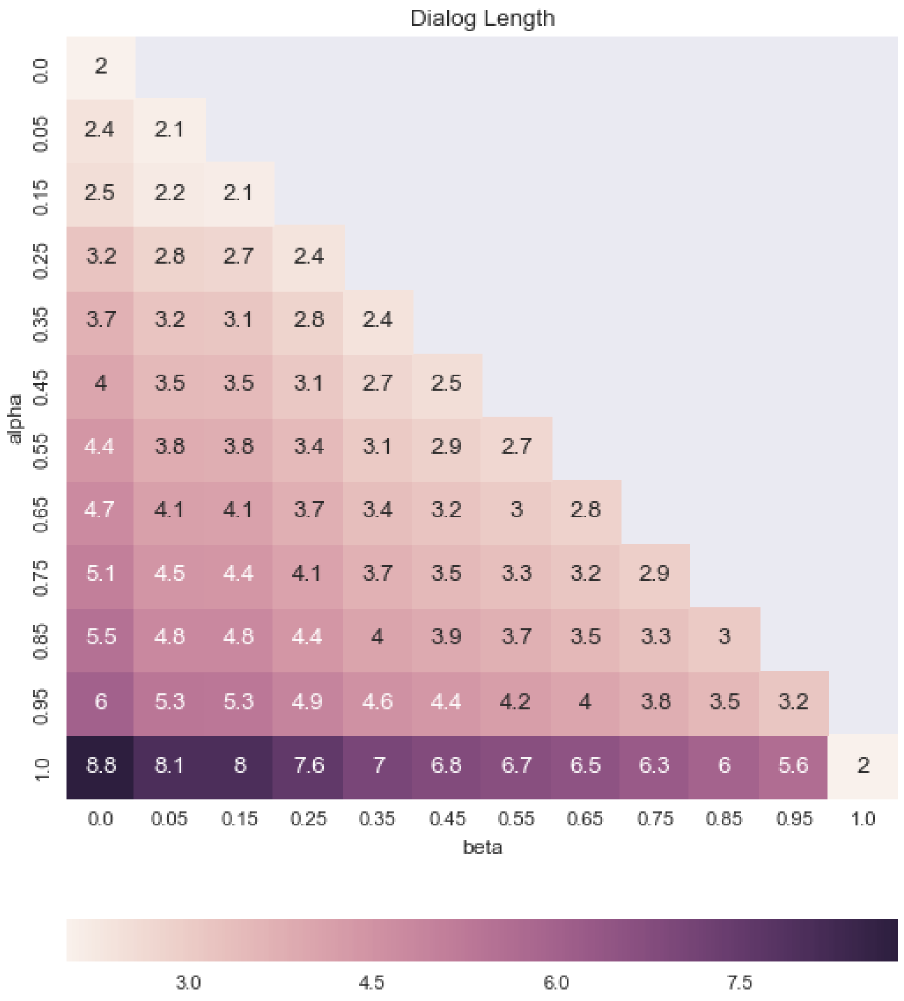


Figure 3.2: Average dialog length for different values of α and β against a simulated user on IFTTT validation set. Average dialog length is measured in terms of number of simulated user's utterances.

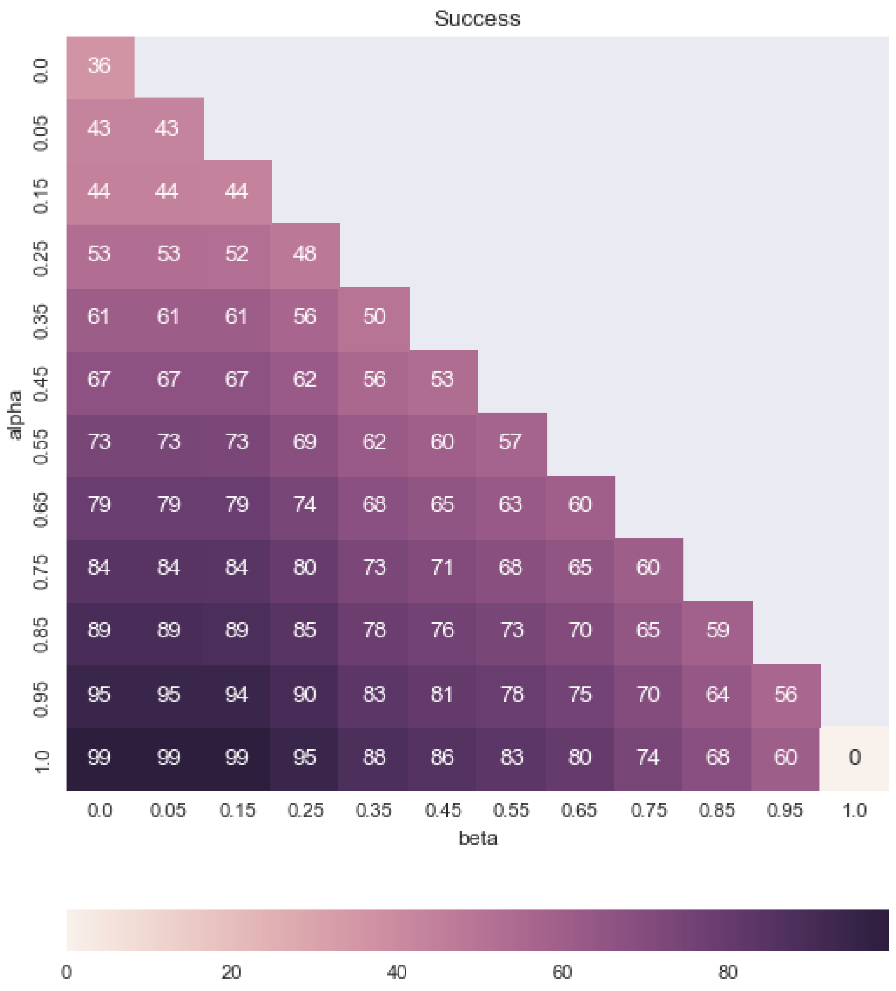


Figure 3.3: Rate of dialog success for different values of α and β against a simulated user on IFTTT validation set.

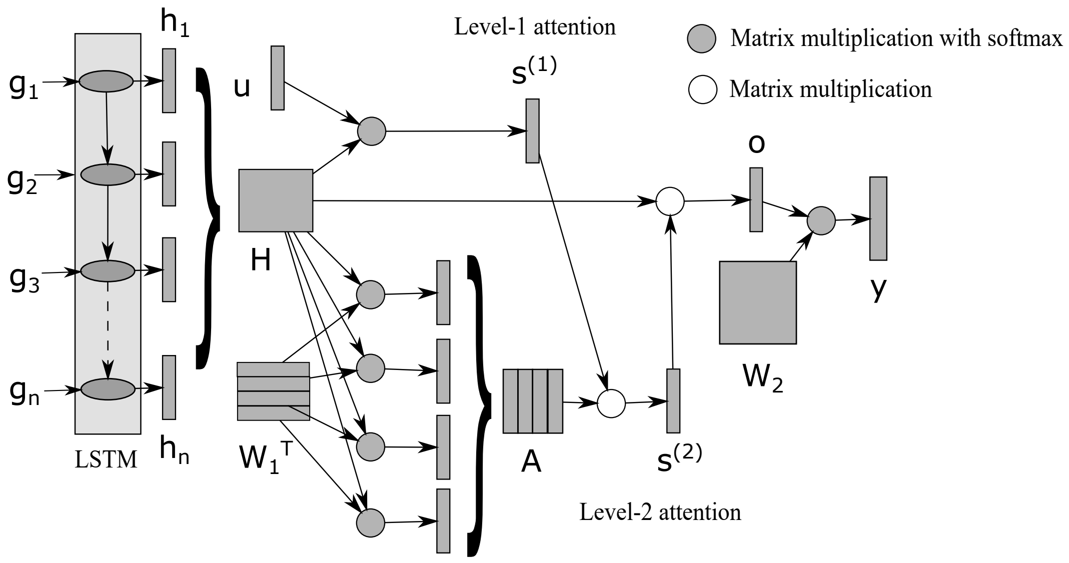


Figure 3.4: Network architecture.

3.3.1 Neural Model

The core of the NLU component uses the neural model proposed by Liu et al. (2016). It is a bidirectional LSTM model enhanced with two layers of attention. The architecture is presented in Figure 3.4. Recurrent Neural Networks (Zaremba et al., 2014), of which LSTM (Hochreiter and Schmidhuber, 1997) is an example, enhanced with attention mechanism (Bahdanau et al., 2014) has seen success in machine translation (Bahdanau et al., 2014), question answering (Sukhbaatar et al., 2015), and syntactic parsing (Vinyals et al., 2015), among others.

Input Representation The input to the model is a sequence of embeddings of tokens extracted from the recipe description. The tokens come from a vocabulary of size v. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be the set of tokens represented as one-hot encoded vectors. The input to the model is the set of d-dimensional embeddings $\mathbf{g}_1, \ldots, \mathbf{g}_n$ of these tokens, where

$$\mathbf{g}_i = \mathbf{G}\mathbf{x}_i \tag{3.1}$$

$\mathbf{G}$ is a $d \times v$ trainable matrix which projects the sparse one-hot vectors for tokens into a dense low-dimensional space.

The final representation of the input description is obtained by running the embeddings, $\mathbf{g_i}$, of the tokens through a bidirectional LSTM network.

$$\mathbf{h}_t = \mathcal{L}\left(\mathbf{g}_t\right) \tag{3.2}$$

The function $\mathcal{L}$ represents the output of the bidirectional LSTM (see Appendix A for details). Let $\mathbf{H}$ be a $d \times n$ matrix whose columns are $\mathbf{h}_1, \ldots, \mathbf{h}_n$.

Level-1 Attention The first-level attention scores are calculated as:

$$\mathbf{s}^{(1)} = \text{softmax}\left(\mathbf{H}^\intercal \mathbf{u}\right) \tag{3.3}$$

where $\mathbf{u}$ is a d-dimensional trainable vector.

Level-2 Attention The second-level attention scores are calculated as:

$$\mathbf{s}^{(2)} = \mathbf{A}\mathbf{s}^{(1)} \tag{3.4}$$

$$\text{where} \quad \mathbf{A} = \operatorname{softmax}(\mathbf{W}_1\mathbf{H}) \tag{3.5}$$

Here, $\mathbf{W}_1$ is an $n \times d$ trainable parameter matrix. The softmax operation in Equation 3.5 is performed column-wise.

Output Representation The input embeddings are weighted by the second-level attention scores to get the final representation, $\mathbf{o} \in \mathcal{R}^d$, of the input description.

$$\mathbf{o} = \mathbf{H}\left(\frac{\mathbf{s}^{(2)}}{||\mathbf{s}^{(2)}||}\right) \tag{3.6}$$

Prediction A softmax-layer is used to get the final predictions of the model.

$$\hat{\mathbf{y}} = \operatorname{softmax}(\mathbf{W}_2\mathbf{o}) \tag{3.7}$$

where $\mathbf{W}_2$ is a $c \times d$ trainable parameter matrix, c being the number of output classes, such as the number of trigger functions.

In this work, we use an ensemble of ten models for each slot, following the best performing system by Liu et al. (2016). The softmax predictions of the models (Equation 3.7) are averaged together to get the final output of the ensemble. Due to variance during training, ensembling can help boost the overall performance because different instantiations of the same model, having slightly different parameter settings, could help compensate for each other's mistakes.

3.3.2 Input Preprocessing

Like in Liu et al. (2016), the input recipe descriptions are preprocessed before being fed to the neural model. Input sentences are converted to lowercase and tokenized on punctuations. A vocabulary of size v is constructed from $(v - 2)$ most frequent tokens (including punctuation symbols) in the IFTTT training set. The vocabulary contains two special tokens: `UNK` token to capture out-of-vocabulary tokens and `PADDING` token to pad sentences if needed. Since the number of tokens that are input to the model are n, descriptions which have fewer than n tokens are padded with the `PADDING` token; for longer descriptions, $n/2$ tokens each are extracted from the beginning and the end.

3.3.3 Training Details

The neural models were trained on the IFTTT training set. Training strategy and hyper-parameter settings were same as the one used by Liu et al. (2016). The embedding size d was set to 50. The input size n was 25. A vocabulary of $v = 4000$ tokens was used.

During training, gradients with L_2 norm greater than 40 were scaled down to have norm 40. The Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.001 was used to train the models. Mini-batches of size 32 were used, and they were randomly shuffled. All trainable parameters were randomly initialized uniformly in the range $[-0.1, 0.1]$.

3.4 Natural Language Generation

The dialog system uses templates and IFTTT API documentation to translate its actions to utterances. Snippets from API documentation of channels and functions are used to translate the system's belief state into a comprehensible utterance. For example, to generate an utterance for requesting a confirmation for a trigger function, the system uses the description of that trigger function and the corresponding trigger channel from the API documentation. Examples of system utterances are given in Table 3.1 and 3.2.

3.5 Fields in the Dialog Setting

A dialog approach to recipe synthesis is particularly well-suited for getting fields' information from the user, because fields generally take custom values, such as path of a `dropbox` folder, or an email id, which users can conveniently specify after a system-initiative for that slot. Inspection of the dataset revealed that minute details such as information about fields are rarely present in recipe descriptions, which is why it is nearly impossible for a single-shot system that relies solely on the recipe descriptions to predict fields. This is why previous work on recipe synthesis that take fields into account do not report accuracy of full-recipe synthesis that includes fields (Quirk et al., 2015; Dong and Lapata, 2016). Such details can be easily furnished in the dialog setting; hence, a

Slot-value pair for confirmation	**System utterance**
Slot:*trigger_channel* Value:`weather`	Do you want an event on the *Weather Underground* service to trigger the applet?
Slot:*trigger_function* Value:`sunrise`	Do you want to trigger the applet *within 15 minutes of the sunrise in your location*?
Slot:*action_channel* Value:`hue`	Do you want to use the *Phillips Hue* service to perform the desired action every time the applet is triggered?
Slot:*action_function* Value:`blink_lights`	Do you want to *briefly turn your hue lights off then back on* every time the applet is triggered?

Table 3.1: Example system utterances for confirmation requests leveraging API documentation.

Slot	**System utterance**
trigger_channel	Which service should I use to look for an event when you want the applet to run?
trigger_function, given *trigger_channel* `weather`	Which event on the *Weather Underground* service should cause the applet to run?
action_channel	Which service should I use to perform the desired action every time the applet runs?
action_function, given *action_channel* `hue`	What should I do on the *Phillips Hue* service every time the applet runs?

Table 3.2: Example system utterances for requesting information about specific slots. For functions, the utterance uses context from the belief state in the form of value assigned to the associated channel.

dialog approach makes full-recipe synthesis, or generating complete, correct programs, possible.

For reasons discussed in Section 2.3, in this work, we take only channels and functions into account for recipe synthesis. However, incorporating fields is straightforward in the proposed dialog system. We can add a slot for each field associated with the trigger function and the action function of a recipe. Since fields are at the bottom of the slot-hierarchy, they will be taken up by the Dialog Manager after the slot for the corresponding function is filled. For fields that take custom values, user utterances corresponding to fields can be directly assigned to the slot. For fields that take values from a predefined list, a parser can be trained to map user utterances to values from that list.

3.6 Retraining NLU using Dialog

Another advantage of using a dialog approach to recipe synthesis is that it unlocks the possibility of continuous improvement of the underlying semantic parser through data extracted from the conversations (Artzi and Zettlemoyer, 2013; Thomason et al., 2015; Azaria et al., 2016; Weston, 2016). An improved semantic parser could potentially lead to shorter dialogs and higher rate of dialog success.

To this end, we extract training data from the dialogs. Opening user utterances and user utterances for each slot after a system-initiative (i.e. a request for a slot) in successful dialogs are paired with inferred slot values to retrain the models. For example, Table 3.3 shows the training data extracted from the conversation in Figure 3.1. Note that the labels that are paired with the descriptions in the extracted data are the ones that the dialog system inferred from the conversation and were affirmed by the user either after a confirmation request or by confirming dialog success at the end.

Analysis of models' predictions on the validation set revealed that the attention mechanism was rather good at attending to relevant parts of an utterance; the models failed because they often couldn't pick the correct channel or function among the similar ones. Therefore, we chose to tune only the non-attention parameters during retraining, keeping the attention-related parameters constant. This strategy is similar to the one employed by Liu et al. (2016) when they propose improving their models' performance on recipes that employ newly released channels and functions on IFTTT for which training data is scarce. Note that the goal of retraining in our system isn't limited to improving system's ability to parse utterances corresponding to only uncommon channels and functions. Through retraining, we hope to improve its general parsing performance, and in particular, as discussed in Section 4.4, on channels and functions for which the existing parser's confidences are low.

Description	**Label**
i want to be receive emails about flickr photos of me	*trigger channel*: `flickr`
i want to be receive emails about flickr photos of me	*trigger function*: `new_public_photo_tagged`
if there's a flickr photo of me	*trigger function*: `new_public_photo_tagged`
someone uploads a photo in which i am tagged	*trigger function*: `new_public_photo_tagged`
i want to be receive emails about flickr photos of me	*action channel*: `email`
i want to be receive emails about flickr photos of me	*action function*: `send_me_an_email`

Table 3.3: Training data for the four models extracted from the conversation in Figure 3.1.

Chapter 4

Experiments

4.1 Chapter Overview

This chapter describes the experiments conducted to evaluate the proposed dialog system and compare it with the single-shot approach to recipe synthesis. Further, it discusses an experiment to test our hypothesis of higher dialog success rates and shorter dialogs as a result of continuous parser improvement through conversations.

In all the experiments, we trained our parser on the training set of the IFTTT corpus. We evaluated our system on the gold subset of the IFTTT test set, consisting of recipes on which at least three humans presented with the recipe descriptions agreed with the true labels for triggers and actions. This subset contains 550 (out of the original 758) recipes. We chose to restrict ourselves to this subset because our experiments involved humans interacting with the dialog system to describe the recipe and answer its questions, and it was crucial that they themselves have a clear understanding of the recipe, something that might not be true for other recipes in the test set, which includes many recipes with unintelligible and non-English descriptions.

4.2 Experimental Setup

We used Amazon Mechanical Turk[1] to conduct our experiments. Users interacted with the system through a web-interface. They were provided with recipe descriptions from the IFTTT corpus to give them an understanding of the recipe. However, since they were not the original authors of the recipes, and since recipe descriptions could often be vague, we also explicitly provided them the details of channels and functions associated with the recipe to make sure that the recipe was clear to them.

As noted by Thomason et al. (2015), who used a similar interface to let users chat with a robot for performing navigation- and delivery-related tasks, information presented using words could linguistically prime the users. They avoided priming completely by presenting the information pictorially. However, in our domain, it is unclear how to succinctly describe a recipe to a user without using words. To avoid linguistic bias as much as possible, we used keywords used to identify channels and functions on IFTTT website (such as `hue`, `any_event_starts`, and `blink_lights`), which usually contain only content words necessary to give an indication of their functionality, but are somewhat distant from natural language. Additionally, we encouraged users to

[1]https://www.mturk.com/mturk/welcome

use their own words based on their understanding of the recipe and restricted direct usage of these keywords. The interface used for the experiments, along with a sample dialog, is shown in Figure 4.1.

4.3 Dialog Experiments

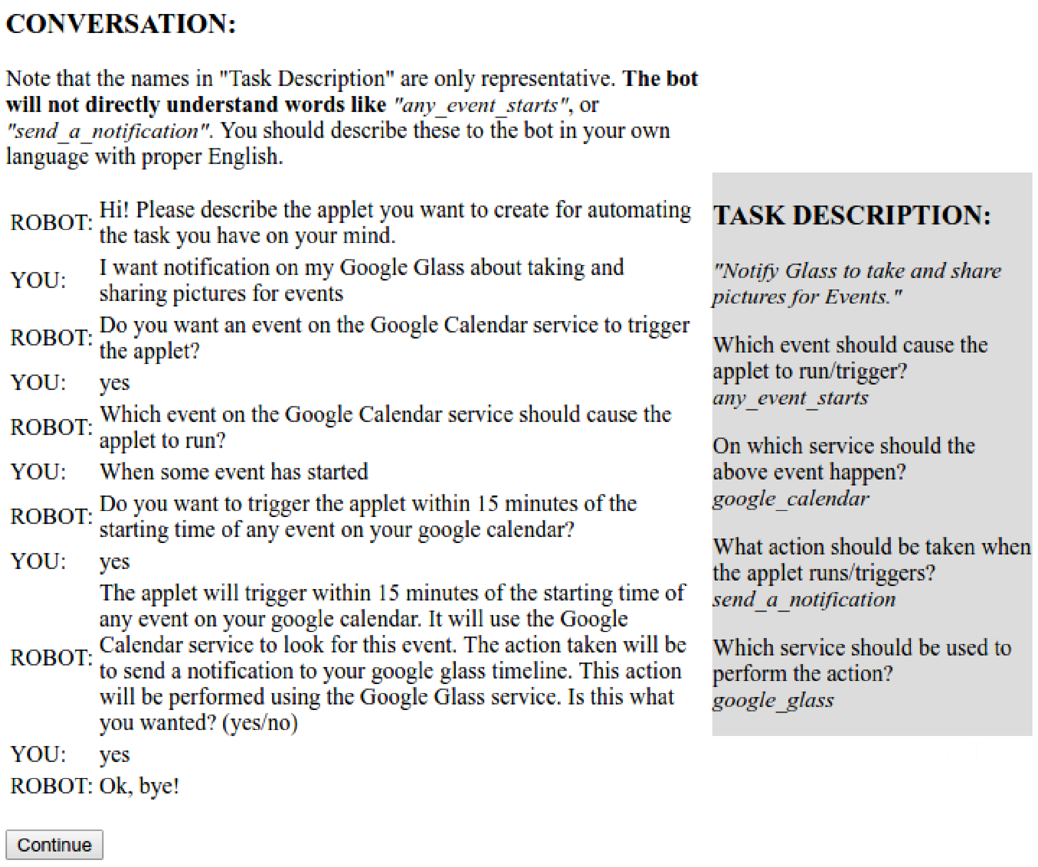
We conducted two experiments to evaluate the efficacy of dialog in synthesizing recipes from natural language. In both the experiments, the benchmark used is the best-performing model from Liu et al. (2016) which is currently the state-of-the-art on this task.

4.3.1 Constrained User-Initiative

To evaluate our system directly on the test set, we constrained the users to use the original recipe descriptions as their first utterance (i.e. the user-initiative) when they were asked by the system to describe the recipe. The users drove the rest of the conversations themselves, answering the system's questions based on their own understanding, using in their own words. This way, we can compare our results with previous work which use this set for evaluation.

4.3.2 Free User-Initiative

To emulate a more realistic setting in which users drive the entire conversation, including the user-initiative, we allowed the users to provide the initial recipe descriptions themselves based on their understanding of the recipe. Users


CONVERSATION:

Note that the names in "Task Description" are only representative. **The bot will not directly understand words like** *"any_event_starts"*, or *"send_a_notification"*. You should describe these to the bot in your own language with proper English.

ROBOT:	Hi! Please describe the applet you want to create for automating the task you have on your mind.
YOU:	I want notification on my Google Glass about taking and sharing pictures for events
ROBOT:	Do you want an event on the Google Calendar service to trigger the applet?
YOU:	yes
ROBOT:	Which event on the Google Calendar service should cause the applet to run?
YOU:	When some event has started
ROBOT:	Do you want to trigger the applet within 15 minutes of the starting time of any event on your google calendar?
YOU:	yes
ROBOT:	The applet will trigger within 15 minutes of the starting time of any event on your google calendar. It will use the Google Calendar service to look for this event. The action taken will be to send a notification to your google glass timeline. This action will be performed using the Google Glass service. Is this what you wanted? (yes/no)
YOU:	yes
ROBOT:	Ok, bye!

TASK DESCRIPTION:

"Notify Glass to take and share pictures for Events."

Which event should cause the applet to run/trigger?
any_event_starts

On which service should the above event happen?
google_calendar

What action should be taken when the applet runs/triggers?
send_a_notification

Which service should be used to perform the action?
google_glass

Figure 4.1: The dialog interface showing a sample conversation over a recipe with trigger `any_event_starts` on `google_calendar` and action `send_a_notification` on `google_glass`. The recipe description along with details of trigger and action were presented to the user alongside the dialog interface.

were encouraged to describe the recipe in their own words, but they weren't restricted from using the original recipe description itself as their opening utterance. To provide a fair comparison, we evaluated the benchmark model, which takes a single natural language description to synthesize a recipe, on the descriptions provided in the conversations (i.e. the first user utterances).

Experiment	**Liu et al. (2016)**	**Ours**
Constrained User-Initiative	85.28[2]	95.28
Free User-Initiative	66.0	81.45

Table 4.1: Accuracy of recipe synthesis (Channel + Function) measured on the gold subset.

Experiment	**Liu et al. (2016)**	**Ours**
Free User-Initiative before retraining	66.0	81.45
Free User-Initiative after retraining	66.0	82.55

Table 4.2: Accuracy of recipe synthesis (Channel + Function) measured on the gold subset before and after retraining the semantic parser.

4.3.3 Results

The results are summarized in Table 4.1. The dialog setting boosts the accuracy of recipe synthesis considerably over the single-shot setting in both the experiments: 10 point increase with constrained user-initiative and over 15

[2]The accuracy reported in Liu et al. (2016) is 87.5%; however, our implementation of their system was able to achieve only 85.28% accuracy. The discrepancy could be because of smaller training set (they had 68k recipes), a smaller test set (they had 584 recipes in gold subset), or randomness during training.

Experiment	**Dialog length**			
	Avg.	**Min.**	**Max.**	**Median**
Constrained User-Initiative	2.55	2.0	15.0	2.0
Free User-Initiative before retraining	4.04	2.0	22.0	4.0
Free User-Initiative after retraining	4.08	2.0	22.0	4.0

Table 4.3: Dialog length statistics measured in terms of number of user utterances.

point increase with free user-initiative. This shouldn't be surprising because the dialog setting gives the system an opportunity to query the user for clarification when the initial description wasn't clear enough. For example, the model doesn't correctly parse "From Facebook to Flickr" possibly because it is not explicitly mentioned in the description that a photo from Facebook needs to be posted to Flickr (a service on which only photos are posted). Similarly, the description "Blog post ==> Tweeted" doesn't specify which of the two blogging channels, Blogger and WordPress, to use as the trigger channel. Our dialog system explicitly asks the user for such missing or ambiguous information.

Surprisingly, the accuracy of both the single-shot approach and the dialog approach fell dramatically in the free user-initiative experiment. We contend that the reason behind this reduction is the difference between the two settings in which the recipe descriptions were created: The Constrained User-Initiative experiment used original descriptions which were written by the authors of the recipes with the aim of summarizing their recipes so that their functionality can be easily understood by others *without any assistance or* *clarification* from the author. The descriptions used in the Free User-Initiative experiment were provided by humans with the aim of describing the recipe to a system with the knowledge that the system can ask clarification questions. The former are expected to be more descriptive and self-contained than the latter, which hurts the performance of a single-shot approach more than that of the dialog approach. The larger average dialog length in the Free User-Initiative experiment corroborates this point (see Table 4.3). Additionally, the user utterances in the conversations are noisy, which further explains the reduction in accuracy of the two systems, especially our dialog system.

4.4 Parser Retraining

In addition to improving the accuracy of recipe synthesis, another advantage of the dialog setting is that it unlocks the possibility of continuous parser improvement through data extracted from the conversations.

Parser retraining would be most helpful when the data is extracted from conversations that involve channels and functions for which the existing parser's confidences are low. Therefore, we randomly sampled 100 recipes from an unused portion of the test set (whose descriptions were guaranteed to be in English and not totally incomprehensible) on which the confidence of existing parser is below β for at least two slots. About 130 data-points were extracted from the conversations with humans over these recipes, and the four models were retrained as described in Section 3.6.

For a direct comparison, the dialog system with retrained models was evaluated using the user utterances from conversations in the Free User-Initiative experiment, except when its actions deviated from the original ones (due to an improved NLU component) in which case new user utterances were obtained.

4.4.1 Results

The accuracy of retrained models is summarized in Table 4.2. While the retrained models didn't improve the single-shot accuracy, there was a marginal improvement of 1.1 point in the dialog setting. Analysis of the conversations revealed that this was because the retrained model had lower confidence for some channels and functions for which it initially had high priors (possibly due to their abundance in the training set).

On one hand, this helped the system avoid getting stuck in an impasse when it assigns an incorrect value to a slot with high confidence, without confirmation, and without any chance of recovery. On the other hand, this pessimism led to a slight increase in dialog lengths, contrary to our expectation that retraining would result in shorter dialogs (see Table 4.3).

Chapter 5

Conclusion and Future Work

In this work, we demonstrated the efficacy of using dialog for mapping natural language to short, executable computer code. We evaluated this idea in the domain of IFTTT recipes. The dialog system was able to engage the user in a dialog, asking questions to elicit additional information until it was confident in its inference, increasing the accuracy on this task over the state-of-the-art models that are restricted to synthesizing recipes in one shot by $10 - 15$ percentage points. Additionally, we demonstrated how data extracted from the conversations can be used for continuous parser learning.

In this work, we focus only on conditionals and, to some extent, procedure invocations. A natural extension would be to consider other programming constructs such as loops and sequence of execution. A slot-based dialog system could be used for loops or more extensive procedure invocations because the pieces of information needed for these primitive programming constructs can usually be predetermined. However, a slot-based approach to dialog might not be feasible for, say, generating the body of a procedure that involves declaration and manipulation of multiple variables. Hand-engineered features for the state space of the dialog system or a hand-coded policy might not scale to general-purpose code generation. Neural dialog systems and end-to-end dialog systems (Serban et al., 2015; Li et al., 2017) that can learn both code generation and dialog strategy from data could be used in such domains.

More elaborate techniques for parser retraining can also be explored. In this work, we paired user utterances with candidate slot-values that were eventually affirmed by the user to generate new training data. In addition, one could also extract negative training data from conversations by pairing user utterances with candidate slot-values that were denied by the user and train the neural models on these negative data-points with a modified loss function.

Parser learning through dialog has an unintended side-effect of inducing non-stationarity in the environment with respect to the Dialog Manager. A dialog policy built for the parser trained on the initial dataset might become sub-optimal after it has been retrained on data extracted from conversations. To account for this, online policy learning through dialog can be integrated with parser learning (Padmakumar et al., 2017).

Appendix A

An RNN processes a set of inputs $\mathbf{x_1}, \ldots, \mathbf{x_n}$ sequentially yielding at time t a d-dimensional vector $\mathbf{h_t}$, computed as:

$$\mathbf{h_t} = \tanh\left(\mathbf{W_{xh}x_t} + \mathbf{W_{hh}h_{t-1}} + \mathbf{b_h}\right) \tag{A.1}$$

Long Short-Term Memory (LSTM) is a type of RNN having sophisticated dynamics that aid in memorizing information for a longer period. Its dynamics are governed by the following set of equations, borrowing notation from Zaremba et al. (2014)

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} \mathbf{h_t^{l-1}} \\ \mathbf{h_{t-1}^{l}} \end{pmatrix} \tag{A.2}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g \tag{A.3}$$

$$h_t^l = o \odot \tanh(c_t^l) \tag{A.4}$$

Here, $T_{n,m} : \mathcal{R}^n \to \mathcal{R}^m$ represents an affine transformation $\mathbf{W}\mathbf{x} + \mathbf{b}$ for some $\mathbf{W}$ and $\mathbf{b}$, and $\mathbf{c}_t^l$ is the state of the memory cell at level l at timestep t. The network stores memory in these cells.

Bibliography

2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers Inc.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics* 1(1):49–62. https://homes.cs.washington.edu/ lsz/papers/az-tacl13.pdf.

Amos Azaria, Jayant Krishnamurthy, and Tom M. Mitchell. 2016. Instructable intelligent personal agent. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, AAAI'16, pages 2681–2689. http://dl.acm.org/citation.cfm?id=3016100.3016277.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473. http://arxiv.org/abs/1409.0473.

I. Beltagy and Chris Quirk. 2016. Improved semantic parsers for if-then statements. In *Proceedings of the 54th Annual Meeting* *of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 726–736. http://www.aclweb.org/anthology/P16-1069.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*. http://www.aclweb.org/anthology/D13-1160.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 33–43. http://www.aclweb.org/anthology/P16-1004.

Michal Gordon and Cynthia Breazeal. 2015. Designing a virtual assistant for in-car child entertainment. In *Proceedings of the 14th International Conference on Interaction Design and Children*. ACM, IDC '15, pages 359–362. http://doi.acm.org/10.1145/2771839.2771916.

Sumit Gulwani. 2012. Synthesis from examples: Interaction models and algorithms. In *Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE Computer Society, SYNASC '12, pages 8–14. http://dx.doi.org/10.1109/SYNASC.2012.69.

Sumit Gulwani and Mark Marron. 2014. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, SIGMOD '14, pages 803–814. http://doi.acm.org/10.1145/2588555.2612177.

Pirkko H Harvey, Edward Currie, Padma Daryanani, and Juan C Augusto. 2015. Enhancing student support with a virtual assistant. In *International Conference on E-Learning, E-Education, and Online Training*. Springer, pages 101–109. https://link.springer.com/chapter/10.1007/978-3-319-28883-3_13.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780. http://bioinf.jku.at/publications/older/2604.pdf.

Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education India.

Rohit J. Kate, Yuk Wah Wong, and Raymond J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI Press, AAAI'05, pages 1062–1068. http://dl.acm.org/citation.cfm?id=1619499.1619504.

Patrick Kenny, Thomas Parsons, Jonathan Gratch, and Albert Rizzo. 2008. Virtual humans for assisted health care. In *Proceedings of the 1st International Conference on PErvasive Technologies Related to Assistive Environments*. ACM, PETRA '08, pages 6:1–6:4. http://doi.acm.org/10.1145/1389586.1389594.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* https://arxiv.org/abs/1412.6980.

Tao Lei, Fan Long, Regina Barzilay, and Martin Rinard. 2013. From natural language specifications to program input parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1294–1303. http://www.aclweb.org/anthology/P13-1127.

Xuijun Li, Yun-Nung Chen, Lihong Li, and Jianfeng Gao. 2017. End-to-end task-completion neural dialogue systems. *arXiv preprint arXiv:1703.01008* https://arxiv.org/abs/1703.01008.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 599–609. http://www.aclweb.org/anthology/P16-1057.

Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. 2016. Latent attention for if-then program synthesis. In *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pages 4574–4582. http://papers.nips.cc/paper/6284-latent-attention-for-if-then-program-synthesis.pdf.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, pages 285–294. http://aclweb.org/anthology/W15-4640.

Mehdi Manshadi, Daniel Gildea, and James Allen. 2013. Integrating programming by example and natural language programming. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, AAAI'13, pages 661–667. http://dl.acm.org/citation.cfm?id=2891460.2891552.

Aishwarya Padmakumar, Jesse Thomason, and Raymond J. Mooney. 2017. Integrated learning of dialog strategies and semantic parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017)*. http://www.cs.utexas.edu/users/ai-lab/pub-view.php?PubID=127615.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (July):878–888. http://www.aclweb.org/anthology/P15-1085.

Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. 2014. Programming by example using least general generalizations. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI Press, AAAI'14, pages 283–290. http://dl.acm.org/citation.cfm?id=2893873.2893919.

Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. 2015. Compositional program synthesis from natural language and examples. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, IJCAI'15, pages 792–800. http://dl.acm.org/citation.cfm?id=2832249.2832359.

Alan Ritter, Colin Cherry, and William B. Dolan. 2011. Data-driven response generation in social media. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, EMNLP '11, pages 583–593. http://dl.acm.org/citation.cfm?id=2145432.2145500.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2015. Hierarchical neural network generative models for movie dialogues. *CoRR* abs/1507.04808. http://arxiv.org/abs/1507.04808.

Lifeng Shang, Zhengdong Lu, and Hang Li. 2015. Neural responding machine for short-text conversation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 1577–1586. http://www.aclweb.org/anthology/P15-1152.

Lanbo She, Shaohua Yang, Yu Cheng, Yunyi Jia, Joyce Chai, and Ning Xi. 2014. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*. Association for Computational Linguistics, pages 89–97. http://www.aclweb.org/anthology/W14-4313.

Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. 2002. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research* 16:105–133. https://www.jair.org/media/859/live-859-1983-jair.pdf.

Sunbeom So and Hakjoo Oh. 2017. Synthesizing imperative programs for introductory programming assignments. *CoRR* abs/1702.06334. http://arxiv.org/abs/1702.06334.

Armando Solar-Lezama. 2008. *Program synthesis by sketching*. University of California, Berkeley.

Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pages 2440–2448. http://papers.nips.cc/paper/5846-end-to-end-memory-networks.pdf.

Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, IJCAI'15, pages 1923–1929. http://dl.acm.org/citation.cfm?id=2832415.2832516.

Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pages 2773–2781. http://papers.nips.cc/paper/5635-grammar-as-a-foreign-language.pdf.

Jason E Weston. 2016. Dialog-based language learning. In *Advances in Neural Information Processing Systems 29*, Curran Associates, Inc., pages 829–837. http://papers.nips.cc/paper/6264-dialog-based-language-learning.pdf.

Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. 2013. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*. Association for Computational Linguistics, pages 404–413. http://www.aclweb.org/anthology/W13-4065.

Bo Wu and Craig A. Knoblock. 2015. An iterative approach to synthesize data transformation programs. In *Proceedings of the 24th International Conference on Artificial Intelligence*. AAAI Press, IJCAI'15, pages 1726–1732. http://dl.acm.org/citation.cfm?id=2832415.2832489.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Type- and Content-Driven Synthesis of {SQL} Queries from Natural Language. *CoRR* abs/1702.01168. http://arxiv.org/abs/1702.01168.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. 2016. Neural enquirer: Learning to query tables in natural language. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. AAAI Press, IJCAI'16, pages 2308–2314. http://dl.acm.org/citation.cfm?id=3060832.3060944.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *The 55th Annual Meeting of the Association for Computational Linguistics (ACL)*. https://arxiv.org/pdf/1704.01696.pdf.

Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179. http://ieeexplore.ieee.org/document/6407655/.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* https://arxiv.org/abs/1409.2329.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*. AAAI Press/MIT Press, Portland, OR, pages 1050–1055. http://www.cs.utexas.edu/users/ai-lab/?zelle:aaai96.

Vita

Shobhit Chaurasia was born in 1993 in Rajasthan, India. He completed his high school from SRDAV Public School, Delhi. He received his Bachelor of Technology degree from Indian Institute of Technology, Guwahati in Computer Science in 2015. He joined The University of Texas at Austin in Fall 2015 to pursue his Master of Science degree, focusing on Machine Learning in general, and Natural Language Processing and Information Retrieval in particular. He has done internships at Adobe Research Labs, Noida, India and BloomReach, Mountain View, California. After graduation, he will be returning to work full-time at BloomReach.

Address: sc.shobhit@gmail.com

This thesis was typeset with LaTeX 2_ε[1] by the author.

[1]LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.