Multi-Task Neural User Simulator for Task Oriented Dialogue System

To Eun Kim

Supervisor:
Aldo Lipani

This report is submitted as part requirement for the

MEng Degree in Computer Science

at

University College London

It is substantially the result of my own work

except where explicitly indicated in the text

Department of Computer Science

University College London

July 10, 2022

I, To Eun Kim, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Public availability will be delayed while the content is submitted for reviewing and publication at scientific venues.

Abstract

Task-oriented dialogue system (TOD) is gaining attention in both academia and industries as it provides an interactive information access to the users. For training and evaluation of the system, user simulators have been researched in the literature. This thesis comprises of two novel work: 1) a multi-task neural user simulator, and 2) a multi-modal TOD with state-machine like architecture. First, the multi-task neural user simulator has achieved a state-of-the-art performance in user satisfaction score and action prediction tasks. Also, this work has found that three dialogue research tasks (user satisfaction score prediction, user action prediction, and user utterance generation) can synergize each other, once jointly trained by multi-task learning. Next, we present a state-machine like architecture for a multi-modal TOD. We discuss multiple conversational tactics as well as how the system is developed, tested, deployed and evaluated to ensure seamless and engaging conversations with users. As both work are extensible and can be adapted to many research, we propose three future work before concluding the thesis.

Acknowledgements

I would like to first thank Dr. Aldo Lipani for his superb supervision. It was a huge privilege for a taught-master student to have a desk with a monitor at UCL office. The kindness of him has allowed me to have an excellent environment, where I could focus on my research as well as coursework. He has not only been my supervisor but also a co-worker during the Alexa Prize TaskBot Challenge. Frequent meetings and non-stop working across different time zones with team members have made our team progress to the quarter-final. Moreover, it was his supervision that made my research on user simulation be published and presented in the 2022 ACM SIGIR conference.

I would like to thank Osman Ipek, and the whole Alexa Prize team for their support during the challenge. Continuous technical help, including computing resources and periodic feedback had helped our task bot improve day by day.

I would like to thank all the members of UCL Web Intelligence group. Professor Emine Yilmaz and Dr. Qiang Zhang have welcomed me to the group and helped my summer research at the group be funded by the CS department. Valuable talks with Jerome Ramos, Zhengxiang Shi, Hossein Rahmani, Xiao Fu, and Ruizhe Li have let me delve into various branches of research in Conversational Artificial Intelligence.

I thank all the professors who have provided us with outstanding lectures and assignments. I thank professor Marianna Obrist for supervising my bachelor's project. I also thank all my friends who spent countless days and nights with me on the coursework back-to-back.

Last but not least, I thank my family for their endless love and support.

Dedication

To my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Conversational Artificial Intelligence (AI) has been a long-standing area of exploration in the research community and has now penetrated both academia and industries with products such as Alexa and Siri. In particular, in recent years, there has been growing interest in the field of task-oriented dialogue system (TOD). Unlike open domain dialogue system, which deals with open-ended knowledge without any pre-defined tasks [2], TOD is designed with specific domains and tasks in mind.

One of the main challenges in building TOD is gathering enough user data to train a model that is able to generalize and reach a satisfactory performance in terms of user satisfaction [3]. Furthermore, the interactive nature of a dialogue system makes its evaluation challenging [4]. In order to mitigate the training and evaluation problems, literature has been seeking answers in dialogue user simulator that can mimic users' behavior [4, 5]. *In this thesis*, to contribute to this area of research, we propose a state-of-the-art user simulator that predicts user's satisfaction score, action, and next utterance.

Current research in TOD can be classified into modularized approach and end-to-end approach. In modularized approach, TOD is constructed with multiple modules, which are specialised in their own specific natural language processing (NLP) tasks. This modules are then coupled together to provide a complete dialogue experience. On the other hand, end-to-end approach attempts to build a system with a single model, which takes user utterance and directly outputs a system response in natural language [6]. *In this thesis,* we focus on improving the modularized approach and develop a state-machine like framework for multi-modal task-oriented dialogue systems.

1.1 Literature Review

Before diving into the main part of this thesis, chapter 2 surveys the advances of language models (LM), conversational user simulation, and the studies of task-oriented dialogue systems (TOD) in order to provide all the necessary background information that help the understanding of the two novel work presented in chapter 3 and 4.

For LM (section 2.1), readers are particularly encouraged to focus on the research in large-scale pre-trained models, such as BERT [7], GPT [8], and T5 [9], all of which are based on the Transformer [10] language model. We look into the architecture of these models and see how they can be applied to the dialogue systems. **For user simulation** (section 2.2), we see the past attempts on constructing the simulators, explore the use cases of user simulation in dialogue systems and how users' satisfaction level can be modeled. **For TOD** (section 2.3), we first scrutinize numerous modules and approaches that consist of the dialogue system which are the paramount background knowledge for chapter 4. Next, we reveal four main challenges in TOD research, and show how these problems are mitigated in the following two chapters.

1.2 User Simulation

In task-oriented dialogue systems, a human-like user simulator that can predict users' satisfaction level and next action can help refine dialogue strategies and evaluate conversations. Moreover, in reinforcement learning (RL) based dialogue policy training, user simulator can act as an environment that gives rewards to a learning agent [6]. However, user simulators have been less researched in literature compared to the other areas, and little work has explored user simulators that are able to handle multiple tasks in a multi-task training strategy.

In chapter 3, we convince the importance of user simulators in TOD, and propose a novel multi-task based neural user simulator that predicts users' satisfaction score and next action while also jointly generating users' next utterance.

In particular, we show that:

1. our deep text-to-text multi-task neural model achieves state-of-the-art (SOTA) performance in user satisfaction and action prediction,

2. adding utterance generation as an auxiliary task can boost the prediction performance of satisfaction and action prediction via positive transfers across the tasks, and

3. via ablation studies, all three tasks (satisfaction prediction, action prediction, and utterance generation) synergize each other.

This work has been accepted to the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22) [1] as a short paper. Source code is made available at `https://github.com/kimdanny/user-simulation-t5`.

1.3 Task-oriented Dialogue Systems

Alexa Prize is an international competition for university students committed to the development of the field of conversational AI. Started from 2017, the Alexa Prize team has launched a series of SocialBot Grand Challenge, where the goal of the challenge was to design and build an open-domain dialogue system that is to be deployed on the Alexa platform. In 2021, the Alexa Prize team launched a new year-long competition, a TaskBot Challenge, where the focus of the challenge was in building a multi-modal task-oriented dialogue system specialized in cooking and do-it-yourself (DIY) knowledge [11]. UCL Condita team was selected in the 2021 TaskBot Challenge, and the author of this thesis is one of the 7 members of the team [12].

In chapter 4, we present a **CO**oking-a**N**d-**DI**y-**TA**sk-based task-oriented dialogue system (Condita) built for the 2021 Alexa Prize TaskBot Challenge. Condita

[1] `https://sigir.org/sigir2022/`

provides an engaging multi-modal agent that assists users in cooking and DIY tasks. Our main goal was to generate engaging dialogue and provide an effortless user experience in order to create a memorable and enjoyable experience for users. We discuss Condita's system architecture and analyze the various conversational strategies that allowed us to achieve excellent performance throughout the competition. This includes how we tackle the dynamics of multi-turn dialogue and integrate domain knowledge, as well as how online evaluation of the system can be incorporated into the continuous development cycle. We also explain how this large interactive software system could be deployed to the production ensuring the low latency.

This joint work with 5 PhD students and Dr. Aldo Lipani [2] at UCL Web Intelligence group [3] will be published in the Alexa Prize Proceedings [4] after the competition. Note that this thesis only includes the contributions by the author of the thesis, To Eun Kim, unless explicitly indicated in the text.

---

[2] supervisor of both Alexa Prize TaskBot Challenge and this thesis

[3] `http://wi.cs.ucl.ac.uk`

[4] `https://www.amazon.science/alexa-prize/proceedings`

Chapter 2

Literature Review

2.1 Language Modeling

This section investigates the advances of Language Models (LM) through the lens of research in task-oriented dialogue systems (TOD).

2.1.1 Frequency-based Language Models

Language Model (LM) is the foundation of various Natural Language Processing (NLP) tasks. In statistical language modeling, LM represents the text information into a computable format by allocating probabilities to the sequence of words or characters. In other words, to represent a sequence S of m number of words $(w_1 \cdots w_m)$, the LM assigns probability $P(S)$ to the sequence S as follows [13]:

$$\begin{aligned} P(S) &= P(w_1 w_2 \cdots w_m) \\ &= P(w_1) P(w_2 \mid w_1) \cdots P(w_m \mid w_1 w_2 \cdots w_{m-1}) \\ &= \Pi_{i=1}^{m} P(w_i \mid w_1, w_2, \cdots, w_{i-1}) \end{aligned}$$

Here, calculating the probability of the sequence can be based on the frequency ($freq$) of the words in the corpus, i.e.,

$$P(w_i \mid w_1, w_2, \cdots, w_{i-1}) = \frac{freq(w_i)}{freq(w_1 \cdots w_m)}$$

This method creates extensive amounts of parameters. Also, as the sequence of words gets longer, the probability that w_i exists after $w_1 \cdots w_m$ becomes lower. In a real world situation, where the size of the corpus is huge, this can let most of the $P(S)$ value close to 0, which makes the representation sparse. Therefore, an approximate method, N-gram model, was introduced [13].

N-gram model considers n number of continuous words as one token. When assigning the probability of w_i, the model only takes previous n number of words into consideration, i.e., $P(w_i \mid w_{i-n}, \cdots, w_{i-1})$. Variation of the model depends on the parameter n; uni-gram ($n = 1$), bi-gram ($n = 2$), and tri-gram ($n = 3$) models can be devised. Note that the N-gram model does not entirely solves the sparsity problem, although it can alleviate it [13].

Another frequency-based model that generally performs better than N-gram is the Term Frequency–Inverse Document Frequency (TF-IDF) model [14]. TF-IDF is a combination of term frequency (TF) and inverse document frequency (IDF). First, TF, as the name suggests, compute the frequency of the term with respect to a given document. This is calculated by dividing the number of occurrences of the word by the length of the document. Second, IDF is for assigning weights to the words [14]. It is calculated by $\log\left(\frac{D}{df_t}\right)$, where D is the set of each document d, and df_t is the aggregation of documents that includes the term t. Finally, the TF-IDF is the product of TF and IDF as below:

$$TF - IDF(t, d, D) = TF(t, d) \times \log\left(\frac{D}{df_t}\right) \tag{2.1}$$

Although it assigns weights to each word unlike N-gram models, since it is based on the Bag-of-Word approach, TF-IDF cannot capture the order of the words, i.e., lack of semantic information representation [15], and this problem was dealt in the later models, such as GloVe[16] and ELMo[17].

2.1.2 Neural Network-based Language Models

In order to solve the problems of frequency-based LMs, neural network-based language models have been introduced. The following models can learn linguistic features and continuous representations [13].

Feed-forward Neural Network (FFNN) was the first artificial neural network developed in the research. It is called feed-forward as the model lets the information in the network flow only in one direction from the input nodes to output nodes. Using perceptron as a fundamental building block of the network, variations of FFNN have been introduced, including kernal perceptron and multi-layer perceptron [18].

Recurrent Neural Network (RNN) is an adaptation of FFNN for sequential data like natural language, thus can introduce a concept of discrete time dynamics [19]. At time step t, with an input $\mathbf{x}_t$, hidden state $\mathbf{h}_t$, and an output $\mathbf{y}_t$, the RNN can be constructed as:

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi_h\left(\mathbf{W}^\top \mathbf{h}_{t-1} + \mathbf{U}^\top \mathbf{x}_t\right) \tag{2.2}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t, \mathbf{x}_t) = \phi_o\left(\mathbf{V}^\top \mathbf{h}_t\right), \tag{2.3}$$

where f_h, f_o, $\mathbf{W}$, $\mathbf{U}$, $\mathbf{V}$ are state transition function, output function, transition matrix, input matrix and output matrix, respectively, and ϕ_h and ϕ_o are non-linear functions, such as sigmoid or hyperbolic tangent function [19].

Due to its recurrent nature, in long data sequences, when the gradients are propagated through time, gradients can become extremely small to the point where parameter updates become insignificant, i.e., the vanishing gradient issue [15]. To address this problem, two alternative networks, Long Short-Term Memory (LSTM) [20] and Gated Recurrent Unit (GRU) [21] have been presented.

2.1.3 Transformer-based Language Models

In 2017, Vaswani et al. [10] have presented, the Transformer, a sequence-to-sequence (encoder-decoder) model, based on a stack of multi-head-attention and point-wise FFNN modules. This structure, completely freed from the recurrent architecture, has renewed a number of previous state-of-the-art (SOTA) benchmarks in NLP tasks. Most importantly, this has spurred the recent advances of NLP, resulting in many remarkable variations in the field [22].

After the introduction of the Transformer architecture [10], many encoder-based language models, such as BERT [7] were used in various tasks in NLP. These models are usually pre-trained with large corpora in a self-supervised way before being fine-tuned to specific downstream tasks. As these encoder-only architectures are more suitable for discriminative tasks, decoder-based models, such as GPT-2 [8] are used for generative tasks. On the other hand, Raffel et al. [9] experimented with an encoder-decoder model, named T5, where the target function is mapped into text. For example, for a regression task, the model is trained to generate a text like "v" where $v \in \mathbb{R}$, from a input sequence prepended with a task-specific prefix. In other words, it converts every task into a text generation task, i.e., a regression task becomes a generation of a text-formatted target. This characteristic of T5 makes its use easy in a multi-task setting without the need to define task-specific heads.

The T5 model has been used in dialogue response generation tasks [23, 24, 25]. Kale and Rastogi [23] transformed every encoded system action into natural language, then concatenated them into a sequence to pass it into the T5 model. The T5 model is then fine-tuned to fuse the input sentences into one. Lin et al. [24] fine-tuned T5 in an end-to-end fashion for a dialogue response generation task, which resulted in SOTA performance with fewer human annotations. Ben-David et al. [25] used the T5 model in a multi-task setting for users' intent prediction, training with the utterance reordering and generation tasks. They showed that these auxiliary tasks improved the performance of the users' intent prediction task. Also, they have proven the importance of generating users' utterances in predicting users' intents.

2.1.4 Evaluation Metrics

In text classification task, accuracy, precision, recall, and F1-score are the most widely used evaluation metrics. These metrics are based on the confusion matrix, which represents the values of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). From the confusion matrix, the four metrics are defined as below [15]:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

$$F1 - score = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{2.7}$$

In text generation task, automatic evaluation is widely used along with human evaluation [26]. For **word-based evaluation**, Perplexity (PPL) [27], Bilingual Evaluation Understudy (BLEU) [28], and Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [29] are popular [26]. PPL is defined as

$$2^{-\frac{1}{N}\sum_{t=1}^{N}\log_2 LM(w_t|w_1 \cdots w_{t-1})}, \tag{2.8}$$

where N is the size of the corpus. The lower the PPL, the better the performance. BLEU and ROUGE are based on the N-gram. They both calculate the overlaps of N-gram between the generated text and the reference. However, BLEU is a precision-based metric, while ROUGE is a recall-based metric [26]. For **embedding-based evaluation**, Average Embedding metric can be used. For both hypothesis (generated sentence) and reference, a sentence-level embedding can be computed by taking an average of each word embeddings in the sentence - e_h (sentence embedding of hypothesis) and e_r (sentence embedding of reference). Then, one can compute the cosine similarity between the two embeddings, i.e., $\cos(e_h, e_r)$ [26].

2.1.5 Multi-Task Learning in NLP

The idea of Multi-Task Learning (MTL) stems from the human ability, where we can learn multiple related tasks together. Given n related learning tasks $\{T_i\}_{i=1}^{n}$ MTL learns n tasks simultaneously under the assumption that MTL training setting can increase the model performance of each task T_i, compared to training n different models separately [30]. Researchers have found that MTL models' knowledge sharing ability can allow them to outperform their single-task counterparts with less chance of overfitting in each task, and this is also found in many NLP research [9, 31, 32, 33, 34, 35, 36].

In supervised MTL, multiple tasks can consist of different types of supervised learning, such as regression and classification. Heterogeneous MTL is deemed to have different types of tasks, while homogeneous MTL consists of tasks of single type [30]. In this regard, the T5 model mentioned in the section 2.1.3 can effectively perform both homogeneous and heterogeneous MTL due to its text-to-text characteristic. In other words, as long as the task-specific prefixes present, multi-task learning is equivalent to simply mixing the datasets together [9].

MTL is also widely used in model pre-training, so called a multi-task pre-training. This approach has been advocated in both T5 [9] and MT-DNN [31] models. There have been attempts to use the multi-task pre-training strategy for dialogue systems. Xu et al. [35] have shown that a BERT-based self-supervised MTL approach can significantly improve the performance on the multi-turn response selection task. This improvement was achieved by adding four auxiliary tasks. All four auxiliary tasks were proven to be useful as removing any of them decreased the performance of the model on the main task. Zhang et al. [36] developed DialogBERT. They pre-trained BERT with five self-supervised tasks, which were found to be helpful in several other dialogue systems-related tasks.

2.2 User Simulation in Dialogue Systems

2.2.1 Uses Cases of User Simulators

Dialogue user simulator has a number of use cases in dialogue systems research. First, to alleviate the data scarcity issue, simulators can be used to generate a vast amount of synthetic dialogues. Acharya et al. [37] used user simulator to generate training data by feeding the simulator with a few seed dialogues. A simulator can also be used when training a dialogue system, particularly dialogue manager (DM) [38], and this is usually done by reinforcement learning (RL) algorithms

[3, 39], such as policy optimization [39] and Dyna-Q [40, 41]. In the RL-based training, a simulator acts as an environment that affects the agent's (DM) behavior by the corresponding rewards. Another use case of the user simulator is the evaluation of the dialogue system [3, 42]. Lipani et al. [5] showed that a user simulator can interact with the dialogue systems while tracking the generated dialogues with its predicted satisfaction scores.

2.2.2 Architecture of the Simulator

One of the early attempts to simulate users' behaviour was an Agenda-Based User Simulation (ABUS). The agenda represents the users' state as a stack of user actions. This probabilistic approach lets simulator randomly set a user's goal and keeps it unchanged during the whole dialogue. During the conversation, the simulator generates simulated user utterances. Then, the agent's (dialogue system's) task is to let users achieve their goals by gradually figuring the users' needs out [43].

Asri et al. [44] employed sequence-to-sequence [45] neural network to generate a sequence of dialogue acts (intents and slots). Similar work by Crook and Marin [46] has used the sequence-to-sequence network for training a user simulator that generates users' utterance. They have followed the common approach of data preparation, where user and system dialogues are concatenated alternately. Unlike the previous work, the simulator was trained in a natural-language-to-natural-language fashion. The simulator was designed to use GRU [21] as encoder and LSTM [20] as decoder. When assessed by both automatic and human evaluation, they have shown that the sequence-to-sequence-based model works better than the traditional approaches in simulating users. This finding is aligned with the later research that data-driven deep neural networks have superior performance to the traditional approaches in user simulations [39].

2.2.3 User Satisfaction Modeling

Modeling users' satisfaction is an important part of designing a user simulator. If a simulator can track the turn-level satisfaction score of users, the automatic evaluation of the system's response based on the user satisfaction score becomes possible [3, 47]. Sun et al. [1] developed a model to predict users' satisfaction scores and actions by cascading one model specialized in the prediction of users' satisfaction scores to another model specialized in the prediction of users' actions. The latter uses the output of the former as an input. The results show that the Hierarchical Gated Recurrent Units (HiGRU) [48] and BERT [7] based models perform well in these two tasks. BERT especially performed well on action prediction, and its cross-domain performance on user satisfaction was notable compared to that of the other models in the experiment. This analysis was performed using the conversational dataset named User Satisfaction Simulation (USS). This dataset contains turn-level annotations of users' satisfaction scores (on a 5-point scale) and actions [1], and it is built as an extension of 5 conversational datasets (MultiWOZ 2.1 [49], SGD [50], CCPE [51], JDDC [52], and ReDial [53]).

Another approach to model user satisfaction is to train a dialogue system with users' feedback in an online setting [54]. While conversing with users, the system attempts to predict their users' satisfaction scores after every user utterance. However, the satisfaction score prediction is performed after each user utterance, so-called, an after-utterance (AU) prediction. This is in contrast with the before-utterance (BU) prediction. The AU prediction of user satisfaction can sometimes be easier than the BU prediction, since there may be patterns in user utterances that indicate a clear dissatisfaction, e.g., "What are you talking about?" [54]. However, the BU prediction, although more difficult, can be used by the system to prevent the user from having a bad experience beforehand by changing the course of the dialogue towards a less unsatisfactory one.

2.3 Task-oriented Dialogue Systems

Approaches in building a task-oriented dialogue system (TOD) can be divided into two categories: modularized approach and end-to-end approach. A modularized system comprises of the following modules: natural language understanding (NLU), dialogue state tracking (DST), dialogue policy learning/generation, and natural language generation (NLG). DST and dialogue policy module are often grouped as a dialogue management (DM) module. On the other hand, end-to-end system employs a single model that takes dialogue history as input and outputs a natural language system response. Since there exists several major challenges left unsolved in designing a well-performing end-to-end dialogue system, current literature and industries are more focused on improving the modularized approach [6].

2.3.1 Modules and Approaches

Natural Language Understanding (NLU) is the first task that a dialogue system should process once users' utterance is input to the system. The aim of NLU is twofold: classification of user's intent and detection of zero or more slots in the sentence. The former is a classification task (intent classification), while the latter is a sequence labeling task (slot filling) [6]. In the literature, joint-training of intent classification and slot filling has shown promising results. Liu and Lane [55] have jointly trained a sequence-to-sequence model with both tasks. By using an attention-based bi-directional LSTM as encoder, the context could be well represented. Then, two task specific RNN decoders, each of which is for intent classification, and slot filling, were concatenated. This model structure allowed an end-to-end joint training of both tasks, giving high F1-score and low intent error rate. Using similar training methodology, but replacing RNNs with BERT, Chen et al. [56] have also shown the excellence of joint training of intent classification and slot filling. In the analysis of impact of intent and slot in NLU, slot filling turned out to be more important in increasing the performance of NLU, compared to intent classification. Therefore, the system may as well put higher weights on slot filling than intent classification [57].

Dialogue State Tracking (DST) is the next task that the dialogue system should process after the NLU. It is the core component of the system that maintains and tracks users' states through the history of the dialogue. DST module is given with dialogue history, users' intents, and slots (output from NLU) and decides the state of the current turn [58]. The state at time t can be regarded as the representation of the dialogue history until t, i.e., Markov Property [6, 59]. Mrkšić et al. [60] have incorporated belief states into the state representation, which comprises of slot values. However, this methods performs poorly when faced with unseen values. To solve this issue, Chao and Lane [61] have presented, a BERT-DST, focusing on a situation, where the ontology is veiled to the state tracker. This work has surpassed most of the prior work on DST the benchmarks.

Dialogue Policy Learning is for deciding which action to take given the current dialogue previously state set by DST. Yang et al. [62] has introduced a joint learning of NLU and Dialogue Manager (DM). Their bi-direction LSTM network removed the DST module, storing the dialogue states within the LSTM state cells as latent variables. For policy learning, they named the task as system action prediction (SAP), and showed that the joint learning of NLU in SAP increased the performance of user intent classification. As mentioned in the last section, conversational states have Markov property. Therefore, it is possible to form a Markov Decision Process (MDP) [59] for training a DM. Other reinforcement learning (RL) based methods were also extensively explored, including model-free RL (DQN and Policy Gradient) [6]. However, since RL requires great number of conversations, user simulation is used to provide its training environment. Deep Dyna-Q is an example which used user simulation in RL-based DM training [41].

Natural Language Generation (NLG), often called as response generation in TOD, is the final step that converts dialogue acts represented in a semantic form into a natural language. There are two ways to map the acts into the natural language: template-based and language model (LM)-based approach [63]. Since template-based approach falls short in fluency, statistical LM-based approach is preferred. To convey the semantics in the dialogue acts, a semantically conditioned LSTM (SC-LSTM) [64] was proposed. After the emergence of the large pre-trained LMs, SC-GPT model [63] has demonstrated a significant improvements in NLG, measured by both automatic evaluation metrics and human evaluations.

Response Selection. If there exists multiple candidates of generated response, then ranking of the responses is needed by matching the dialogue context with responses. BERT-FP [65], which uses a fine-grained post-training method to response selection serves as a SOTA model in this field.

2.3.2 Current Challenges in TOD

There are specific issues that arise in task-oriented systems. First issue originates from the importance of tracking **multi-turn dialogues** [6]. Unlike open domain dialogue system, where the system does not have to strictly track users' goal, in TOD, a role of the DST and policy learning is critical to the system performance. In chapter 3, we show how pre-trained language model can learn the representation of the multi-turn dialogue history. In chapter 4, we show how this problem can be handled by policies on a state machine like architecture in conjunction with a key-value pair database.

Second, **integrating domain knowledge** into a system is hard [6]. Most of the modularized system has its own knowledge base (KB) for querying and retrieving domain-specific information. However, as demand for multi-domain TOD grows, ontology integration tasks are becoming harder [49, 50]. We show how KB can be integrated into the TOD system in chapter 4.

Next is **data efficiency** [6]. Recent models in dialogue systems are dependent on data-driven neural architecture. However, obtaining a large dialogue dataset is very difficult and costly (needs human annotations), and this is more so in TOD. Therefore, low-resource training is getting a limelight. We show how data efficiency issue can be handled by multi-task transfer learning with pre-trained language models in chapter 3.

Last but not least, **evaluation** of a conversational system is challenging [6]. In offline evaluation, where conversation is evaluated by test collections, in accordance with the Cranfield paradigm, the system can only be evaluated with predefined response candidates. Moreover, it can either judge a single turn or a single path of the conversation, without any reflection of multi-turn history or the other branches of the responses that could have been made by the system during the dialogue [4]. In online evaluation, the dialogue system has to be deployed and accessible to real users. However, this is highly costly and difficult for individual researchers to do so. Chapter 3 and 4 demonstrate how offline and online evaluation can be performed.

Chapter 3

Multi-Task User Simulator

3.1 Introduction

Two recent papers have influenced our work. The first work includes user satisfaction analysis, and the second work includes user utterance generation. In the former, Sun et al. [1] investigated how users' satisfaction influences users' actions during the dialogue with an agent. They noticed that the system's unprofessional responses or failure to catch users' requirements can result in users' dissatisfaction. In the latter, Ben-David et al. [25] recommended the use of the T5 model [9] and used auxiliary tasks like utterance reordering and utterance generation, as an approach to improve the performance of the users' intents prediction task which are proven to be effective. Following this study, we hypothesize that the users' utterance generation task can give a positive transfer to the users' satisfaction scores and actions prediction tasks when trained together.

In this chapter, we make the following contributions:

- We develop a neural architecture and train it to predict both satisfaction scores and actions at the same time in a multi-task learning (MTL) setting. We abbreviate this model as ***SatAct***. By doing this, we see an increase in performance for both prediction tasks through positive transfers across the tasks.
- We develop a neural architecture and train it to predict satisfaction scores

and actions, and generate users' utterances at the same time in an MTL setting. We abbreviate this model as ***SatActUtt***.

- We perform an ablation study to further validate our hypothesis.

We use the User Satisfaction Simulation (USS) [1] as our main dataset, because the turn-level satisfaction annotations are made before the users' last utterance, i.e., before-utterance (BU) prediction. We take the HiGRU [48] and BERT [7] as our baseline models for the user satisfaction and action prediction tasks, since the two models performed the best in the USS benchmark as mentioned in section 2.2.3. Among the 5 different sources of datasets in USS, we do not use ReDial and JDDC. We do not use the former because it does not contain the annotation of users' actions. We do not use the latter because this source is in Chinese, whereas this paper we focuses on English. There are a few more differences we make in the preparation of the USS dataset, which will be discussed in Section 3.3.1.

3.2 Multi-Task Learning

In this work, we fine-tune the T5 model [9] in an MTL setting on the USS dataset to make a user simulator that predicts users' satisfaction scores and actions, and generates users' utterances. We hypothesize that the use of the pre-trained T5 model fine-tuned in an MTL setting will allow the transfer of prior and task-specific knowledge across the tasks. Moreover, the simplistic nature of the T5 model allows the training of a regression task for satisfaction scores, classification task for actions, and generation task for users' utterances simultaneously in a fully text-to-text manner without any task-specific layers.

We design three models based on T5 as illustrated in Figure 3.1: the *SatAct*, *SatActUtt* and *Utt* models. These models differ based on the considered learning tasks. In what follows we will indicate the dialogue history with $\mathscr{H}$, the satisfaction score with s, the action with a and the user-utterance with u.

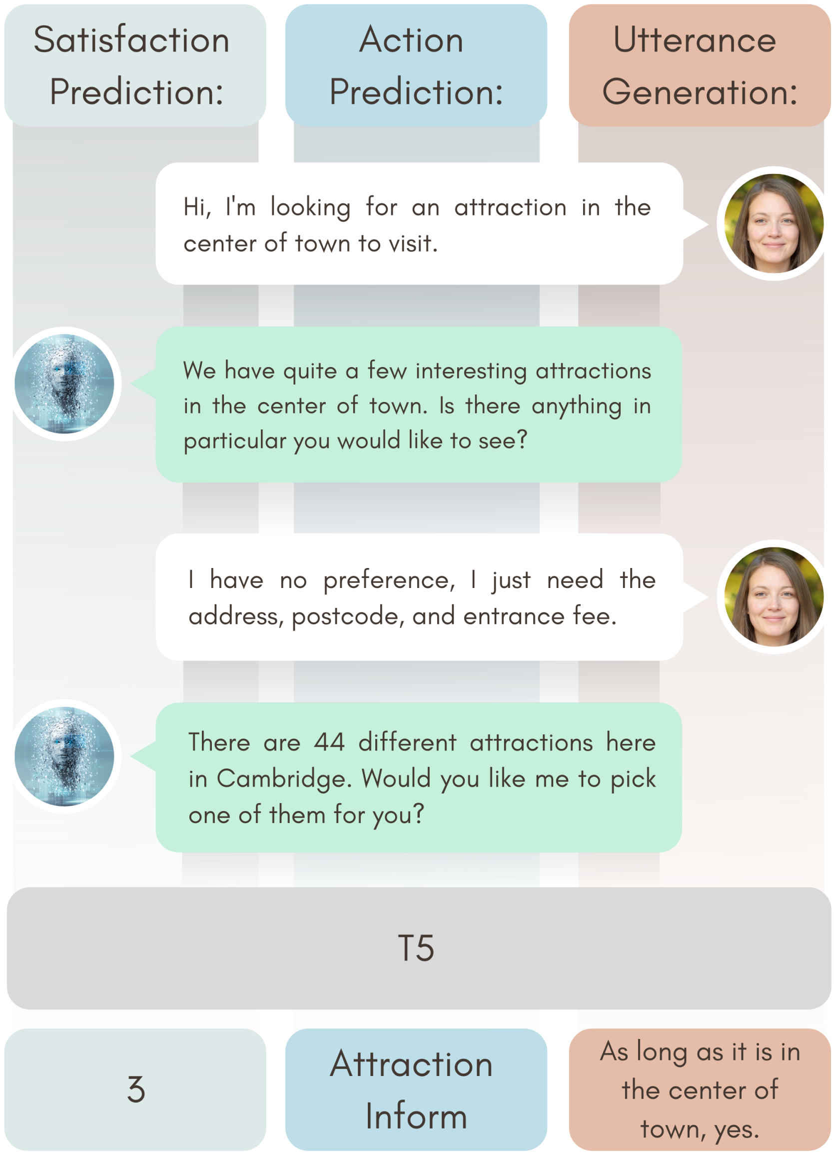

Figure 3.1: Multi-task learning with T5 and task-specific prefixes: "Satisfaction Prediction:", "Action Prediction:", and "Utterance Generation:". These are prepended to the dialogue context. Here we show a MultiWOZ 2.1 sample and the result generated by the fine-tuned T5 model.

3.2.1 SatAct

The goal of *SatAct* is to learn the function $\mathbf{M}(s, a|\mathcal{H})$, where an MTL based generative model $\mathbf{M}$ generates satisfaction scores and actions given a dialogue context $\mathcal{H}$. This model is used to compare the performance of an MTL based T5 model against the models developed by Sun et al. [1].

3.2.2 SatActUtt

The goal of *SatActUtt* is to learn the function $\mathbf{M}(s, a, u|\mathcal{H})$, where an MTL based generative model $\mathbf{M}$ generates users' utterances along with satisfaction scores and actions given the dialogue context $\mathcal{H}$. Should this model be successful it would validate our hypothesis: adding utterance generation as an additional task to the *SatAct* model may provide a positive transfer across the tasks.

3.2.3 Utt

The goal of *Utt* is to learn the function $\mathbf{M}(u|\mathcal{H})$, where a generative model $\mathbf{M}$ generates only a user utterance given the dialogue context $\mathcal{H}$. The purpose of this model is to investigate the impact of satisfaction score and action prediction tasks on the utterance generation task. If the performance of *SatActUtt* is better than *Utt* then the transfer across the tasks is deemed positive.

3.3 Experiments

3.3.1 Dataset Preparation

We are following the general preparation strategy of Sun et al. [1] with the exception of the use of different upsampling factors and the introduction of a new augmentation strategy. The former is used to mitigate a dataset imbalance and the latter to improve learning.

First, we transform the USS dataset into:

$$\mathcal{D} = \{(\mathcal{H}_i, s_i, a_i, u_i)\}_{i=1}^{N} \tag{3.1}$$

where i is the index of a sample, $\mathcal{H}_i$ is the dialogue history, $s_i \in \{1, \ldots, 5\}$ is the user's satisfaction score, a_i is the user's action, and u_i is user's utterance. When training, we provide the models a dialogue history with up to 10 previous turns ({i-9, ..., i}) and let the models learn to predict the next turn $(i+1)$. 10 turns are suitable considering the maximum token length of the T5-base model.

Second, we mitigate a dataset imbalance. As shown in Table 3.1, there is a large discrepancy between the ratios of non-3-rated and 3-rated satisfaction

Table 3.1: Statistics of the dataset with upsampling factors.

	MultiWOZ 2.1	SGD	CCPE
#utterances	12 553	13 833	6 860
#(non-3) / #(3)	0.13	0.20	0.29
upsampling factor	7.5	5.0	3.5

scores. To overcome this issue, Sun et al. [1] over-sampled the cases of non-3-ratings by a factor of 10. This factor was constant across the datasets. In our case, we calculate a specific upsampling factor for each dataset. In this way, we prevent a model from seeing too many non-3-ratings, which can harm the performance of the user simulator.

Third, while upsampling, rather than simply copying the same utterances, we randomly select one augmentation strategy among random deletion, random swap, random insertion, WordNet-based synonym replacement, and back-translation (to and from a random language). We perform this augmentation because it has been proven to be effective in several NLP tasks [66].

3.3.2 Training Details

We split the dataset into train, validation and test sets with a ratio of 8:1:1. When splitting the dataset, we ensured that each satisfaction scores were evenly spread across the splits. When training the model, we used T5-base (220M parameters, vocabulary size: 32 128) [9], and train up to 7 epochs with the following hyperparameters: *batch size* = 4, *Optimizer* = AdamW, *learning rate* = 1e-3. *max_length* is set to 10 when training a *SatAct*, while set to 100 when training *SatActUtt*. The loss function is defined as the negative log-likelihood.

Throughout the training phase, we evaluated an updated model with the validation set and saved the best model based on the validation loss. We used early stopping and linear learning rate scheduling. During the token generation phase, we set the *beam_size* to 5, the *top_p* to 0.95, and the *repetition_penalty* to 2.0. The training took around 6 hours for each model with an NVIDIA Tesla V100 GPU.

Table 3.2: Performance for the User Satisfaction Score Prediction.

	MultiWOZ 2.1				SGD				CCPE			
	UAR	Kappa	Rho	F1	UAR	Kappa	Rho	F1	UAR	Kappa	Rho	F1
HiGRU (Sun *et. al.*)	0.225	0.143	**0.886**	0.238	0.293	0.118	0.451	0.086	0.237	0.167	0.881	0.274
BERT (Sun *et. al.*)	0.256	0.133	0.823	0.224	0.261	0.094	0.477	0.048	0.232	0.147	**0.891**	0.245
SatAct	0.535	**0.824**	0.873	**0.901**	0.449	0.619	0.681	0.713	0.222	0.094	0.347	0.165
SatActUtt	**0.572**	0.767	0.815	0.838	**0.608**	**0.763**	**0.822**	**0.847**	**0.437**	**0.612**	0.690	**0.734**

Table 3.3: Performance for the User Action Prediction.

	MultiWOZ 2.1				SGD				CCPE			
	Acc	Prec	Recall	F1	Acc	Prec	Recall	F1	Acc	Prec	Recall	F1
HiGRU (Sun *et.al.*)	0.518	0.216	0.162	0.167	0.643	0.534	0.505	0.507	0.672	0.503	0.472	0.482
BERT (Sun *et.al.*)	0.519	0.255	0.183	0.191	0.661	0.570	0.572	**0.560**	**0.674**	**0.696**	**0.495**	**0.496**
SatAct	0.616	0.295	0.309	0.299	0.630	0.433	0.471	0.447	0.544	0.267	0.235	0.207
SatActUtt	**0.621**	**0.439**	**0.407**	**0.402**	**0.678**	**0.538**	**0.574**	0.550	0.623	0.458	0.429	0.430

Table 3.4: Performance for the User Utterance Generation.

		BLEU-1	BLEU-4	ROUGE-1-F	ROUGE-2-F	ROUGE-L-F	STS
MultiWOZ 2.1	*SatActUtt*	0.206	**0.019**	**0.193**	**0.065**	**0.182**	**0.327**
	Utt	**0.210**	0.018	0.192	0.063	0.180	0.318
SGD	*SatActUtt*	**0.281**	**0.050**	**0.269**	**0.125**	**0.263**	**0.403**
	Utt	0.267	0.048	0.251	0.117	0.247	0.394
CCPE	*SatActUtt*	**0.195**	**0.019**	**0.240**	**0.072**	**0.230**	**0.410**
	Utt	0.170	0.012	0.222	0.064	0.208	0.394

Table 3.5: Cross-domain UAR for the User Satisfaction Score Prediction.

Trained	Generate	MultiWOZ 2.1	SGD	CCPE
MultiWOZ 2.1	BERT	–	0.233	0.226
	SatActUtt	–	0.247	**0.275**
SGD	BERT	0.249	–	0.223
	SatActUtt	**0.280**	–	0.233
CCPE	BERT	0.213	0.216	–
	SatActUtt	0.266	**0.264**	–

3.3.3 Evaluation Measures

For the evaluation of the models on the satisfaction score and action prediction tasks, we follow the previous work [1]. For the satisfaction score prediction, we use: Unweighted Average Recall (UAR), Cohen's Kappa, Spearman's Rho, and binary-F1-score (positive when satisfaction score > 2). For the action prediction, we use: Accuracy, Precision, Recall, and F1-score. We do not use the accuracy measure for the satisfaction score prediction task due to the imbalance of the labels. For the utterance generation task, we use BLEU [28] and ROUGE [29] scores. For BLEU, we use the cumulative 1-gram (BLEU-1) and 4-gram (BLEU-4). For ROUGE, we use the ROUGE-1-F, ROUGE-2-F, and ROUGE-L-F. Also, we use the Semantic Textual Similarity (STS) score to evaluate the contextual similarity between the ground truth sentence and the generated user utterance. This aligns with the concept of average embedding approach, introduced in section 2.1.4. STS is defined as the cosine similarity between the sentence level embedding vector $\mathbf{e_1}$ from the ground truth utterance and embedding vector $\mathbf{e_2}$ from the model-generated utterance. The average of this similarity across the whole dataset is used for evaluating the performance of models trained on the user utterance generation task. The embeddings are taken from the pre-trained *SRoBERTa-STSb-large* model [67].

3.4 Results

In Table 3.2, we show the performance of the models in predicting user satisfaction scores. In Table 3.3, we show the performance of the models in predicting user actions. In Table 3.4, we show the performance of the models in generating the user's utterances. Finally, in Table 3.5 we show the generalization ability of the models on satisfaction score prediction by testing them on different datasets.

First, as it is shown in Tables 3.2 and 3.3, MTL models achieve state-of-the-art performance in most of the metrics in both satisfaction score and action predictions by a large margin. Second, *SatActUtt* model is better than *SatAct* model in most cases. This means that utterance generation task has given a positive transfer to satisfaction and action prediction tasks. Third, in Table 3.4, *SatActUtt* model always beats the *Utt* model with the exception of one case. This means that the training on the satisfaction score and action prediction tasks also gives a positive effect on the training on the user utterance generation task. These results show that all three tasks can help each other to better simulate users. Moreover, in Table 3.5, the T5 model always shows better generalization ability than BERT in a cross-domain user satisfaction prediction task. This is most likely due to the larger number of parameters of the T5 model and the use of a bigger corpus used to pre-train it. Another noticeable result is that the T5 model does not work very well in CCPE. We believe that it is due to the small number of training samples in the dataset that hindered the model from being well fine-tuned on the CCPE domain.

3.5 Discussions

As can be seen in Table 3.4, the BLEU, ROUGE, and STS metrics are used to evaluate the performance of the models on the user utterance generation task. Although the STS focuses more on the semantic structure of the sentence rather than on their syntactic structure unlike BLEU and ROUGE, simply averaging the scores of the STS seems not complex enough to reflect the erratic nature of user-side utterances. Given the diverse nature of dialogue, utterance generation task might have been better to be additionally evaluated by human evaluators. This aligns with a study conducted by Liu et al. [68] that showed the scores from the automatic evaluation metrics commonly used in natural language generation tasks tend to have little correlation with the human evaluation of the generated dialogue response. However, the finding that the joint learning with satisfaction and action prediction improves utterance generation remains intact.

One critical limitation of the proposed simulator is that it lacks the ability of modeling users' knowledge and mental status. For instance, for the ground-truth utterance: *"Yes, book the tickets, also I want places to go in town."*, our simulator generates: *"Yes, please book it for me."* However, the second part of the ground-truth sentence (*"also I want places to go in town."*) is very challenging to predict as it is a new topic that the user brought up. This is where a personal knowledge graph [4] or a memory augmented neural model [69] can come into play to incorporate the users' mental status or preferences into the simulator. Additionally, this reinforces the need for the design of better evaluation metrics for user utterance generation which are able to capture these subtleties.

3.6 Summary

In this paper, we have shown that the T5 model achieves state-of-the-art performance in predicting user satisfaction scores and actions in the USS dataset with cross-domain generalization ability. This is the first work that combines the user-utterance generation task with the user satisfaction score and action prediction tasks. Moreover, in our analysis, we proved that satisfaction score and action prediction, and utterance generation tasks give a positive transfer to each other when trained in an MTL setting.

Chapter 4

Task-oriented Dialogue System

4.1 Introduction

COoking-a**N**d-**DI**y-**TA**sk-based task-oriented dialogue system, *Condita*, which means 'seasoned' in italian, is a multi-modal task-based dialogue system designed for the 2021 Alexa Prize TaskBot Competition to assist users with tasks related to cooking and Do-It-Yourself (DIY) tasks. Our goal in building this TaskBot was to generate engaging dialogue and provide easy-to-understand instructions in order to make a memorable user experience.

Condita uses a state machine like architecture in order to keep track of the state of the conversation. This architecture lends itself particularly well to task based dialogue systems because each step in a set of instructions can be mapped to its corresponding state in the state machine. After mapping a user utterance to a particular intent, the state machine transitions to the next state. After each state transition, the system will update the user with the next set of possible utterances.

Condita also takes advantage of the multi-modal experience of Alexa devices with screens. Using the Alexa Presentation Language (APL) API, our team heavily customized the user interface to improve aesthetics and usability. Additionally, many Alexa users communicated with Condita using a headless device without any screen. To account for the absence of visuals to convey extra information, we edited response prompts in headless mode to return more detailed text. Condita achieved admirable scores, particularly towards the end of the competition. In this chapter, we analyze the implementation of Condita as well as areas of improvement for future work.

4.2 System Requirements

As this work was for the challenge, there were several requirements mandated by the Alexa Prize team. The requirements were mostly motivated by keeping the anonymity of the creator of the TaskBot (for fair competition), and the mental and physical safety of the user (see Appendix B.1 for the list of system requirements).

Moreover, the system is responsible for safeguarding the customer experience from inappropriate responses. They must detect user utterances that contain content violation keywords (see Appendix B.2 for selected examples) to decline inappropriate tasks that could cause harm to users or their property. For example, ‘constructing a wall’ is deemed as a dangerous task that only skilled workers can do, so the system must be able to decline the task with redirection prompts. Additionally, the system must not give users legal, financial, and medical advice.

Requirement on the development side is the use of Alexa Skills Kit (ASK). The system has to be published as an Alexa Skill. ‘Skill’ is an application for Alexa, which provides a new channel for various content and services. Users of Alexa can explore many Skills published by other developers in the Alexa Skills Store to achieve their daily tasks, such as enjoying music and listening to the news [70]. In our case, the Skill has to be able to meet users’ cooking and DIY tasks. To make development process easier, Amazon provides a self-service APIs, called Alexa Skills Kit (ASK). By using the ASK, developers not only can easily get a transcribed version of the spoken utterances, but also can they define the intents and slot values for NLU. Then the APIs in ASK can send all the information retrieved from the users’ utterance to the connected AWS Lambda endpoint, where the other part of the dialogue system resides.

The Skill must pass a certification process before it can be published live to the real customers in the Alexa Skills store. During the certification process, the system is scrutinized to check its satisfiability of the requirements.

4.3 System Architecture

After the series of Alexa Prize SocialBot challenge, Khatri et al. [71] have released a CoBot, a Python framework that facilitates the development process of conversational systems. It provides a CoBot Core Python SDK, which includes commonly used NLU modules like noun phrase extraction, coreference extraction, and sentiment classification. Furthermore, in an engineering perspective, 1) it provides excellent debugging, testing and logging functionalities which can be difficult on a cloud environment, 2) the CoBot command-line interface (CLI) makes the deployment of the system on ASK and AWS simple.

As can be seen in Figure 4.1, Condita is designed on top of the CoBot framework. Its architecture was devised to reflect all the system requirements mentioned above, and each turn consists of the following steps:

1. The user either speaks a spoken utterance, which is then transcribed by Alexa's Speech Recognition (ASR) service, or uses a touch input.

2. The user's input is then sent to an AWS Lambda endpoint. Because Lambda is a stateless function, all information of the bot's state is held within AWS DynamoDB.

3. The input is ingested into the natural language processing and understanding pipeline (see section 4.4) to produce NLP and task related annotations.

4. The state machine will then select several possible response generators to run based on the user's input and the current state of the bot (see 4.6.1)

5. The chosen response generators all run in parallel and return their generated output. Because running response generators can affect the information stored in the state model, any updates to state are first stored in temporary copies of the state model.

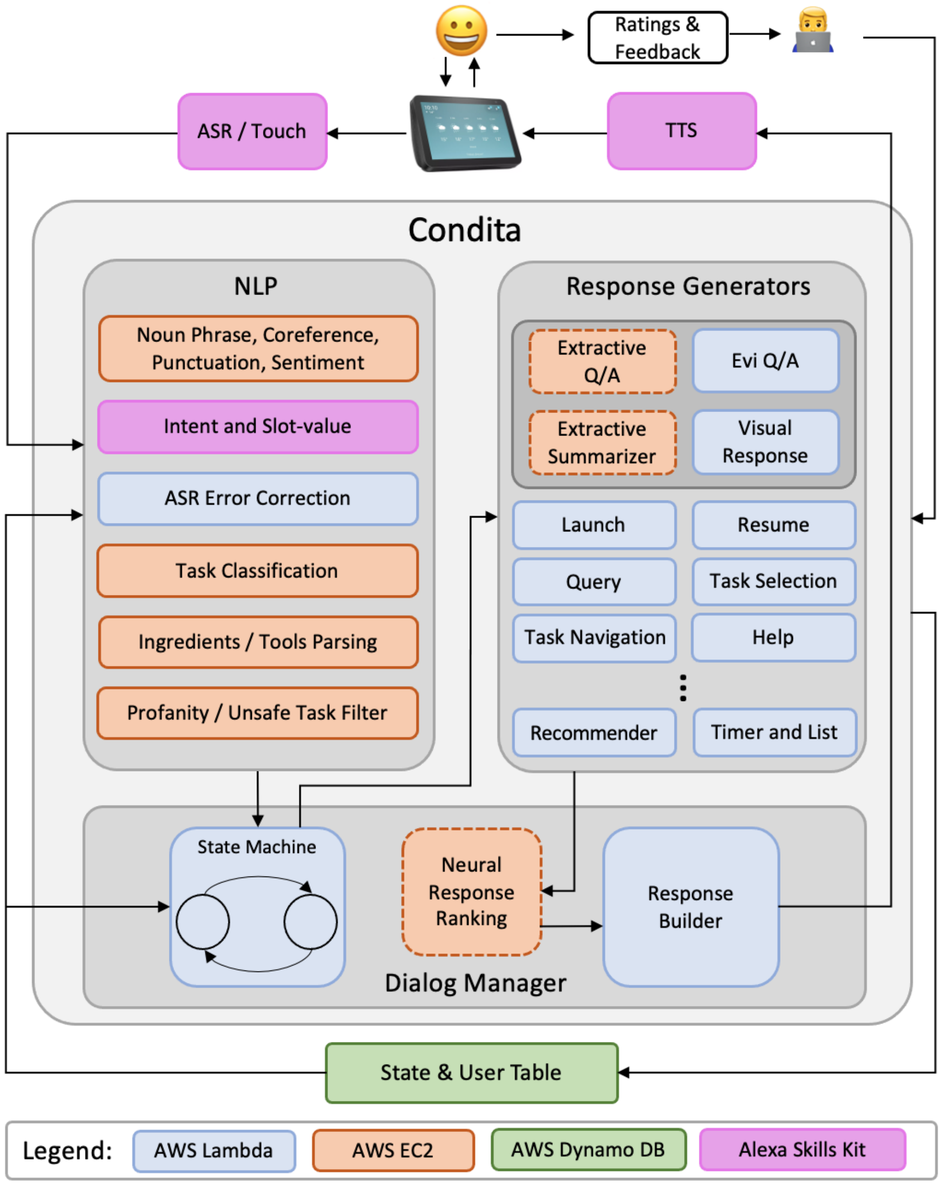

Figure 4.1: System Architecture of Condita. Best viewed in color. 1) Not all response generators are depicted in the Figure. 2) Extractive Q/A and Summarizer, Evi Q/A, and Visual response generators are modules that interact with each response generator. 3) Modules with dotted box are experimented within the system but not deployed in the production. 4) Noun Phrase, Coreference, Punctuation, and Sentiment modules were provided by the CoBot framework. 5) After every conversation, user can rate the session on a scale of 1 to 5 with textual feedback. User ratings and feedback pipeline is constructed for continuous development. GitHub Workflows streamlines the fetching of ratings from Alexa Prize S3 bucket, followed by a table joining with state table. The joined table is used not only for continuous development but also for creating a response ranking dataset.

6. The neural response ranker then ranks all responses generated and selects the best response (see section 4.6.2). The bot then updates any information changed including the state based on the selected response and discards all other temporary information. The state and user tables in AWS DynamoDB are updated accordingly.

7. The response builder then post-processes the output text and sends the bot utterance to Alexa's Text To Speech (TTS) Service to say the bot's response to the user. Additionally, if the user has a screen device, the response builder will also return a visual response built on top of the Alexa Presentation Language (APL).

4.4 Natural Language Processing

4.4.1 Initial NLP Modules

The *Neural Coreference* module extracts the coreference clusters from the previous and current turn. It returns the pronouns referenced along with the noun it points to. As the name suggests, the *Nounphrase Extraction* module provides all the nounphrases detected from the current user utterance. Along with these modules, information about *punctuation* and *sentiment* from the users' utterance are stored in the state table. *Nounphrase Extraction* is heavily used when filling the slot values that were missed from the Alexa Skills Kit (See section 4.4.2). Low sentiment level of the utterance is used for triggering a sensitive responder. These modules are also used when correcting ASR errors.

4.4.2 Intent Classification and Slot Filling

Alexa Skills Kit (ASK) is used to define necessary intents and slots for understanding users' utterance. Custom defined intents are: TaskRequestIntent (DIYTask, RecipeTask, Food, and Drink as slots) TimerManagementIntent, ListManagementIntent, CompleteIntent, UserNameIntent, DietaryPreferencesIntent, SkipIntent, RecommendationsIntent. Other than these, Amazon built-in intents are used, such as Cancel, Stop, Resume, Yes, No, Select, Next, Previous, Help, and

Fallback intent. See Appendix B.3 to check how intent and slots are defined in the console, and their results (quick-test) shown from ASK console.

Since ASK does not guarantee robust performance of intent classification and slot filling, *Task Classification* layer (section 4.4.3) and *Nounphrase Extraction* module are used to complement the performance of the ASK.

4.4.3 Task Classification

Understanding the topic of users' question is important for a seamless conversation in task oriented conversation. Therefore, we introduce an extra *Task Classification* layer within the NLP pipeline.

For model training, we have collected questions related to Cooking [1] and DIY [2] from `https://stackexchange.com/questions`. 24,954 and 50,000 task-specific questions are collected for cooking and DIY, respectively. The corpus is vectorized based on the frequency of the tokens, then fed into the multinomial Naive Bayes text classifier. It showed 98% test accuracy, with high F1-scores (0.97 in cooking task, 0.99 in DIY task). After the deployment to the EC2 instance, authors have not seen any failing cases from this model.

4.4.4 ASR Error correction

After the utterances go through the previous NLP-steps, substantial amount of information is stored in a state table. This information can not only be utilized during response generation but also be referred to when attempting ASR correction. ASR correction is done in a rule-based continuous-development fashion. User Feedback pipeline (see Figure 4.1) can alert developers to look into conversational logs if low ratings are given by users continuously. Once ASR error is detected by developers from the logs, rule-based logic can be added to the ASR Error Correction module within the NLP pipeline. The logic reads to the previously stored NLP information from the state table and makes a correction. For example, if user is in a TASK_SELECTION State and said "I want option tree", it is highly likely that ASR processor made error in catching a word "three".

[1] `https://cooking.stackexchange.com/questions/`
[2] `https://diy.stackexchange.com/questions/`

4.4.5 Avoiding dangerous and sensitive conversation

In accordance with the Alexa Prize guidelines, any conversation that is considered dangerous must be stopped immediately. Dangerous tasks are collected through Amazon Mechanical Turk [3] crowd-sourcing. Paid workers were given both dangerous and safe tasks and told to decide if they would allow their kids to do the task or not. This allowed us to have a dataset with binary labels (safe and unsafe). Additionally, if the user asks a sensitive question, the bot must say that they cannot answer this question and offer to continue working on the current task. Examples of sensitive topics include legal, financial, and medical domains. To classify an utterance as dangerous or sensitive, we use regular expressions to check for banned words from a list of violation keywords provided by the Alexa Prize team. If the conversation is deemed safe and non-sensitive, the utterance is then passed along to the dialogue management module.

4.5 Response Generations

4.5.1 Stateful Responders

Stateful responders change the information stored in the state table. Refer to section 4.6.1 for a description of each stateful responder.

4.5.2 Stateless responders

Additionally, there are stateless responders that do not make any state changes when called. If the system selects one of these responders, the system remains in its current state for the next turn in the conversation.

- **Help Responder:** Sometimes, users are unsure what the system is expecting as input. In order to alleviate this confusion, the help responder returns example utterances that the user can say. At any point in the conversation, users can say, "help" to hear available commands.

- **Evi Open domain QA Responder:** While working on a task, users may have related questions. For example, a user might ask "How many teaspoons are

[3] `https://www.mturk.com/worker`

in a tablespoon?" To answer these questions, we implemented the Evi QA API into Condita to answer these questions. Additionally, if Evi is unable to return an answer, it will return a fallback prompt saying that it is unable to respond to the user's utterance.

- **Dietary Preferences:** Users can set their dietary preferences as described in 4.5.3.2.

- **Sensitive Responder:** The sensitive responder is run for every utterance to check if an utterance is classified as sensitive. If the utterance is sensitive, the bot will inform the user that it cannot talk about that particular subject and asks the user if they would like to continue with the current task.

- **List Responder:** Users can create shopping lists on their Alexa app or add/remove items from existing shopping lists.

- **Timer Responder:** Timers can be useful when cooking or working on a DIY task. Users can ask Alexa to set up timers while they are completing a task.

4.5.3 Sub-modules for Response Generators

4.5.3.1 Extractive QA and Summarizer

Extractive QA was developed to handle questions that are asked during the steps of a task. Although we have Evi Question Answering API, it is not useful when the answer should be found within a specific passage. For example, Extractive QA module can be used when user asks how many/much a certain ingredient is needed for a recipe given a recipe information. RoBERTa-Base model [72] pre-trained on SQuAD dataset [73] is used and deployed as one of the remote service on EC2.

Extractive Summarizer is used to prevent from providing users with lengthy task description. Lengthy task information are usually retrieved in DIY task. Once this description is outputted to the users without any summarization or truncation, users tended to give low ratings after the conversation. Therefore, within a response generator, we had extractive summarization model to first summarize the description before passing it to the response builder. Pre-trained Distilled-BERT [74] model is used for faster computation.

4.5.3.2 Filter by dietary preference

To accommodate for users with dietary preferences, the dietary preferences filter was designed to allow users to exclude certain ingredients when searching for recipes. For example, users can say, "add peanuts to my dietary preferences" to filter out any recipes that list peanuts as an ingredient. This user-related information can be saved in the user table.

In order to advertise this feature to users, the dietary preference feature was added in the "Help" prompt. Whenever the bot fails to understand a user's request, the bot informs the user that they can say "help" to hear additional options. If the user responds with "help", the system replies with instructions on how to use the dietary preferences feature.

4.5.3.3 Fun Facts

Many TaskBots can feel rigid to interact with because of the cut-and-dry responses. To make the bot feel more human-like and interesting to interact with, we manually scraped fun facts from cooking websites such as `https://facts.net/cooking-facts/`. All facts were manually scraped to prevent any inappropriate facts from being added. The collection of fun facts were stored in a key-value database with the food name as a key and fun fact as the value. When a user asks for a recipe, the fun fact database is queried for the user's requested recipe and appends the relevant fact to the systems output. If the user's recipe is not found in the fun fact database, a general fun fact about cooking is returned instead.

4.6 Dialogue Management

4.6.1 State Machine

Users can converse with an Alexa Prize Taskbot by saying, "assist me". They are then routed to one of several taskbots in the competition. Condita users select a

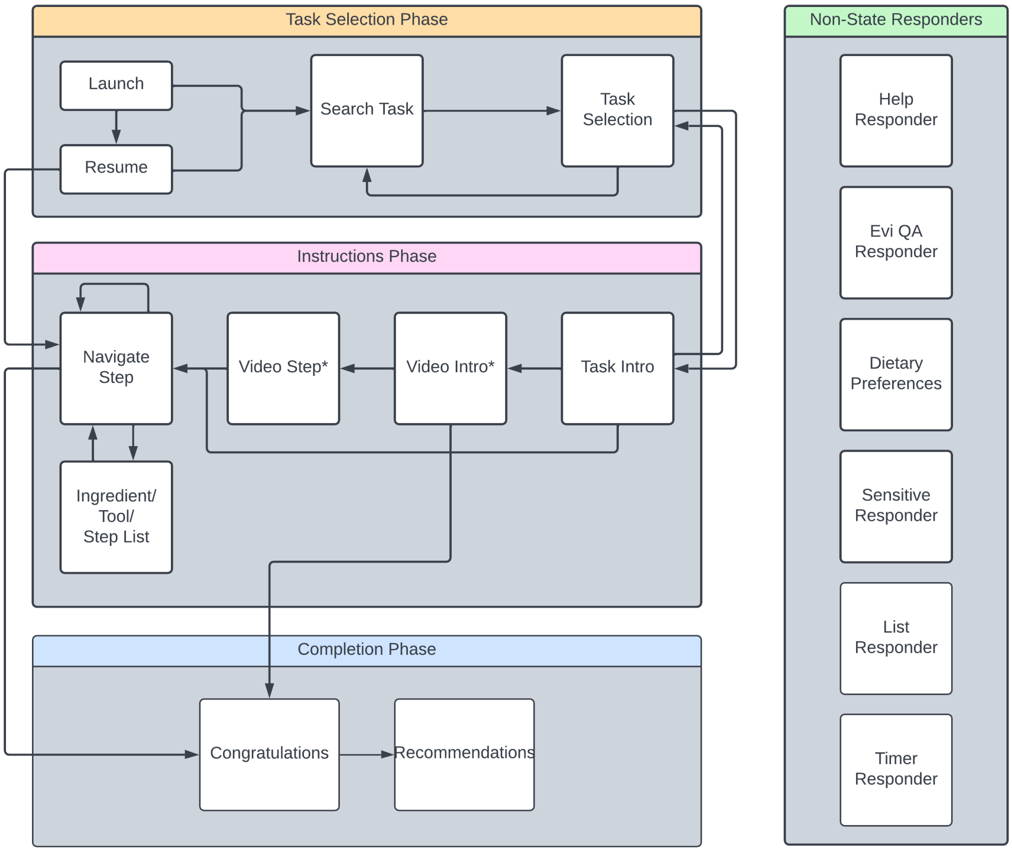

Figure 4.2: State machine like architecture of Condita's DM. The user begins at the Launch State. Steps with an asterisk (*) can only be reached if the user is using a screen device.

task to work on by walking through the Task Selection Phase. Users start at the *Launch State* and transition to the next states as described below:

- **Launch:** On start, the bot will check if whether the user is still working on a task from a previous conversation. If they have a task in progress, the bot will move to the Resume state. Otherwise, the bot will move to the Search Task State.
- **Resume:** The bot will ask users if they would like to continue working on their task. If the user responds yes, they will move to the Navigate Step state and will continue working on the step they were last working on. If the user responds no or asks for a new task, they will move to the Search Task state.
- **Search Task:** After the user requests for results for a certain task, the bot

will call the WikiHow and Whole Foods Market APIs to search for and return relevant results.

- **Task Selection:** Users can select tasks by either saying the title of the task, uttering a phrase with an ordinal (e.g. the third one), or by touching the corresponding image if they have a screen device. Additionally, the can say, "show me more" to view additional tasks or "go back" to hear the previous tasks again.

After selecting a task, the user enters the Instructions Phase. The Instructions Phase consists of the following states:

- **Task Intro:** Before starting the task, there is an introductory state where the bot explains several key metrics of the task, such as user ratings, estimated time of completion, ingredients needed, etc. Users can also choose to go back to the task selection state if they wish to select another task.

- **Video Intro (screen device only):** Several WikiHow articles include videos on how to perform the task. The bot will ask the user if they would like to view the video if there is a video available. If the user responds yes, the bot will then begin playing the video. If the user says no, the state then moves to the Task Intro step. If there is no screen present, the bot will skip to the navigate steps state.

- **Video Step (screen device only):** Using the APL Video Template, Condita plays the video. Users can control the video either through touch input or through voice. At the end of the video or at user request, the bot asks if they would like to continue to the step by step instructions or if they are finished with the task. The user will then be routed to the Task Intro state and Congratulations step, respectively.

- **Navigate Step:** In the Navigate Step state, the bot instructs the user about the current step of the task. Users can say "next" or "previous" to see the next or previous state respectively.

- **Ingredient List:** The Ingredient List state lists all of the ingredients in the recipe. If the user has a screen device, the system will return a list view of the ingredient list. Note that this step is only usable if the task is classified as a cooking recipe.

- **Tools List:** The Tools List state lists all of the tools in the task. For example, it may list the cooking utensils needed in a recipe or the equipment needed for a DIY task. If the user has a screen device, the system will return a list view of the tools list.

- **Step List:** The Step List state shows all of the steps in list view form if the user has a screen device. Otherwise, it reads out every step in the task.

Once a user finishes a task, they move on to the Completion Phase. The Completion Phase consists of the following two states:

- **Congratulations:** Once the user completes the final step, the user is congratulated for finishing the task.

- **Recommendations:** After the user is congratulated for finishing the task, the bot recommends two tasks that the user can do for next time. Due to the rules of Alexa Prize, the user cannot start a new task after task completion. Thus, the bot informs the user that they can start a new conversation with an Alexa Prize TaskBot and ask about this task.

Detected intents/slots, selected responders, and state changes are annotated with the conversations in Appendix B.6. The example conversations were generated by interacting with Condita.

4.6.2 Neural Response Selection

This section (4.6.2) contains the work co-developed with one of the team members of the competition. To achieve a better performance in response selection, ranking strategy has been developed based on a neural response ranker, BERT-FP

[75]. BERT-FP focuses on a short conversation context and achieves the state-of-the-art results in many datasets. We have fine-tuned this model with two datasets and obtained reasonably good performance.

4.6.2.1 Dataset

Stack Exchange is a popular web-forum for general questions. Each question has a thread of answers and comments. Here we take the question and the its comments as the context and answers as the targets. Besides the targets, we randomly pick the other two unrelated documents as the wrong answers. This process was done both in cooking-related and DIY-related questions from *stack-exchange.com*. 96,691 and 189,477 questions are collected for cooking and DIY, respectively. For each target, we generated five wrong responses. These 286,168 cases were then divided into train, test, and validation set by 3:1:1 ratio.

Real user data is also used to fine-tune the model as the TaskBot has been deployed. With the conversations between our bot and users, we generated a number of cases in this dataset. We used the same ratio when splitting the dataset for training.

4.6.2.2 Performance

Table4.1 shows that BERT-FP has a strong potential to handle the response ranking task in cooking and DIY domain on production.

Dataset	Stage	MAP	MRR	P@1	R@1	R@2	R@5
Stack Exchange data	test	0.9935	0.9935	0.9871	0.9017	1	1
Stack Exchange data	validate	0.9766	0.9908	0.9846	0.4193	0.7042	0.9816
Real user data	test	0.9767	0.9767	0.9534	0.9534	1	1
Real user data	validate	0.9496	0.9688	0.9483	0.6336	0.9267	0.9784

Table 4.1: Performance of BERT-FP model on Stack Exchange data and the real user data. Mean Average Precision, Mean Reciprocal Rank, Precision at one, and Recall are denoted as MAP, MRR, P@1, and R@k, respectively.

4.7 Multimodal Customer Experience

This section (4.7) contains the work co-developed with one of the team members of the competition. One of the main features of Condita is the multi-modal user experience for users with an Alexa screen device. Using the Alexa Presentation Language (APL), we developed an easy-to-use interface to enhance the user experience. By showing information visually, we reduce the verbosity of prompts. This resulted in higher user satisfaction and longer user engagement. Figure 4.3 shows a few examples of Condita's visual responses.

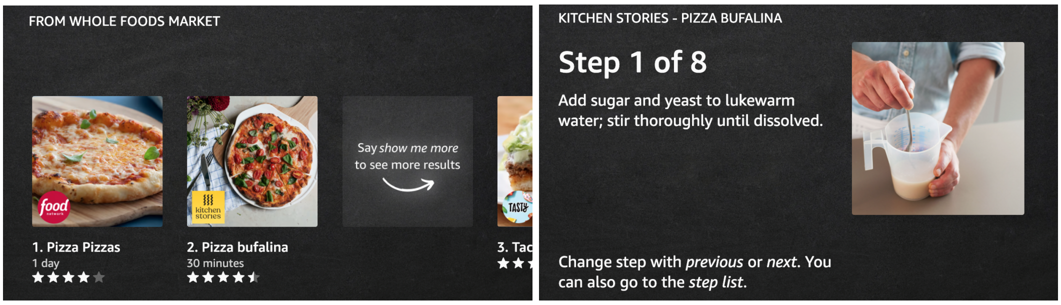

Figure 4.3: UI for the Task Selection (left) and Navigate Step (right) states.

Due to the absence of a screen in headless mode, information that would normally be shown visually must be communicated verbally. For example, a recipe's rating and estimated time of completion can easily be seen on the screen device. However, users would not know this information if they are unable to see it on the screen. To alleviate this issue, we added additional information in the responses of headless prompts. For example, in the bot tells users the task's rating and estimated time of completion during the task intro state in headless mode.

4.8 Deployment

Figure 4.4 shows the overall system deployment pipeline via multiple AWS services, and how the Echo device communicates with the deployed service.

Developers prepare micro-services in the source code. Modules like *Task Classification*, *Ingredients/Tools Parser*, *Nounphrase Extraction*, and *Neural Coreference* are to be containerized using Docker images and hosted on Amazon ECR (Elastic Container Registry), which provides a highly available and scalable architecture. This is managed by the ECS (Elastic Container Service). Once developers push the source code via the Git version control service, AWS CodePipeline

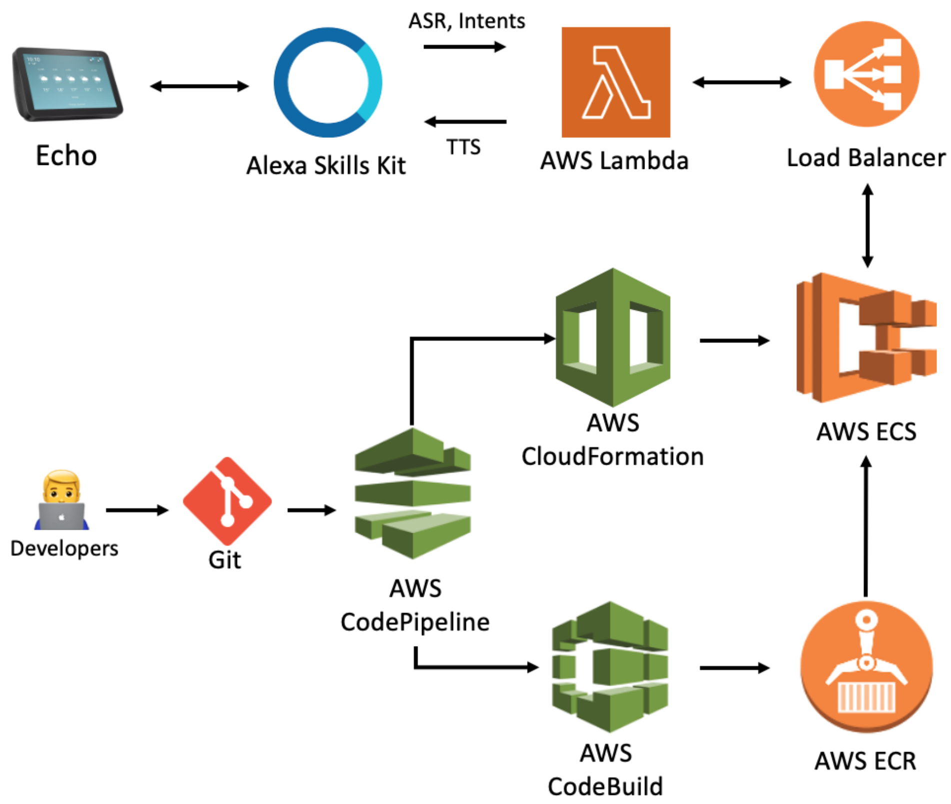

Figure 4.4: Condita depolyment process

is triggered for the CI/CD purpose. The CodePipeline 1) monitors the changes of the source code 2) build Docker images from the source 3) pushes the images to the ECR, and 4) updates the ECS with the new images via the CloudFormation. The source code is also updated in the serverless compute service, Lambda, which interacts with the ASK and ECS in an event-driven way.

4.9 Debugging

Debugging a software that runs on a cloud requires a reliable logging functionality. We have used AWS CloudWatch for the logging service, as it lets developers divide log groups for better debugging and set alarms for anomalous behaviors of the system. Also, the CoBot [71] has its own CLI, *cobot transcribe -t <state-table-name> -s [session-id]*, that shows an annotated conversation given the session ID of the conversation which facilitates debugging process (see Appendix B.4).

4.10 Testing

Unit Testing. After each intent and slot are defined through the ASK console, we could test the classification performance by the quick-test functionality as can be seen in Appendix B.3. This allowed a single-turn NLU testing. In order to unit test the containerized remote modules, Docker's CLI was used to locally build a docker image and run the container. Then, the remote modules can be tested by sending a *curl* command to the local docker containers. This method can be slow, but it keeps the dependencies isolated.

Integration Testing. By running the dialogue system in a local environment, interactive chatting in the command line was possible. This way, multi-turn integration testing could be executed after the batch of unit testing (see Appendix B.5).

Beta Testing. By the AWS CodePipeline, the deployed service will first stay in the beta-level. After the internal beta testing, the beta version can be pushed to the production-level once approved by one of the member of the team. Beta testing can be done in two ways. First way is to use the ASK developer console. In the console, the developer can check the input and output in JSON format in each turn of the conversation. Also, by speaking into the microphone, developers can also check the n-best ASR result in the JSON formatted log. Another way of beta testing is to actually speak with the Alexa device. That way, we could test the service in the same environment that users experience.

4.11 Online Evaluation

After every interaction session is ended, user could rate our dialogue system on a scale from 1 to 5. A CSV file that contains user ratings for each session was daily updated in our AWS S3 bucket by the Alexa Prize team. To actively utilize the benefits of the online evaluation, we have introduced a feedback loop into the continuous development pipeline. By using AWS Athena, an interactive query service that uses S3 folders as its data source, we 1) fetched the conversation history from the State Table, 2) merged the fetched history with the user satisfaction score, and 3) saved the histories in a private repository sorted by the satisfaction score for easy navigation. Moreover, this feedback loop was automated to update everyday by the cron job provided by the GitHub Workflows. This let developers quickly identify the part that users express their dissatisfaction, and fix in accordance with their feedback. See Code List B.2 and B.3 for more details.

4.12 Summary

In this work, we have described the architecture design for Condita, a state-machine inspired task-oriented dialogue system that assists users with cooking or home improvement tasks. We presented how multi-turn dynamics of the TOD is handled by our framework with domain knowledge integration. Moreover, we discussed how the system is developed, tested, deployed, and evaluated robustly. Our focus on improving the multi-modal customer interface and generating engaging dialogue helped create an easy-to-use, memorable user experience to the users.

Part of the source code can be found in the Appendix B.7. Note that the whole source code of Condita cannot be submitted, distributed nor published as it contains the properties of Alexa Prize. Only the subset of the code that do not include Alexa Prize's property and that are written by the author of this thesis are listed in the Appendix.

Chapter 5

Proposing Future Research

5.1 Conversation Look-ahead by User Simulation

Section 2.3.2 has introduced the four big challenges in task-oriented dialogue systems. However, the black box nature is endemic in most of the research that involves neural networks, and this is the same in the dialogue systems research, i.e., the problem of explainability and reasoning of actions.

We propose *Conversation Look-ahead by User Simulation* to mitigate the problem. Although there has been some efforts to create a conversation look-ahead in a few research [25, 76], as far as we know, there has been no attempt to use the look-ahead for response ranking in Dialogue Manager.

User simulator that helps reasoning. After user's utterance goes through the NLU component of the dialogue system, Dialogue Manager (DM) pulls relevant information from the NLU component, and generates one or more candidate responses. In Condita, we proposed the use of a neural response selection model to select the best system response. However, this is a complete black box model, thus does not provide a proper reasoning about the selection. Can we do better? We can use the multi-task user simulator introduced in chapter 3.

To alleviate this problem, we can let the user simulator generate possible user's utterances for each candidate system response. Once the system has collected all the generated user utterances, it can clone each state and generate candidate responses for each of the generated user utterances. This will allow the bot to construct a future conversation tree (conversation look-ahead). Now, the agent needs to have a strategy to select the best branch among the multiple branches in the conversation tree. Here, we can use user satisfaction score (that was given by the user simulator) to evaluate each branch and select the branch that can give higher satisfaction to the user. Once we have obtained the satisfaction score for each branch of the conversation tree, the agent can use Minimax or Monte Carlo search algorithm to choose the most rewarding branch.

Points that can be criticized. Although user simulator can be used to mitigate the problems, it does not perfectly solves them. First, it can improve the explainability, but the fact that the user simulator itself is also a large language model prevents the solution from completely free from the black box characteristic. However, it still improves a lot on the reasoning on an action, as constructing a branches of future conversation tree is a great example of a What-if experiment. Second, in production, creating more than a single turn of look-ahead might not be feasible, as state cloning and searching is computationally heavy and time-consuming. Although the Minimax algorithm can be substituted by Alpha-beta pruning for cheaper computation, it is still open to question whether creating more than a single look-ahead will improve the overall user satisfaction or not.

5.2 Personalized User Simulator

As it was shortly discussed in section 3.5, a user simulator based on a personal knowledge is a possible direction of its improvement. Regarding how the user simulator should be structured, a detailed explanation with a figure can be found from a research suggestion paper written by Balog [4]. User Simulator should mimic users. To do that, the architecture of the simulator can be similar to the conversational agents, but has to have extra modules in order to give the following ability: personal preference, persona, stochastic behaviour, ability to learn the system, and form expectations. For the NLU module, unlike that of the dialogue system, it has to be adapted to consider personal knowledge, because users act differently based on their background knowledge. By using personal knowledge graph in a User Model, we can refine the outcome of the original NLU model within the user simulator.

5.3 Reinforcement Learning on Dialogue Manager

In chapter 4, Condita showed state machine like structure to track the dialogue states and make actions. This can be seen as an RL problem. Condita's states have Markov property, i.e., state at time step t contains all the information necessary for future predictions. Anything that happened prior to time t does not have to be considered to predict future actions, as the current state captures all the necessary information. Therefore, Condita's dialogue manager can be viewed as a Markov Decision Process (MDP) [59]. The multi-task user simulator can give rewards to the system when the system choose an action value based on the current state. Probability can be assigned for each action (transition function in Condita), which is called policy learning. Policy is a mapping of $\pi : S \times A \rightarrow [0,1]$, where S is the set of states and A is set of actions. For every state $s \in S$, policy can assign the probability of taking the action to each $a \in A$, i.e., $\pi(a|s)$ [59]. There will be always one or more deterministic optimal policy for any MDP. In case there are multiple optimal policy, we can either select random response based on stochasticity or pass those multiple results to the neural response selector introduced in section 4.6.2. This MDP problem can be solved by iterations of policy evaluation and policy improvement, i.e., policy iteration (Dynamic Programming).

Chapter 6

Conclusions

In conclusion, we have alleviated all four challenges of task-oriented dialogue system identified in section 2.3.2: 1) multi-turn dynamics, 2) domain knowledge integration, 3) data efficiency, and 4) evaluation.

First, the novel multi-task neural user simulator has mitigated the **data efficiency** and **evaluation** problems. The multi-task learning has let the transfer of prior and task-specific knowledge across the tasks. This allowed the model to learn the data in the most efficient way. Once this simulator is used for evaluation of dialogue systems, the current limitations of the offline evaluation (single-turn or single path evaluation) can be overcome.

Second, the Condita has presented a solution for **multi-turn dialogue tracking** and **domain knowledge integration**. Its dialogue manager's architecture motivated by the state-machine could track dialogue states with the help of multiple transition functions. Based on the state and policy information set by the manager, the system could make appropriate queries to the external knowledge base.

Overall, the thesis has successfully achieved the following aims:

1. To provide an extensive survey of the related work to understand the literature and find a niche in research.

2. To implement a novel multi-task neural user simulator that can mimic users' utterance while predicting the users' satisfaction score and action at the same inference-time.

3. To provide a proof of the hypothesis that the muti-task learning of user satisfaction prediction, action prediction, and utterance generation will give positive transfers to each other.

4. To build a multi modal task-oriented dialogue system specialized in cooking and home improvement task.

5. To rigorously test and deploy the service over the Alexa Platform.

6. To propose future research to tackle the remaining challenges in conversational artificial intelligence.

Bibliography

[1] Weiwei Sun, Shuo Zhang, Krisztian Balog, Zhaochun Ren, Pengjie Ren, Zhumin Chen, and Maarten de Rijke. *Simulating User Satisfaction for the Evaluation of Task-Oriented Dialogue Systems*, page 2499–2506. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380379. URL https://doi.org/10.1145/3404835.3463241.

[2] Minlie Huang, Xiaoyan Zhu, and Jianfeng Gao. Challenges in building intelligent open-domain dialog systems, 2019.

[3] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54(1):755–810, 2020. doi: 10.1007/s10462-020-09866-x. URL https://hal.archives-ouvertes.fr/hal-03006231.

[4] Krisztian Balog. Conversational AI from an information retrieval perspective: Remaining challenges and a case for user simulation. In Omar Alonso, Stefano Marchesin, Marc Najork, and Gianmaria Silvello, editors, *Proceedings of the Second International Conference on Design of Experimental Search & Information REtrieval Systems, Padova, Italy, September 15-18, 2021*, volume 2950 of *CEUR Workshop Proceedings*, pages 80–90. CEUR-WS.org, 2021. URL http://ceur-ws.org/Vol-2950/paper-03.pdf.

[5] Aldo Lipani, Ben Carterette, and Emine Yilmaz. How am i doing?: Evaluating conversational search systems offline. *ACM Trans. Inf. Syst.*, 39(4), aug 2021.

[6] Zheng Zhang, Ryuichi Takanobu, Qi Zhu, MinLie Huang, and XiaoYan Zhu. Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, 63(10):2011–2027, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://aclanthology.org/N19-1423`.

[8] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[9] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[11] Alexa Prize - Amazon Science. `https://www.amazon.science/alexa-prize`, . [Online; accessed 28-April-2022].

[12] Condita - Amazon Science. `https://www.amazon.science/`

alexa-prize/teams/university-college-london-condita, 2021. [Online; accessed 28-April-2022].

[13] Kun Jing and Jungang Xu. A survey on neural network language models. *arXiv preprint arXiv:1906.03591*, 2019.

[14] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[15] Usman Naseem, Imran Razzak, Shah Khalid Khan, and Mukesh Prasad. A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 20(5), jun 2021. ISSN 2375-4699. doi: 10.1145/3434237. URL https://doi.org/10.1145/3434237.

[16] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

[17] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202.

[18] Nikolaos Pappas and Thomas Meyer. A survey on language modeling using neural networks. Technical report, Idiap, 2012.

[19] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

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

[21] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[22] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *arXiv preprint arXiv:2106.04554*, 2021.

[23] Mihir Kale and Abhinav Rastogi. Template guided text generation for task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6505–6520, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.527. URL `https://aclanthology.org/2020.emnlp-main.527`.

[24] Zhaojiang Lin, Andrea Madotto, Genta Indra Winata, and Pascale Fung. MinTL: Minimalist transfer learning for task-oriented dialogue systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3391–3405, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.273. URL `https://aclanthology.org/2020.emnlp-main.273`.

[25] Eyal Ben-David, Boaz Carmeli, and Ateret Anaby-Tavor. Improved goal oriented dialogue via utterance generation and look ahead, 2021.

[26] Ananya B. Sai, Akash Kumar Mohankumar, and Mitesh M. Khapra. A survey of evaluation metrics used for nlg systems. *ACM Comput. Surv.*, 55(2), jan 2022. ISSN 0360-0300. doi: 10.1145/3485766. URL `https://doi.org/10.1145/3485766`.

[27] Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 1977.

[28] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL `https://doi.org/10.3115/1073083.1073135`.

[29] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/W04-1013`.

[30] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021. doi: 10.1109/TKDE.2021.3070203.

[31] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1441. URL `https://aclanthology.org/P19-1441`.

[32] Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. Multi-domain dialog state tracking using recurrent neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 794–799, Beijing, China, July 2015. As-

sociation for Computational Linguistics. doi: 10.3115/v1/P15-2130. URL `https://aclanthology.org/P15-2130`.

[33] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML ’08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390177. URL `https://doi.org/10.1145/1390156.1390177`.

[34] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[35] Ruijian Xu, Chongyang Tao, Daxin Jiang, Xueliang Zhao, Dongyan Zhao, and Rui Yan. Learning an effective context-response matching model with self-supervised tasks for retrieval-based dialogues. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14158–14166, May 2021. URL `https://ojs.aaai.org/index.php/AAAI/article/view/17666`.

[36] Zhenyu Zhang, Tao Guo, and Meng Chen. *DialogueBERT: A Self-Supervised Learning Based Dialogue Pre-Training Encoder*, page 3647–3651. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450384469. URL `https://doi.org/10.1145/3459637.3482085`.

[37] Anish Acharya, Suranjit Adhikari, Sanchit Agarwal, Vincent Auvray, Nehal Belgamwar, Arijit Biswas, Shubhra Chandra, Tagyoung Chung, Maryam Fazel-Zarandi, Raefer Gabriel, et al. Alexa conversations: An extensible data-driven approach for building task-oriented dialogue systems. *arXiv preprint arXiv:2104.09088*, 2021.

[38] Aciel Eshky, Ben Allison, and Mark Steedman. Generative goal-driven user

simulation for dialog management. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 71–81, 2012.

[39] Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. Neural user simulation for corpus-based policy optimisation of spoken dialogue systems. In *Proceedings of the 19th Annual SIGdial Meeting on Discourse and Dialogue*, pages 60–69, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5007. URL https://aclanthology.org/W18-5007.

[40] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. End-to-end task-completion neural dialogue systems. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 733–743, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL https://aclanthology.org/I17-1074.

[41] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2182–2192, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1203. URL https://aclanthology.org/P18-1203.

[42] Hua Ai and Fuliang Weng. User simulation as testing for spoken dialog systems. In *Proceedings of the 9th SIGdial Workshop on Discourse and Dialogue*, pages 164–171, 2008.

[43] Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Lin-*

guistics; Companion Volume, Short Papers, pages 149–152, Rochester, New York, April 2007. Association for Computational Linguistics. URL `https://aclanthology.org/N07-2038`.

[44] Layla El Asri, Jing He, and Kaheer Suleman. A sequence-to-sequence model for user simulation in spoken dialogue systems. *arXiv preprint arXiv:1607.00070*, 2016.

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[46] Paul A Crook and Alex Marin. Sequence to sequence modeling for user simulation in dialog systems. In *INTERSPEECH*, pages 1706–1710, 2017.

[47] Praveen Kumar Bodigutla, Aditya Tiwari, Spyros Matsoukas, Josep Valls-Vargas, and Lazaros Polymenakos. Joint turn and dialogue level user satisfaction estimation on multi-domain conversations. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3897–3909, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.347. URL `https://aclanthology.org/2020.findings-emnlp.347`.

[48] Wenxiang Jiao, Haiqin Yang, Irwin King, and Michael R. Lyu. HiGRU: Hierarchical gated recurrent units for utterance-level emotion recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 397–406, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1037. URL `https://aclanthology.org/N19-1037`.

[49] Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-

Tur. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 422–428, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL `https://aclanthology.org/2020.lrec-1.53`.

[50] Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8689–8696, 2020.

[51] Filip Radlinski, Krisztian Balog, Bill Byrne, and Karthik Krishnamoorthi. Coached conversational preference elicitation: A case study in understanding movie preferences. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*, pages 353–360, Stockholm, Sweden, September 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5941. URL `https://aclanthology.org/W19-5941`.

[52] Meng Chen, Ruixue Liu, Lei Shen, Shaozu Yuan, Jingyan Zhou, Youzheng Wu, Xiaodong He, and Bowen Zhou. The JDDC corpus: A large-scale multi-turn Chinese dialogue dataset for E-commerce customer service. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 459–466, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL `https://aclanthology.org/2020.lrec-1.58`.

[53] Raymond Li, Samira Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. Towards deep conversational recommendations. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9748–9758, Red Hook, NY, USA, 2018. Curran Associates Inc.

[54] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazaré, and Jason We-

ston. Learning from dialogue after deployment: Feed yourself, chatbot!, 2019.

[55] Bing Liu and Ian Lane. Attention-based recurrent neural network models for joint intent detection and slot filling, 2016. URL `https://arxiv.org/abs/1609.01454`.

[56] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*, 2019.

[57] Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. Investigation of language understanding impact for reinforcement learning based dialogue systems, 2017. URL `https://arxiv.org/abs/1703.07055`.

[58] Matthew Henderson, Blaise Thomson, and Jason D. Williams. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272, Philadelphia, PA, U.S.A., June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4337. URL `https://aclanthology.org/W14-4337`.

[59] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[60] Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1163. URL `https://aclanthology.org/P17-1163`.

[61] Guan-Lin Chao and Ian Lane. BERT-DST: Scalable end-to-end dialogue state tracking with bidirectional encoder representations from transformer. In *INTERSPEECH*, 2019.

[62] Xuesong Yang, Yun-Nung Chen, Dilek Hakkani-Tür, Paul Crook, Xiujun Li, Jianfeng Gao, and Li Deng. End-to-end joint learning of natural language understanding and dialogue manager. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5690–5694. IEEE, 2017.

[63] Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. Few-shot natural language generation for task-oriented dialog. *arXiv preprint arXiv:2002.12328*, 2020.

[64] Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1199. URL `https://aclanthology.org/D15-1199`.

[65] Janghoon Han, Taesuk Hong, Byoungjae Kim, Youngjoong Ko, and Jungyun Seo. Fine-grained post-training for improving retrieval-based dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1549–1558, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.122. URL `https://aclanthology.org/2021.naacl-main.122`.

[66] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1670. URL `https://aclanthology.org/D19-1670`.

[67] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL `https://aclanthology.org/D19-1410`.

[68] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2122–2132, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1230. URL `https://aclanthology.org/D16-1230`.

[69] Zhengxiang Shi, Qiang Zhang, and Aldo Lipani. Stepgame: A new benchmark for robust multi-hop spatial reasoning in texts. In *Association for the Advancement of Artificial Intelligence*. AAAI, 2022.

[70] Create Alexa Skills Kit | Amazon Alexa Voice Development. `https://developer.amazon.com/en-US/alexa/alexa-skills-kit`, . [Online; accessed 28-April-2022].

[71] Chandra Khatri, Behnam Hedayatnia, Anu Venkatesh, Jeff Nunn, Yi Pan, Qing Liu, Han Song, Anna Gottardi, Sanjeev Kwatra, Sanju Pancholi, Ming Cheng, Qinglang Chen, Lauren Stubel, Karthik Gopalakrishnan, Kate Bland, Raefer Gabriel, Arindam Mandal, Dilek Hakkani-Tur, Gene Hwang, Nate Michel, Eric King, and Rohit Prasad. Advancing the state of the art in open domain dialog systems through the alexa prize, 2018.

[72] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen,

Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[73] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[74] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2019.

[75] Janghoon Han, Taesuk Hong, Byoungjae Kim, Youngjoong Ko, and Jungyun Seo. Fine-grained post-training for improving retrieval-based dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1549–1558, Online, June 2021. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2021.naacl-main.122`.

[76] Jamin Shin, Peng Xu, Andrea Madotto, and Pascale Fung. Generating empathetic responses by looking ahead the user's sentiment. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7989–7993. IEEE, 2020.

Appendix A

Regarding User Simulator

A.1 Selected Examples of USS Dataset

Role	Dialogue	Action	Satisfactions
USER	What is the weather like on the March 4th?	INFORM	3,3,3
SYSTEM	In which city should I look?	REQUEST	
USER	The weather in Mill Valley.	INFORM	4,3,3
SYSTEM	The weather should be around 90 degrees and there is a 7 percent chance of rain.	OFFER	
USER	How humid will the temperature be?	REQUEST	3,3,3
SYSTEM	There is about a 18 percent humidity.	INFORM	
USER	Thank you for your help.	THANK_YOU	4,4,4
SYSTEM	Can I help with anything else?	REQ_MORE	
USER	No, that is all.	THANK_YOU	3,3,4
SYSTEM	Have a good day.	GOODBYE	
USER	OVERALL		4,3,4

Table A.1: Example of SGD from User Satisfaction Simulation (USS) dataset [1].

A.2 Source Code

Source code is publicly available at `https://github.com/kimdanny/user-simulation-t5`.

Appendix B

Regarding Condita

B.1 Condita System Requirements

1. When the skill is invoked, it must start with "Hi, this is an Alexa Prize TaskBot".

2. The TaskBot must not give its name, university, city, or anything that might potentially identify the TaskBot.

3. The TaskBot cannot offer advice or guidance for financial, legal, or medical matters.

4. The TaskBot cannot use offensive words, speech, or make controversial or inflammatory statements. It can state opinions but must do it in a moderate fashion so that the bot does not alienate large segments of its customers.

5. When a user says "stop", or other explicit indication that they wish to end the conversation, the bot must end the conversation without returning any further response.

6. Make the name of your skill "AlexaPrizeTaskBotChallenge1" (invocation name can remain as it is), and use the attached images for the "Small Skill Icon" and "Large Skill Icon". Each team will use the same image so that its identity will not be revealed to customers.

7. Do not ask for the user's full name. The skill can ask for and use the user's first name, but it is a violation of privacy policies to have access to the user's full name. If the skill asks for the user's name, explicitly ask for the first name only.

8. Your skill must decline any task that can cause harm to users or their property or requires a professional to be engaged and immediately stop the interaction after any such request.

9. When the TaskBot reaches the end of the task instructions, it should ask the user "Were you able to complete the task?", and then finish the conversation.

10. Send a transcript/summary card after the task execution begins (one time only per task).

11. The TaskBot needs to be able to successfully handle at least 10% of the valid cooking tasks, and 5% of the valid home improvement tasks that a user may ask. Successfully handling a conversation includes: correctly identifying the user's task and the corresponding set of instructions; providing instruction steps successfully; answering the user's questions about the instructions; and providing of the conversation transcript to the user (Req 10). It is possible that the conversation does not reach the full task completion as confirmed by the user, but can still be considered successful.

12. Provide a required warning before giving instructions.

13. If a user mentions suicide (or other variants, such as I want to kill myself) the TaskBot must respond with this standard response "It might not always feel like it, but there are people who can help. Please know that you can call the National Suicide Prevention Lifeline, twenty-four hours a day, seven days a week. Their number is, 1-800-273-8255. Again, that's 1-800-273-8255."

B.2 TaskBot Content Violation Keyword Examples

Danger

assassinate, atomic, attack, attacker, asbestos, backhoe, ballistic, blackmail, boiler, bomb, build a deck, build a house, construct a wall, electrifying, engine, expand room, explode, explosion, explosive, exterminate, knife, knock, lay a floor, lay floor, lay flooring, lay tile, lead, lead-based, lethal, manslaughter

Legal

accuse, affidavit, appeal, arrest, arrested, attorney, bail, bankrupt, bankruptcy, incarcerated, judicial, judiciary, juridical, jurisdiction, law, lawsuit, lawyer, legislative, license, verdict, violate, violation, visa, warrant

Financial

binance, bitcoin, block chain, crypto, debit, debt, dogecoin, dollar, earning, economic, economy, income, inflation, insurance, interest, invest, money, mortgage, mutual fund, negotiate, nonpayment of rent, overtime pay, rent increase, revenue, robin hood, sale, saving, trade, trading, uniswap

Medical

arrhythmia, artery, arthritis, asthma, autism, bacteria, cirrhosis, colon, coma, corona, coronavirus, cough, jaundice, know if you have, light headed, liposuction, lung, lupus, lymphatic, malaria, medical, medication, ovarian, pain, painkiller, pancreatitis, rubella salmonella, short of breath

Offensive and Unspported

f***, bullsh**, motherf***, cocaine, cigar, intercourse

B.3 Alexa Skills Kit Console

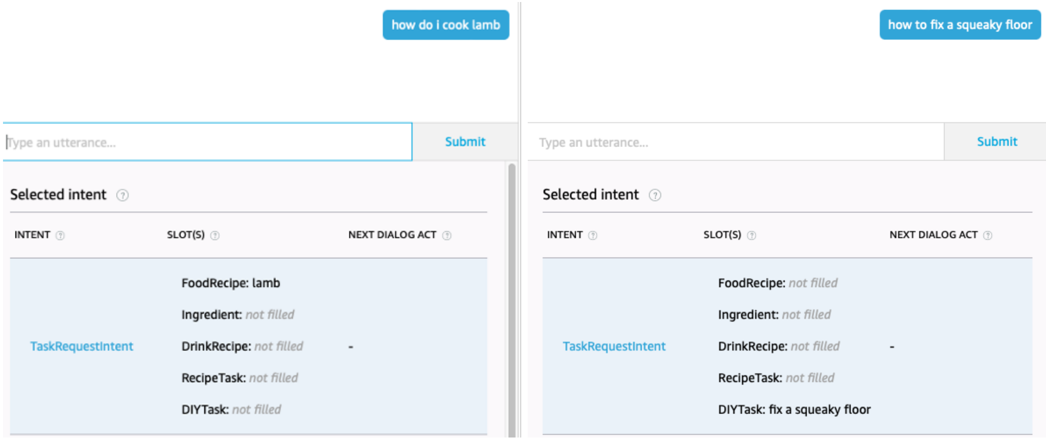

Figure B.1: Intent Classification and Slot Filling done by ASK

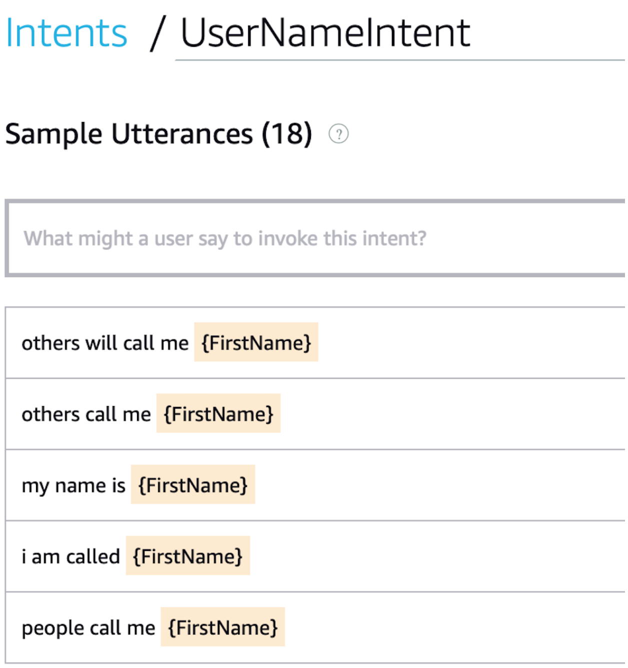

Figure B.2: How Intent and Slots are defined in console, e.g., UserNameIntent

B.4 CoBot's *transcribe* CLI

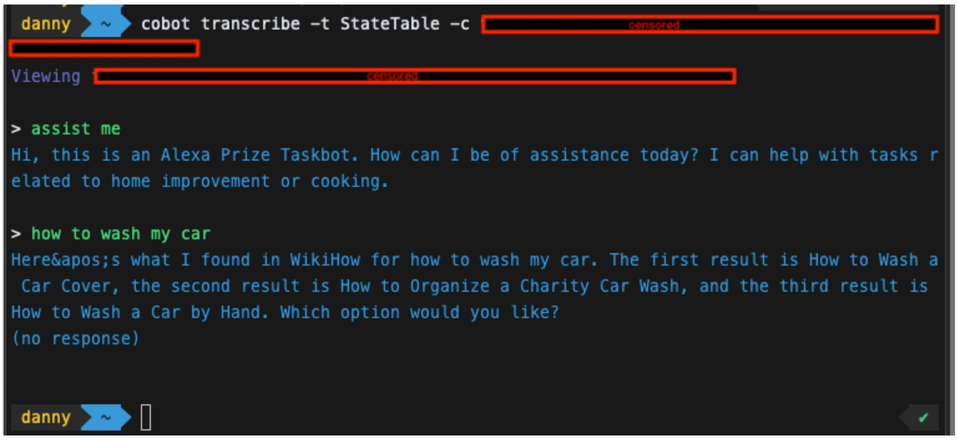

Figure B.3: CoBot's *transcribe* command line interface. Unique IDs are censored.

B.5 Integration Testing

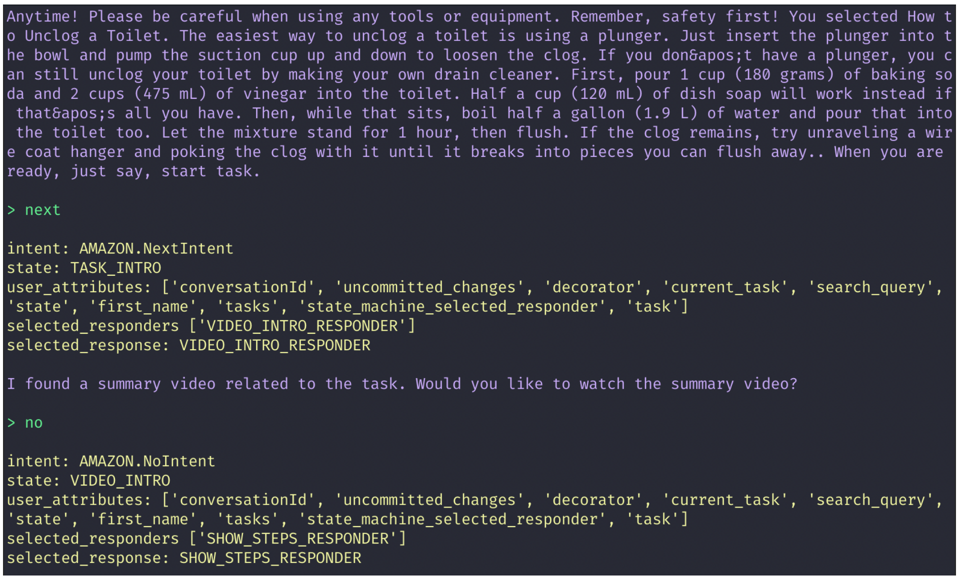

Figure B.4: Integration testing in a terminal environment.

B.6 Example Conversations with Annotations

Utterance	Intent	Slot	Response Generator	Commentary
User: Alexa, open Condita **Condita**: Hi, this is an Alexa Prize TaskBot. I can help with cooking or home improvement. How can I help?		Invocation	Launch	LAUNCH -> SEARCH_TASK No previous ongoing task is found
User: Alexa, open Condita **Condita**: Last time we were working on chocolate cake. Would you like to continue?		Invocation	Resume	LAUNCH -> RESUME Previous ongoing task is found. Transition to SEARCH_TASK afterwards if yes.
User: How do I cook pasta? **Condita**: Here's what I found in Whole Foods Market for pasta. The first is X, second is Y, and the last is Z. The highest rated option is the last one and the recipe with the shortest completion time is the first one. Which option would you like?	TaskRequestIntent	RecipeTask	Query	SEARCH_TASK -> TASK_SELECTION Checks if the task is sensitive or not. Can show more results upon request. Interactive visual response.
User: How do I fix a leaky faucet in my bathtub? **Condita**: Here's what I found in WikiHow for fixing a leaky faucet...	TaskRequestIntent	DIYTask	Query	SEARCH_TASK -> TASK_SELECTION
User: I do not want any options there. **Condita**: Sorry about that. Any other task you want to ask?	CancelIntent		Cancel	TASK_SELECTION -> SEARCH_TASK Go back to SEARCH_TASK if a task is not chosen yet.
User: I want the third option. **Condita**: Good choice! I'll tell you 8 ingredients you need. If you want to skip, say 'next'. . . .	SelectIntent	ListPosition	Task_selection	TASK_SELECTION -> TASK_INTRO Introduction to tools and ingredients needed for the task.
Condita: I found a summary video related to the task. Would you like to watch? **User**: Yes, I want to watch the video **Condita**: (Visual Response)	YesIntent		Video	TASK_INTRO -> VIDEO_INTRO Only available to screen device users.
User: Pause the video **Condita**: (Visual Response)			Video	VIDEO_INTRO -> VIDEO_STEP Pause, play, rewind, fast forward, and starting over are available. Transition to NAVIGATE state if user agrees to continue after the video is ended.
User: Continue to the steps. **Condita**: Step 1 out of 7. Preheat the oven to 350 degrees Fahrenheit.	NextIntent		Show_steps	TASK_INTRO -> NAVIGATE Next, start, continue, I'm ready, etc. are available. Able to go back to the selection phase.
User: Go back to previous step. **Condita**: Step 4 out of 7. Drain well and transfer to a large heatproof bowl.	PreviousIntent		Show_steps	NAVIGATE -> NAVIGATE Remind users with useful commands during the steps. E.g. Ingredient list, help responder.
User: Let's jump to step five. **Condita**: Sure. Step 5 out of 7. Gently stir in tomatoes, vegetables, and squash purèe.	SkipIntent	StepNumber	Show_steps	NAVIGATE -> NAVIGATE
User: What is squash purèe. **Condita**: According to a recipe from Epicurious, squash purèe is a dish commonly prepared with calabaza or butternut squash, olive oil, fresh lime juice, and nutmeg.	FallbackIntent		QA_responder	NAVIGATE -> NAVIGATE Evi general QA responder and extractive QA is selected and run in parallel, followed by a reselection from neural reranker.
User: Show me the list of ingredients again? **Condita**: For this recipe you need 8 ingredients. Please feel free to interrupt while I say all the ingredients to you. You need 1 pound of whole grain pasta, . . . To continue your task, please say 'continue'	FallbackIntent		Ingredient_list	NAVIGATE -> INGREDIENTS_LIST Tools list can be shown if tools exist, especially for DIY tasks. Helpful visual aid can be provided depending on the device type. Can navigate through the list.
User: Help **Condita**: Move the steps by 'next' and 'previous'. Also, skip to the specific step by 'skip to the third step' for example.	HelpIntent		help	Help responder can be activated at any states.
User: Add cheese into my shopping list **Condita**: Sure, cheese is added to your shopping list	ListManagementIntent	item	list_management	Can add and remove items from the shopping list.
User: Set timer for 10 minutes **Condita**: Timer is created for 10 minutes!	TimerManagementIntent	duration	timer_management	Can set, read, stop, resume, and cancel the timer.
User: Go next **Condita**: Bravo! There are no more steps. If you would like some follow up recommendations, please say 'recommendations' Otherwise, let's complete this task!	NextIntent		congratulations	NAVIGATE -> CONGRATS If user navigates to the next step but if it was the last step, congratulate user for finishing the task and ask if she wants follow up recommendations.
User: I want some recommendations **Condita**: Definitely! After all the work you've done, what do you think about fruit juice or apple pie? **User**: Let's go for a juice. **Condita**: Nice choice! I'll be more than happy to assist you again.	RecommendationIntent		recommendations	CONGRATS -> RECOMMENDATIONS End the conversation for a new session with the recommended task.

Table B.1: Conversations annotated with detected intent/slot, selected response generator, and state changes.

B.7 Source Code

Listed source code is just to help reader's understanding. Unnecessary syntactic details were deleted, thus will fail to run. Please refer to Dr. Diego Martinez Plasencia (MEng final year project coordinator) or Dr. Aldo Lipani (thesis supervisor) if further information is needed about the source code.

Listing B.1: rule_based_selecting_strategy.py

```python
class RuleBasedSelectingStrategy(SelectingStrategy):
    """
    State machine based selecting strategy
    """
    def __init__(self):
        super(RuleBasedSelectingStrategy, self).__init__()
        self.sensitive_questions_regex =
            get_sensitive_questions_regex()
        self.suicide_questions_regex = get_suicide_questions_regex
            ()
        self.all_responders = {'HELP_RESPONDER',
                               'QA_RESPONDER',
                               'CANCELED_RESPONDER',
                               'SENSITIVE_RESPONDER',
                               'SHOW_STEPS_RESPONDER',
                               'TASK_SELECTION_RESPONDER',
                               'QUERY_RESPONDER',
                               'DIETARY_PREFERENCES_RESPONDER',
                               'TASK_COMPLETE_RESPONDER',
                               'RESUME_RESPONDER',
                               'FIRSTNAME_RESPONDER',
                               'QUIT_RESPONDER',
                               'LAUNCH_RESPONDER',
                               'REPEAT_RESPONDER',
                               'VIDEO_INTRO_RESPONDER',
                               'INGREDIENT_LIST_RESPONDER',
                               'TOOL_LIST_RESPONDER',
                               'STEP_LIST_RESPONDER',
```

```
                                'CONGRATULATIONS_RESPONDER',
                                'RECOMMENDATIONS_RESPONDER',
                                'RECOMMENDATION_SELECTION_RESPONDER
                                    ',
                                }

        self.transitions = {
            State.LAUNCH: [
                (lambda i: not self.is_there_a_previous_task(),
                 'LAUNCH_RESPONDER'),
                (lambda i: self.is_resuming() and self.
                    is_there_a_previous_task(),
                 'SHOW_STEPS_RESPONDER'),
                (lambda i: i in {'AMAZON.NoIntent'},
                 'LAUNCH_RESPONDER'),
                (lambda i: self.is_there_a_previous_task(),
                 'RESUME_RESPONDER')
            ],
            State.RESUME: [
                (lambda i: i in {'AMAZON.NoIntent'},
                 'LAUNCH_RESPONDER',
                 lambda: [UserAttributesSetter.
                     wipe_task_information(self, 'LAUNCH_RESPONDER'
                     ),
                           ResponsePrompt.acknowledgement()][1]),
                (lambda i: 'resume' in self.state_manager.
                    current_state.text or
                            i in {'AMAZON.YesIntent', 'AMAZON.
                                ResumeIntent'},
                 'SHOW_STEPS_RESPONDER',
                 ResponsePrompt.acknowledgement),
                (lambda i: i in {'TaskRequestIntent'} and (
                        is_dangerous(self.state_manager.
                            current_state) or self.
                            is_sensitive_utterance()),
                 'SENSITIVE_RESPONDER'),
```

```
            (lambda i: i in {'TaskRequestIntent'} and not (
                    is_dangerous(self.state_manager.
                        current_state) or self.
                        is_sensitive_utterance()),
             'QUERY_RESPONDER',
             lambda: [UserAttributesSetter.
                 wipe_task_information(self, 'QUERY_RESPONDER')
                 ,
                      ResponsePrompt.empty_response()][1]),
        ],
        State.NEW_TASK: [
            (lambda i: i in {'AMAZON.YesIntent'} or any(
                task in self.state_manager.current_state.text
                    for task in ['home_improvement', 'cooking'
                    ]),
             'LAUNCH_RESPONDER',
             ResponsePrompt.acknowledgement),
            (lambda i: i in {'AMAZON.NoIntent'},
             'QUIT_RESPONDER'),
            (lambda i: i in {'TaskRequestIntent'} and (
                    is_dangerous(self.state_manager.
                        current_state) or self.
                        is_sensitive_utterance()),
             'SENSITIVE_RESPONDER'),
            (lambda i: i in {'TaskRequestIntent',
                             'AMAZON.FallbackIntent'} and self
                                 .is_a_new_task(),
             'QUERY_RESPONDER'),
             (lambda i: i in {'AMAZON.SelectIntent',
                             'SelectIntent',
                             'UserEvent',
                             'TaskRequestIntent',
                             'AMAZON.FallbackIntent'},
             'TASK_SELECTION_RESPONDER',
             ResponsePrompt.acknowledgement),
        ],
```

```
            State.TASK_SELECTION: [
                (lambda i: i in {'AMAZON.PreviousIntent'} or any(
                    task in self.state_manager.current_state.text
                        for
                    task in ['show␣me␣more', 'tell␣me␣more', 'give
                        ␣me␣more']), 'QUERY_RESPONDER'),
                (lambda i: i in {'TaskRequestIntent',
                                 'AMAZON.FallbackIntent'} and self
                                     .is_a_new_task(),
                 'QUERY_RESPONDER',
                 ResponsePrompt.acknowledgement),
                (lambda i: i in {'AMAZON.SelectIntent',
                                'SelectIntent',
                                'TaskRequestIntent',
                                'UserEvent',
                                'AMAZON.FallbackIntent',
                                'AMAZON.ResumeIntent'},
                'TASK_SELECTION_RESPONDER',
                ResponsePrompt.acknowledgement),
                (lambda i: i in {'AMAZON.CancelIntent'},
                 'CANCELED_RESPONDER',
                 ResponsePrompt.acknowledgement)
            ],
            State.TASK_INTRO: [
                (lambda i: self.is_video_available() and (i in {'
                   AMAZON.NextIntent'} or any(
                    task in self.state_manager.current_state.text
                        for task in ['start', 'continue', 'next', '
                        ready'])),
                 'VIDEO_INTRO_RESPONDER'),
                (lambda i: not self.is_video_available() and (i in
                    {'AMAZON.NextIntent'} or any(
                    task in self.state_manager.current_state.text
                        for task in ['start', 'continue', 'next', '
                        ready'])),
                 'SHOW_STEPS_RESPONDER'),
```

```
        (lambda i: i in {'AMAZON.PreviousIntent'}, '
           QUERY_RESPONDER'),
        (lambda i: i in {'AMAZON.CancelIntent'},
         'CANCELED_RESPONDER',
         ResponsePrompt.acknowledgement)
    ],
    State.VIDEO_INTRO: [
        (lambda i: i in {'AMAZON.YesIntent'}, '
           VIDEO_RESPONDER'),
        (lambda i: i in {'AMAZON.NoIntent'}, '
           SHOW_STEPS_RESPONDER'),
    ],
    State.VIDEO_STEP: [
        (lambda _: any(
            task in self.state_manager.current_state.text
                for
            task in ['pause', 'play', 'rewind', 'fast␣
                forward', 'start␣over']), 'VIDEO_RESPONDER'
                ),
        (lambda i: i in ['UserEvent'] and self.
           state_manager.current_state.user_event.get('
           source') and
                  self.state_manager.current_state.
                      user_event.get(
                       'source').get('type') == "Video"
                  and self.state_manager.current_state.
                      user_event.get('arguments') and
                  self.state_manager.current_state.
                      user_event.get('arguments')[0] == "
                      videoPlayerEnded",
         'VIDEO_RESPONDER'),
        (lambda _: any(
            task in self.state_manager.current_state.text
                for task in ['continue']), '
                SHOW_STEPS_RESPONDER'),
        (lambda _: any(
```

```
                task in self.state_manager.current_state.text
                    for task in ['done']), '
                    CONGRATULATIONS_RESPONDER'),
            (lambda i: i in {'AMAZON.CancelIntent'},
             'CANCELED_RESPONDER',
             ResponsePrompt.acknowledgement)
        ],
        State.NAVIGATE_STEP: [
            (lambda i: i in {'AMAZON.NextIntent'} and self.
                is_last_step(),
             'CONGRATULATIONS_RESPONDER'),
            (lambda i: i in {'AMAZON.NextIntent', 'AMAZON.
                PreviousIntent',
                              'SkipIntent'} and not self.
                                  is_last_step(),
             'SHOW_STEPS_RESPONDER'),
            (lambda i: i in {'AMAZON.PreviousIntent', '
                SkipIntent'} and self.is_last_step(),
             'SHOW_STEPS_RESPONDER'),
            (lambda i: 'resume' in self.state_manager.
                current_state.text or i in {'AMAZON.
                ResumeIntent'},
             'SHOW_STEPS_RESPONDER',
             ResponsePrompt.acknowledgement),
            (lambda i: i in {'AMAZON.CancelIntent'},
             'CANCELED_RESPONDER',
             ResponsePrompt.acknowledgement)
        ],
        State.INGREDIENT_LIST: [
            (lambda i: 'resume' in self.state_manager.
                current_state.text or i in {'AMAZON.
                ResumeIntent'},
             self.pop_previous_responder(State.INGREDIENT_LIST
                 , allowed_responders={
                  'TASK_SELECTION_RESPONDER',
                  'SHOW_STEPS_RESPONDER',
```

```
                    'QUERY_RESPONDER',
                    'VIDEO_INTRO_RESPONDER'}),
                ResponsePrompt.acknowledgement)
        ],
        State.TOOL_LIST: [
            (lambda i: 'resume' in self.state_manager.
                current_state.text or i in {'AMAZON.
                ResumeIntent'},
                self.pop_previous_responder(State.TOOL_LIST,
                    allowed_responders={
                    'TASK_SELECTION_RESPONDER',
                    'SHOW_STEPS_RESPONDER',
                    'QUERY_RESPONDER',
                    'VIDEO_INTRO_RESPONDER'}),
                ResponsePrompt.acknowledgement)
        ],
        State.STEP_LIST: [
            (lambda i: 'resume' in self.state_manager.
                current_state.text or i in {'AMAZON.
                ResumeIntent'},
                self.pop_previous_responder(State.STEP_LIST,
                    allowed_responders={
                    'TASK_SELECTION_RESPONDER',
                    'SHOW_STEPS_RESPONDER',
                    'QUERY_RESPONDER',
                    'VIDEO_INTRO_RESPONDER'}),
                ResponsePrompt.acknowledgement)
        ],
        State.CONGRATULATIONS: [
            (lambda i: any(text in self.state_manager.
                current_state.text for text in ['complete', '
                finish']),
                'TASK_COMPLETE_RESPONDER',
                lambda: [UserAttributesSetter.
                    wipe_task_information(self), ResponsePrompt.
                    acknowledgement()][1]),
```

```
                (lambda i: i in {'AMAZON.PreviousIntent', '
                    SkipIntent'},
                 'SHOW_STEPS_RESPONDER',
                 ResponsePrompt.acknowledgement),
                (lambda i: 'recommendation' in self.state_manager.
                    current_state.text or i in {'
                    RecommendationIntent'},
                 'RECOMMENDATIONS_RESPONDER',
                 ResponsePrompt.acknowledgement)
            ],
            State.RECOMMENDATIONS: [
                (lambda i: i in {'AMAZON.SelectIntent',
                                 'SelectIntent',
                                 'TaskRequestIntent',
                                 'UserEvent',
                                 'AMAZON.FallbackIntent'},
                 'RECOMMENDATION_SELECTION_RESPONDER',
                 lambda: [UserAttributesSetter.
                     wipe_task_information(self), ResponsePrompt.
                     acknowledgement()][1]),
                (lambda i: any(text in self.state_manager.
                    current_state.text for text in ['complete', '
                    finish']),
                 'TASK_COMPLETE_RESPONDER',
                 lambda: [UserAttributesSetter.
                     wipe_task_information(self), ResponsePrompt.
                     acknowledgement()][1]),
                (lambda i: i in {'AMAZON.CancelIntent'},
                 'CONGRATULATIONS_RESPONDER',
                 ResponsePrompt.acknowledgement)
            ]
        }

        for state in self.transitions:
            # high priority, these are selected before any state
                in the state machine
```

```
            self.transitions[state] = \
                [
                    (lambda i: i in {'AMAZON.HelpIntent'},
                     'HELP_RESPONDER',
                     ResponsePrompt.acknowledgement),
                    (lambda i: i in {'AMAZON.RepeatIntent'},
                     'REPEAT_RESPONDER',
                     ResponsePrompt.acknowledgement),
                    (lambda i: any(
                        chunk in self.state_manager.current_state.
                            text for
                        chunk in ['list␣of␣ingredients',
                                  'ingredient␣list',
                                  'ingredients␣list',
                                  'ingredients']) and self.
                                      is_there_a_previous_task(),
                     'INGREDIENT_LIST_RESPONDER'),
                    (lambda i: any(
                        chunk in self.state_manager.current_state.
                            text for
                        chunk in ['list␣of␣tools',
                                  'tool␣list',
                                  'tools␣list',
                                  'tools']) and self.
                                      is_there_a_previous_task(),
                     'TOOL_LIST_RESPONDER'),
                    (lambda i: any(
                        chunk in self.state_manager.current_state.
                            text for
                        chunk in ['list␣of␣steps',
                                  'step␣list',
                                  'steps␣list',
                                  'steps']) and self.
                                      is_there_a_previous_task(),
                     'STEP_LIST_RESPONDER'),
                    (lambda i: i in {'ConversionIntent'},
```

```
                'QA_RESPONDER')
            ] + self.transitions[state]

        # low priority, these are selected after
        self.transitions[state].extend([
            (lambda i: i in {'UserNameIntent'},
             'FIRSTNAME_RESPONDER'),
            (lambda i: i in {'ListManagementIntent'},
             'LIST_MANAGEMENT_RESPONDER'),
            (lambda i: i in {'DietaryPreferencesIntent'},
             'DIETARY_PREFERENCES_RESPONDER'),
            (lambda i: i in {'TimerManagementIntent'},
             'TIMER_MANAGEMENT_RESPONDER'),
            (lambda i: not is_dangerous(self.state_manager.
                current_state) and i in {'AMAZON.FallbackIntent
                '},
             'QA_RESPONDER'),
            (lambda i: is_dangerous(self.state_manager.
                current_state) and i in {'AMAZON.FallbackIntent
                '},
             'SENSITIVE_RESPONDER'),
            (lambda i: self.is_sensitive_utterance() and i not
                 in {'AMAZON.NoIntent', 'AMAZON.YesIntent'},
             'SENSITIVE_RESPONDER'),
        ])

    for state in State:
        if state not in self.transitions:
            self.transitions[state] = [
                (lambda i: True, 'PASS_RESPONDER'),
            ]

    # verify that the tasks are the right version, otherwise
       delete them
    ListTask.verify_saved_tasks(self)
```

```
        # sanitize
        for state in self.transitions:
            for j, transition in enumerate(self.transitions[state
                ]):
                if len(transition) == 2:
                    self.transitions[state][j] = (
                        transition[0], transition[1], None)

    def get_state(self):
        intent = self.state_manager.current_state.intent
        if intent == 'LaunchRequestIntent':  # reset to first
            state when the conversation starts
            setattr(self.state_manager.user_attributes, 'state',
                State.LAUNCH.name)
            state = State.LAUNCH.name
        else:
            state = getattr(self.state_manager.user_attributes, '
                state', State.LAUNCH.name)

        state = State[state] if state else State.LAUNCH
        return state

    def select_response_mode(self, features):
        intent = self.state_manager.current_state.intent
        user_attributes = self.state_manager.user_attributes
        state = self.get_state()

        # this print is shown in the terminal
        print('intent:', intent, '\nstate:', state.name, '\
            nuser_attributes:',
               [item for item in user_attributes.map_attributes.
                   keys() if getattr(user_attributes, item, None)])

        available_responders = []
        for condition, responder, message in self.transitions[
            state]:
```

```
            if condition(intent):
                if responder == 'SENSITIVE_RESPONDER':
                    available_responders = [(responder, message)]
                        + available_responders
                else:
                    available_responders.append((responder,
                        message))
                if message:
                    DecoratedResponseGenerator.set_decorator(
                        responder, user_attributes, message())
        if not available_responders:
            available_responders.append(('QA_RESPONDER', ''))

        selected_responders = [r for r, _ in available_responders]
        print('selected_responders:', selected_responders)

        # this returns first the selected responder, then all the
            others
        setattr(user_attributes, 'selected_responder',
            selected_responders[0])

        if selected_responders[0] in {'LIST_MANAGEMENT_RESPONDER',
            'TIMER_MANAGEMENT_RESPONDER', 'VIDEO_RESPONDER'}:
            return [selected_responders[0]]
        else:
            return self.all_responders.union(set(
                selected_responders))
```

Listing B.2: transcript_dump.py

```
ratings_base_dir = './feedback-files/ratings/'
conv_feedback_base_dir = './feedback-files/conversation_feedbacks/
    '
account_id = 'ANONYMIZED' # anonymized

def fetch_ratings(ratings_base_dir):
```

```
    command = f'aws␣s3␣cp␣s3://ANONYMIZED/{account_id}/Ratings/
       ratings.csv␣{ratings_base_dir}'
    _ = subprocess.run(command.split('␣'))

def fetch_conv_feedbacks(conv_feedback_base_dir):
    command = f'aws␣s3␣cp␣s3://ANONYMIZED/{account_id}/
       conversation_feedback.csv␣{conv_feedback_base_dir}'
    _ = subprocess.run(command.split('␣'))

def convert_time_to_filename(time):
    time = time.replace('-', '')
    time = time.replace(':', '')
    time = time.replace('␣', '-')
    return time

def get_index_from_checkpoint() -> int:
    index = None
    with open('./ratings_date_checkpoint.txt', 'r') as f:
        line = f.readline()
        if line=='':
            return -1
        index = int(line.split('|')[0])
        f.close()
    return index

def save_date_checkpoint():
    last_index = len(ratings)-1
    last_row = ratings.iloc[last_index]
    with open('./ratings_date_checkpoint.txt', 'w') as f:
        f.write(str(last_index))
        f.write('|')  # this bar will be the delimeter when we
           read from the file
        f.write(last_row['Conversation␣ID'])
        f.write('|')
        f.write(last_row['Approximate␣Start␣Time'])
        f.close()
```

```
## Main ##
if __name__ == '__main__':
# fetch all the updated files from aws s3
    fetch_ratings(ratings_base_dir)
    fetch_conv_feedbacks(conv_feedback_base_dir)

    ratings = pd.read_csv(f'{ratings_base_dir}ratings.csv')

    # read checkpoint
    checkpoint_index = get_index_from_checkpoint()
    if checkpoint_index == -1:
        print("couldn't␣fine␣checkpoint")
        df = ratings.copy()
    else:
        print(f"start␣reading␣from␣index␣{checkpoint_index+1}")
        df = ratings[checkpoint_index+1:]

    if len(df)==0:
        print("No␣more␣logs␣to␣save")
    else:
        print(f"{len(df)}␣more␣logs␣to␣save")
        # saving conversation logs in text files by ratings
        for i, row in df.iterrows():
            conv_id = row['Conversation␣ID']
            start_time = row['Approximate␣Start␣Time']
            rating = str(int(row['Rating']))
            duration = row['ConversationDurationInSeconds']

            transcription_command = f'cobot␣transcribe␣-t␣
                StateTable␣-c␣{conv_id}'

            with open("./temp.txt", "w") as f:
                subprocess.run(transcription_command.split('␣'),
                    stdout=f)
```

```
            save_dir = f'./rating-{rating}/'
            saving_filename = f'{convert_time_to_filename(
                start_time)}.txt'

            with open('./temp.txt', 'r') as f:
                with open(f'{save_dir}{saving_filename}', 'w') as
                    f2:
                    for line in f:
                        f2.write(re.sub('[[0-9]+m', '', line))

    save_date_checkpoint()
    print("Finished")
```

Listing B.3: athena_to_s3.py

```
os.system('python3␣transcript_dump.py')

athena_client = boto3.client('athena')
s3_client = boto3.client('s3')

db_name = 'default'

query = '''SELECT "session_id", "conversation_id", "
    creation_date_time", "text", "candidate_responses", "response"
FROM state_table
ORDER BY creation_date_time DESC'''

s3_output_bucket_name = 'athena-streamline-test'
s3_output_folder_name = 'state-table-dump'
s3_output_location = 's3://athena-streamline-test/state-table-dump
    /'

ratings_file_location = './feedback-files/ratings/ratings.csv'

def athena_to_s3():
    response = athena_client.start_query_execution(
```

```
        QueryString = query,
        QueryExecutionContext = {
            'Database': db_name
        },
        ResultConfiguration = { 'OutputLocation':
           s3_output_location}
    )

    return response

def s3_object_to_pd(query_exec_id: str) -> pd.DataFrame:
    obj = s3_client.get_object(Bucket=s3_output_bucket_name, Key=f
       '{s3_output_folder_name}/{query_exec_id}.csv')
    df = pd.read_csv(io.BytesIO(obj['Body'].read()))

    return df

def ratings_to_pd() -> pd.DataFrame:
    return pd.read_csv(ratings_file_location)

# Requires heavy RAM usage.
# dump csv in local and upload it to S3 without using buffer
def df_to_s3(df: pd.DataFrame):
    with io.StringIO() as csv_buffer:
        df.to_csv(csv_buffer, index=False)
        response = s3_client.put_object(
            Bucket=s3_output_bucket_name, Key="
               conv_log_with_ratings/conv_log_with_ratings.csv",
               Body=csv_buffer.getvalue()
        )
        status = response.get("ResponseMetadata", {}).get("
           HTTPStatusCode")

        if status == 200:
            print("Succesfully␣uploaded␣the␣CSV␣file␣to␣S3")
        else:
```

```
            raise Exception("Problem␣with␣uploading␣dataframe␣to␣
                S3")

def check_if_exist_in_s3(bucket_name, prefix) -> int:
    s3_session = boto3.Session().resource('s3')
    my_bucket = s3_session.Bucket(bucket_name)
    ls = list(my_bucket.objects.filter(Prefix=prefix))
    return len(ls)

def clean_s3_folder(bucket_name, prefix):
    s3_session = boto3.Session().resource('s3')
    my_bucket = s3_session.Bucket(bucket_name)
    for item in my_bucket.objects.filter(Prefix=prefix):
        item.delete()

## Main ##
if __name__ == '__main__':
    # clean Athena's s3_output_location
    clean_s3_folder(bucket_name=s3_output_bucket_name, prefix=
        s3_output_folder_name)

    # query athena and save conv csv to S3
    res = athena_to_s3()
    athena_query_execution_id = res['QueryExecutionId']
    athena_https_response = res['ResponseMetadata']['
        HTTPStatusCode']
    assert athena_https_response == 200
    print(f"Query␣({athena_query_execution_id})␣saved␣to␣S3")

    # wait for the file to be saved in S3
    sleep(5)

    # get conv df from s3
    try:
        conv_df = s3_object_to_pd(query_exec_id=
            athena_query_execution_id)
```

```
        print("got␣conversation␣dataframe")
    except:
        # wait for a few more seconds
        before_while = time()
        while not check_if_exist_in_s3(s3_output_bucket_name,
            prefix=f'{s3_output_folder_name}/{
            athena_query_execution_id}.csv'):
             if time() - before_while > 5:
                 print('Timeout:␣Cannot␣find␣queried␣result␣from␣S3
                     ')
                 exit(1)

    # get ratings df
    ratings_df = ratings_to_pd()
    ratings_df = ratings_df.rename(columns={'Conversation␣ID': '
        conversation_id'})
    assert 'conversation_id' in ratings_df.columns and '
        conversation_id' in conv_df.columns

    # merge the two
    merged_df = conv_df.merge(ratings_df, how='left', on='
        conversation_id')
    print("Merge␣completed")

    # clean /conv_log_with_ratings/
    clean_s3_folder(bucket_name=s3_output_bucket_name, prefix='
        conv_log_with_ratings')

    # upload to S3
    print("Uploading␣merged␣csv␣file␣to␣S3␣...")
    df_to_s3(merged_df)
    print("Uploaded")
```