MUST: A Framework for Training Task-oriented Dialogue Systems with Multiple User SimulaTors

Anonymous ACL submission

Abstract

Recent works try to optimize a Task-oriented Dialogue System with reinforcement learning (RL) by building user simulators. However, most of them only focus on training the dialogue system using a single user simulator. In this paper, we propose a framework called MUST to improve the dialogue agent by utilizing multiple user simulators simultaneously shown in Figure 1. Two core research problems of the proposed MUST are: (1) how to specify these different simulators effectively in the RL training? and (2) what model architecture should we use to learn a user simulator with better generalization capability? To tackle the first problem, we formulate the simulator selection task to train the system agent as a Multi-armed bandit (MAB) problem and modify one Upper Confidence Bound (UCB) algorithms called UCB1 to guide this selection process. To deal with the second problem, we present a new user simulator model called U-GPT based on the Generative Pre-trained Transformer (GPT). Extensive empirical results demonstrate that the dialogue system trained by the proposed MUST achieves a better performance than those trained by a single user simulator and our *modified* UCB1 algorithm can accelerate the MUST training. Furthermore, we reveal that our GPT-based user simulator outperforms previous learning-based simulators through direct and indirect evaluations.

1 Introduction

Task-oriented dialogue systems aim to help users accomplish their various tasks such as requesting information, restaurant reservations through natural language conversations. They have recently gained increasing attention in both academia and industries. Researchers usually divide the task-oriented dialogue systems into four components: Natural Language Understanding (NLU), Dialog State Tracker (DST), Dialog Policy Learning (POL), and Natural Language Generation (NLG). These different modules can be trained independently or jointly in an end-to-end manner (Ham et al., 2020; Peng et al., 2021; Hosseini-Asl et al., 2020). To build more intelligent and human-like dialogue systems, reinforcement learning (RL) is often adopted to learn system actions through interacting with users. However, directly interacting with human users is time-consuming and expensive. The most used approach is to build a user simulator that is agenda-based (Schatzmann et al., 2007; Schatzmann and Young, 2009) or learned with neural networks such as RNN based on a corpus of dialogues (Asri et al., 2016; Kreyssig et al., 2018; Gur et al., 2018) instead of real humans to train the system agent.

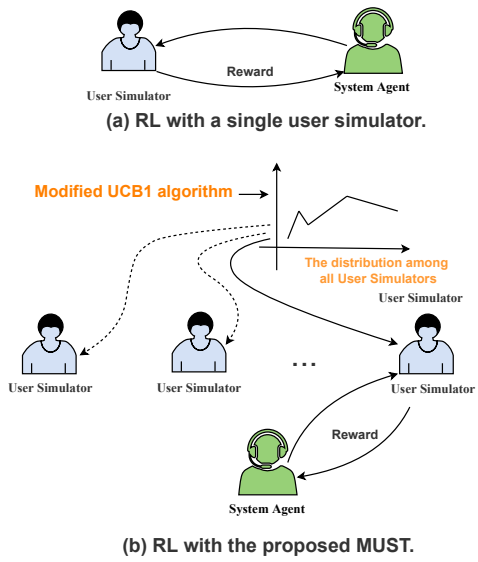


Figure 1: The comparison of training the system agent between previous works with a single simulator (a) and our proposed MUST with multiple user simulators (b).

To learn a user simulator, the agenda-based user simulators (ABUS) design handcrafted rules to mimic user behaviors, yet this approach is laborious and can hardly generalize across domains. The RNN-based simulators are prone to overfitting

and lack of mechanisms to flexibly explore various user behaviors. Existed works either train a system agent by building one user simulator with these methods or train different system agents using different user simulators (Shi et al., 2019) and pick the best performing one. In realistic scenarios, different users could have very different behaviors, we contend that the system agent trained by a single user simulator is not efficacious and a better way is to use multiple user simulators modeling different users to train the system agent simultaneously.

In this work, we propose a framework called MUST to utilize different user simulators simultaneously to train a system agent. Motivated by the Multi-armed bandit (MAB) problem, we treat each user simulator as a bandit's arm. As our goal is to obtain a more robust system agent, we modify one Upper Confidence Bound (UCB) (Auer et al., 2002) algorithm[1] called UCB1 to help the proposed MUST to accelerate the system agent training by learning more from those user simulators that the system agent has not performed very well i.e. low success rate during the RL training process. Furthermore, the quality of user simulators will affect the effectiveness of our MUST since we cannot obtain a good system agent if all user simulators are bad. To improve previous RNN-based methods on building user simulators and validate the effectiveness of MUST, we also present a new user simulator model named U-GPT. U-GPT recasts all sub-modules in modeling user simulators as a sequence prediction problem and completes all sub-tasks sequentially in an end-to-end manner by leveraging the auto-regressive language model GPT (Radford et al., 2018, 2019).

Our extensive experimental results on the restaurant search task from MultiWOZ (Budzianowski et al., 2018) show that the dialogue system trained by the proposed MUST achieves a better performance than those trained by any single user simulator, including the one (referred to as U-GPT$_{\text{IL}}$ in our later experiments.) trained with dialogue sessions sampled from different user simulators. Moreover, our newly proposed GPT-based user simulator is more preferable to previous RNN-based simulators through direct and indirect evaluations.

Altogether, our technical contribution in this work is three-fold: (1) To the best of our knowledge, we are the first to train the system agent using multiple user simulators, and we propose MUST with an effective RL perspective using a *modified* UCB1 algorithm. (2) We present a new model named U-GPT which leverages GPT to learn the user simulator. (3) The overall results show that the dialogue system trained with MUST performs best, and our newly proposed U-GPT by itself is more preferable to previous RNN-based simulators through direct and indirect evaluations.

[1]Selecting the arm maximizing the cumulative expected reward.

2 Preliminary

Before presenting our methods, we first provide some details about the Multi-armed bandit problem and the UCB1 algorithm (Auer et al., 2002).

2.1 Multi-armed Bandit Problem

Reinforcement learning policies face the exploration versus exploitation trade-off, which can be described as the search for a balance between exploring the environment to find profitable actions while taking the empirically best action as often as possible. This exploration vs exploitation dilemma has been widely studied as a Multi-armed bandit (MAB) problem.

In the MAB problem, there are K arms, and each arm j has a fixed but unknown reward distribution R_j with an expectation μ_j. At each time step $t = 1, 2, ..., T$, the decision maker must choose one of these K arms. We denote the arm pulled at time step t as $j_t \in \{1, ..., K\}$. After pulling an arm, it will receive a reward X_{j_t} which is a realization drawn from the arm's underlying reward distribution. The decision masker's objective is to maximize the cumulative expected reward over the time horizon $\sum_{t=1}^{T} E[X_{j_t}] = \sum_{t=1}^{T} \mu_{j_t}$.

UCB1. The Upper Confidence Bound (UCB) algorithms are classic allocation strategies to solve the MAB problem. Here we give an introduction to UCB1 (Auer et al., 2002), which is one of the UCB algorithms. This policy first pulls each arm once. Then the index of the arm will be played from $t = K + 1$ to T is the sum of two terms:

$$j_t = \arg\max_j \bar{X}_j + \sqrt{\frac{2 \ln t}{T_{j,t}}},$$

where $\bar{X}_j$ is the average reward obtained from arm j , $T_{j,t}$ is the number of times arm j has been played so far. The first term is simply the current average reward and the second one is related to the size of the one-sided confidence interval for the average

reward within which the true expected reward falls with overwhelming probability.

3 Methodology

3.1 Problems

We propose a framework called MUST to obtain a better system agent by utilizing multiple simulators simultaneously. There are two key points to implement this framework: the first is how to use these different simulators effectively in the RL training, and the second is what model architecture we should adopt to build a good simulator to be used with MUST. In the following two sections, we will detail our proposals to the above two problems.

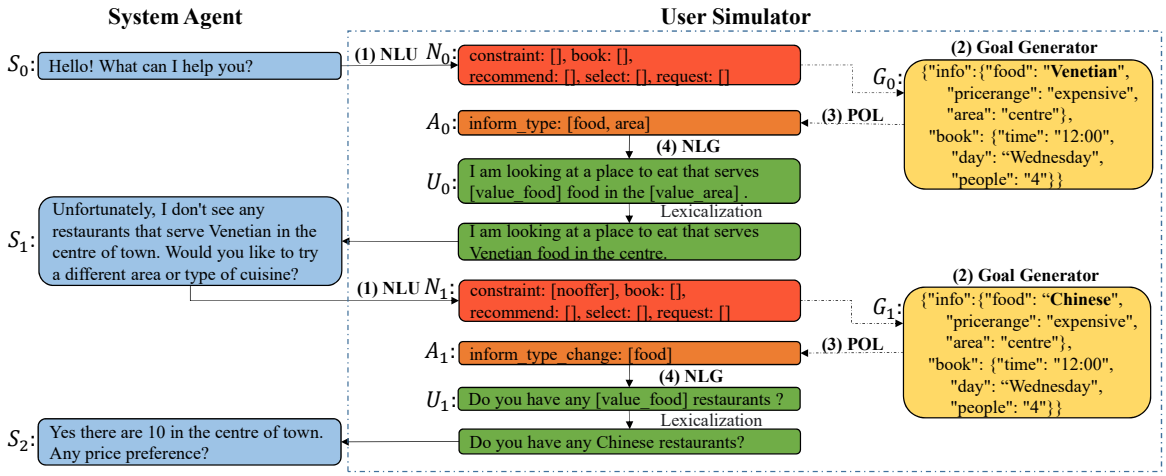
3.2 RL with Multiple User Simulators

Given K user simulators denoted as $U_1, U_2, \ldots, U_K$ which have different behaviors, we aim to train a system agent S with these simulators by RL. The simplest way is that we put these K user simulators in the RL environment and pick a simulator randomly with a uniform distribution to interact with S when collecting each new dialogue and use the obtained reward to update the system agent S with policy gradient. It is not efficient nor optimal because the system agents trained by different user simulators have different convergence speeds with RL. To allow the system agent to learn from different user simulators efficiently by MUST, we think it should sample the user simulators whose corresponding system agents are easy to converge fewer times and pay more attention to those user simulators whose corresponding system agents are harder to converge.

Therefore, motivated by the MAB problem, we treat each user simulator as a bandit's arm and modify the UCB1 algorithm to calculate a distribution D used to guide how to specify different user simulators in the RL training to train the system agent S. This distribution D is designed to *assign lower weights to user simulators that the system agent S already performs well and higher weights to those that S performs not very well*. The implementation of MUST with the *modified* UCB1 algorithm is presented in Algorithm 1. We use T to denote the total number of dialogues that will be sampled in the whole training process. In the first T_0 dialogues, we use a uniform distribution to sample these user simulators to train S. After that, we let the system agent S interact d times with each simulator U_j to obtain its success rate $\bar{X}_j$ and use these K success rates to initiate the distribution D (lines 2-8). In the following steps, we specify different user simulators to train S by the distribution D (lines 10-12). And we will evaluate the RL model S every e episodes and update the distribution D with the new success rates (lines 14-19). The hyperparameter of s is used to calculate a baseline and control the smoothness of distribution D. If s is larger, D is sharper.

Algorithm 1: Implementing MUST with the *modified* UCB1 algorithm

1 Input: Initiate the values of hyperparameters T, T_0, e, d, s;
2 **for** $t = 0, ..., T_0$ **do**
3 Sample a simulator $U_j, j \in \{1...K\}$ with a uniform distribution;
4 Let the system agent S interact with U_j to have a new dialogue;
5 Use the reward obtained for the dialogue to update S with a RL algorithm;
6 Let S interact d times with each simulator U_j to calculate its success rate $\bar{X}_j$;
7 Calculate a baseline with these success rates and a scalar s, $b = \min(\bar{X}_1, ..., \bar{X}_K) * s$;
8 Initiate the distribution $D = \{d_1, ..., d_K\}, d_k = \frac{\frac{1}{\bar{X}_j - b}}{\sum_{j=1}^{K} \frac{1}{\bar{X}_j - b}}$.
9 **for** $t = T_0 + 1, ..., T$ **do**
10 **if** $t\%e! = 0$ **then**
11 Sample a simulator $U_j, j \in \{1...K\}$ from the distribution D;
12 Let the system agent S interact with U_j to have a new dialogue and use the obtained reward to update S;
13 **else**
14 **for** j = $1, ..., K$ **do**
15 Let S interact d times with the simulator U_j to recalculate its success rate $\bar{X}_j$;
16 calculate $\hat{x}_j = \bar{X}_j + \sqrt{\frac{2 \ln t}{T_{j,t}}}$;
17 ***Recalculate the baseline*** $b_t = \min(\hat{x}_1, ..., \hat{x}_K) * s$;
18 ***Calculate*** $\hat{x}_j = 1/(\hat{x}_j - b_t)$;
19 ***Update the distribution***: $D = \{d_1, ..., d_K\}$, $\mathrm{d}_k = \frac{\hat{x}_k}{\sum_{j=1}^{K} \hat{x}_k}$;
20 Output: The dialogue system S.



(a) The details of the first two-turn interactions between a system agent and our U-GPT.

hello! what can i help you? <eos_resp> [eos_constraint] [eos_book] [eos_recommend] [eos_select] [eos_request] <eos_nlu> [info] food venetian pricerange expensive area centre [request] [book] time 12:00 day wednesday people 4 <eos_goal> [inform_type] food area <eos_pol> i am looking at a place to eat that serves venetian food in the centre. <eos_utt> unfortunately, i do not see any restaurants that serve venetian in the centre of town. would you like to try a different area or type of cuisine? <eos_resp> nooffer [eos_constraint] [eos_book] [eos_recommend] [eos_select] [eos_request] <eos_nlu> [info] food chinese pricerange expensive area centre [request] [book] time 12:00 day wednesday people 4 <eos_goal> [inform_type_change] food <eos_pol> Do you have any [value_food] restaurants ? <eos_utt>

(b) An example of the model input for training U-GPT.

Figure 2: The overview of our U-GPT which consists of Natural Language Understanding (NLU), Goal Generator, Dialog Policy Learning (POL), and Natural Language Generation (NLG) and uses the auto-regressive language model GPT to understand the system inputs, generate the user actions and the user utterances given the dialogue context and the user goals sequentially in an end-to-end manner. (a) gives a detailed description of the first two-turn interactions between a system agent and our U-GPT. For training U-GPT, we need to convert the dialogue context and all annotations to sequences of tokens. (b) presents the training example of the first two-turn dialogues in (a).

3.3 Modeling User Simulator with GPT

In this section, we illustrate how our U-GPT models the user simulator based on GPT. And We will demonstrate that it is a better choice if we want to train a single user simulator or add a new simulator into MUST in later experiments (Section 4.4.2).

User Simulator. If we treat a user simulator as a dialog agent, we can use the same framework as the dialog system to build it. However, their roles are different. The user agent has a goal describing a target entity (e.g., a restaurant at a specific location), and should express its goal completely in an organized way by interacting with the system (Takanobu et al., 2020). For the system agent, it does not know the user's goal at the beginning and should gradually understand the user's utterances, query the database to find entities, and provide useful information to see if accomplishing the user's task. Since only the system can access the database, the user does not know if its goal can be satisfied. Once the database result returned by the system agent is empty, the user agent should learn to compromise and change its goal. Therefore, the user agent has another module called Goal Generator (Kreyssig et al., 2018), which is responsible for initiating a goal or generating a new goal.

As Figure 2(a) shown, our U-GPT consists of four modules, which are Natural Language Understanding (NLU), Goal Generator, Dialog Policy Learning (POL), and Natural Language Generation (NLG). Dialogues consist of multiple turns. In the first turn $t = 0$, U-GPT (1) first outputs its NLU results N_0 by understanding the system input S_0, and (3) decide its actions A_0 which is a list of pairs: (action_type, slot_name) based on (2) its initial goal G_0 and $\{S_0, N_0\}$. U-GPT then (4) conditions on $\{S_0, N_0, G_0, A_0\}$ to generate the delexicalized utterance U_0. The generated placeholders in U_0 will be filled using the corresponding slot values in the goal G_0. When the conversation proceeds to turn t, U-GPT (1) generates the NLU results N_t based on all of previous dialogue history and generated outputs $\{C_0, \ldots, C_{t-1}, S_t\}$, here $C_i = [S_i, N_i, G_i, A_i, U_i]$. If there has "nooffer" intent in N_t representing that no entities could satisfy current constraints, then (2) Goal Generator should generate a new goal G_t. Then U-GPT will continue to (3) generate the user acts A_t

and (4) generate delexicalized utterance U_t conditioned on $\{C_0, \ldots, C_{t-1}, S_t, N_t, G_t\}$ sequentially. We should notice that the user utterances occurred in the history context should be lexicalized because they contain important information.

Figure 2(b) shows an example of training sequence which consists of the concatenation $x = [C_0, C_1]$. In order to leverage GPT, we need to convert the generated outputs $\{N_i, G_i, A_i, U_i\}$ to sequences of tokens resembling a text. And we introduce delimiter tokens *<eos_resp>*, *<eos_nlu>*, *<eos_goal>*, *<eos_pol>*, *<eos_utt>* to signal the ending of sequence representations of different modules. For the NLU results N_t, we use five categories: "inform", "request", "book inform", "select", "recommend" same as Shi et al. (2019) to represent them. And we also introduce five tokens *[eos_constraint], [eos_book], [eos_recommend], [eos_select], [eos_request]* to record different information. All of these tokens and the intents of user actions will be added to the vocabulary of GPT as additional special tokens. For training U-GPT, we use the same training objective as GPT which is to maximize the following likelihood:

$$L(U) = \sum_i \log P(u_i | u_{i-k}, ..., u_{i-1}; \Theta),$$

$$\forall\, u_i \in \{S_0, N_0, G_0, A_0, U_0, ..., A_t, U_t\},$$

where k is the size of the context window, and the conditional probability P is parameterized with Θ.

4 Experiments

Our experiments can be divided into two parts. In the first part, we train a system agent S-MUST with the proposed MUST and compare its performance with other system agents trained by a single simulator to prove that we can obtain a better system agent with MUST. In the second part, we will prove that our GPT-based user simulator can outperform previous RNN-based simulators through direct and indirect evaluations.

4.1 Baselines

There are six user simulators provided by Shi et al. (2019), which are Agenda-Template (U-AgenT), Agenda-Retrieval (U-AgenR), Agenda-Generation (U-AgenG), RNN-Template (U-RNNT), RNN-Retrieval (U-RNNR), RNN-End2End (U-RNN) [2] trained with different dialog planning and generation methods.

[2]Here we rename these six simulators for clarifying the role of agents.

U-AgenT, U-AgenR, U-AgenG. The DM manager modules of U-AgenT, U-AgenR, and U-AgenG are rule-based methods. For the NLG module, these three simulators are respectively using the template, retrieval, and generation methods.

U-RNNT, U-RNNR, U-RNN. The DM manager modules of U-RNNT, U-RNNR use Sequicity (Lei et al., 2018) as its backbone which is an RNN-based seq2seq model with copy mechanism. For the NLG module, they are using the template, retrieval methods respectively. U-RNN uses Sequicity as its backbone in an end-to-end manner.

These user simulators will be used to train the system agent S-MUST by our proposed MUST with the *modified* UCB1 algorithm. Any system agent trained by a single user simulator should be the baseline of S-MUST. Because our U-GPT is an end-to-end model based on GPT, we will use U-RNN as the baseline of U-GPT.

4.2 Dataset and Evaluation Measures

MultiWOZ Restaurant Domain Dataset. The original task in MultiWOZ (Budzianowski et al., 2018) was to model the system response. Shi et al. (2019) annotate the user intents and the user-side dialog acts in the restaurant domain of MultiWOZ to build user simulators, which has a total of 1,310 dialogues.

Simulated Agenda Dataset. We simulated 2,000 dialogues from each rule-based simulator U-AgenT, U-AgenR, U-AgenG, and their corresponding system agents respectively, and processed these dialogues to have the same annotation format as the MultiWOZ restaurant domain dataset.

Evaluation Measures. For evaluations on dialogue systems, we report the average success rate of them interacting with different user simulators. The success rate between a pair of a user simulator and a system agent is calculated by sampling 200 dialogues between them in later experiments. To evaluate user simulators, we adopt both **indirect** evaluations and **direct** evaluations as in Shi et al. (2019). In fact, we can also evaluate a user simulator indirectly using the average success rate of the system agent trained by this simulator. It is called cross-model evaluation (Schatzmann and Young, 2009) which assumes a strategy learned with a good user model still performs well when tested on poor user models. It can indirectly evaluate the behavior diversity generated by user simula-

System\User	U-AgenT	U-AgenR	U-AgenG	U-RNNT	U-RNNR	U-RNN	U-GPT$_{SL}$	U-GPT$_{IL}$	Avg. ↑	Std. ↓
S-AgenT	97.5	54.0	72.5	98.5	92.5	77.0	78.0	89.0	82.4	14.1
S-AgenR	96.0	90.0	97.5	98.5	97.5	82.0	80.5	96.0	92.3	6.8
S-AgenG	79.0	78.5	95.0	98.5	96.5	81.5	79.0	91.0	87.4	8.2
S-RNNT	30.5	23.0	35.5	99.0	97.5	84.0	75.5	66.0	63.9	28.5
S-RNNR	30.0	23.0	30.0	96.5	93.5	70.5	68.5	56.0	58.5	26.9
S-RNN	20.0	23.5	20.0	73.0	63.0	77.0	56.5	45.0	47.3	22.2
S-GPT$_{SL}$	60.5	51.5	59.5	97.0	94.0	92.0	82.0	84.5	77.6	16.7
S-GPT$_{IL}$	97.5	83.5	97.5	94.5	94.0	82.5	80.5	96.5	90.8	6.8
S-MUST	97.5	89.5	96.5	97.0	97.5	90.0	82.5	96.0	**93.3**	**5.1**

Table 1: Success rates of the system agents tested against various user simulators. Each column represents one user simulator, each row represents one RL system trained with a specific simulator, e.g. S-AgenT means the RL system trained with U-AgenT. Each entry shows the success rate of 200 dialogues obtained by having the user simulator interact with the system agent.

tors. For direct evaluations, we adopt six evaluation measures for the automatic evaluation: average utterance length, vocabulary size, Dist-1, Dist-2 (Li et al., 2016) and Entropy (Zhang et al., 2018). We also ask human users to rate the simulated dialogues[3] to assess the user simulators directly. We use five same metrics as (Shi et al., 2019) which are Fluency, Coherence, Goal Adherence, Diversity, and an Overall quality to assess the behaviors of user simulators from multiple aspects.

4.3 Implementations

U-GPT$_{IL}$. A simple method to utilize multiple user simulators is learning a new simulator with dialogue sessions collected from these user simulators and training the system agent with this new simulator by Imitation Learning (IL). U-GPT$_{IL}$ is first pre-trained on the simulated agenda dataset which has a total of 6,000 dialogues. Then we sample 1,400 dialogues from the simulated agenda dataset and merge them with 1,310 MultiWOZ restaurant domain dialogues to continue to fine-tune our U-GPT. We denote the system agent trained by U-GPT$_{IL}$ as S-GPT$_{IL}$ and use it as another baseline of S-MUST.

U-GPT$_{SL}$[4]. It is also pre-trained on the simulated agenda dataset and fine-tuned on the 1,310 MultiWOZ restaurant domain dataset with our U-GPT.

S-MUST. Six user simulators provided by Shi et al. (2019) have very different behaviors according to the reported experimental results, therefore they become a good choice to implement MUST. Because the system agents trained by U-AgenG, U-RNNR, U-RNN have no advantages over U-AgenT, U-AgenR, U-RNNT, we decide to use U-AgenT, U-AgenR, U-RNNT, and U-GPT$_{SL}$ to train S-MUST. All system agents trained by user simulators with RL have the same architecture described in Shi et al. (2019). The RL algorithm and the reward used are also the same for a fair comparison.

U-GPT$_{SL\text{-}RNN}$[5]. Because the implementation of U-RNN mainly consists of NLU and NLG, we remove the POL module from U-GPT and use the same annotated data as U-RNN to fine-tune it to compare our U-GPT with the RNN-based methods fairly. And we name it as U-GPT$_{SL\text{-}RNN}$.

User Simulators	NLU	DM	NLG
U-AgenT	RNN	Agenda	Template
U-AgenR	RNN	Agenda	Retrieval
U-AgenG	RNN	Agenda	Generation
U-RNNT	RNN		Template
U-RNNR	RNN		Retrieval
U-RNN	RNN (NLU + NLG)		
U-GPT$_{SL\text{-}RNN}$	Transformer (NLU + NLG)		
U-GPT$_{SL}$	Transformer (NLU + POL + NLG)		
U-GPT$_{IL}$	Transformer (NLU + POL + NLG)		

Table 2: The architectures of all user simulators.

We give an overview (Table 2) of the architectures of all user simulators that occurred in this paper to illustrate the differences between them. The dialogue management (DM) module contains two sub-modules, the DST and POL modules.

4.4 Results and Analysis

4.4.1 Evaluations on Dialogue Systems

The last column and the second last column in Table 1 show that S-MUST achieves the highest average success rate (93.3) interacting with eight user simulators and these eight success rates have the

[3]The system agent for simulating dialogues is a third-party system provided by (Shi et al., 2019) which was built based on hand-crafted rules.

[4]SL is short for Supervised Learning.

[5]We provide the implementation details of all user simulators that are based on U-GPT in appendix A.1.

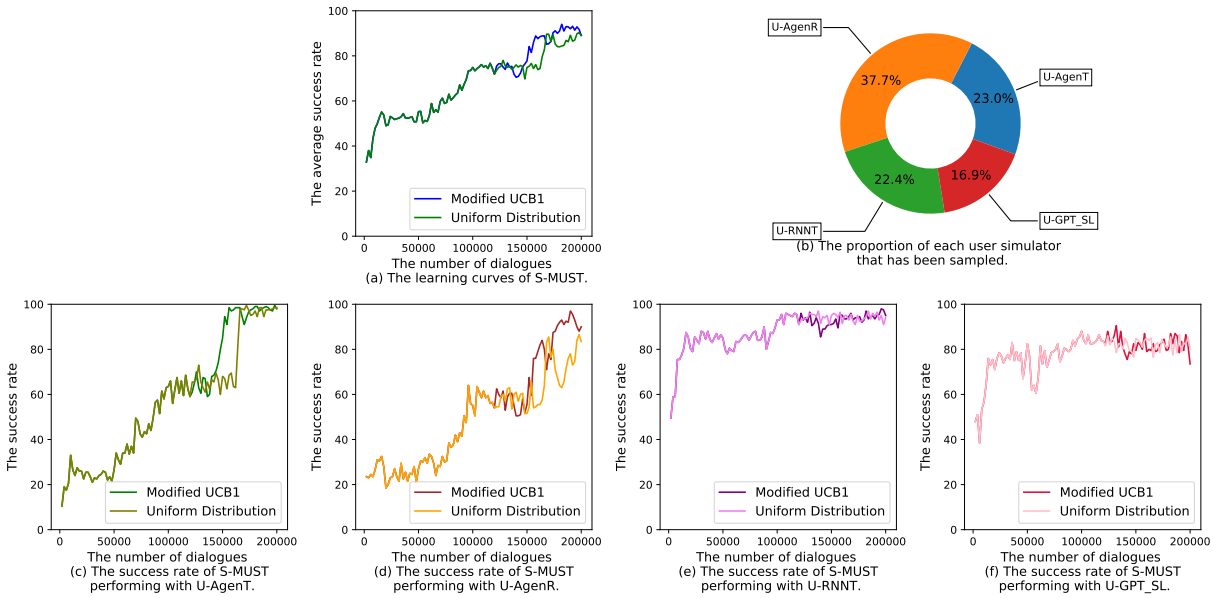


Figure 3: The analysis of training S-MUST with four good user simulators of U-AgenT, U-AgenR, U-RNNT, U-GPT$_{\text{SL}}$ by the *modified* UCB1 algorithm and the uniform distribution. (a) shows the learning curves of S-MUST in the RL training. In the first 120,000 episodes in both situations, we use the uniform distribution to sample these simulators. (b) gives an overview of the proportion of each user simulator that has been sampled by the *modified* UCB1 algorithm in the last 80,000 episodes. We further analyze the performance of S-MUST interacting with each user simulator in the RL training and the details are shown in (c)-(f).

smallest standard deviation. Moreover, S-MUST improves over S-GPT$_{\text{IL}}$ by 2.5 points in the average success rate. We also give an analysis about our training strategy. From Figure 3(a), we can see that training S-MUST with U-AgenT, U-AgenR, U-RNNT, U-GPT$_{\text{SL}}$ by the *modified* UCB1 algorithm converges faster than by the uniform distribution. To understand this result, we further plot the performance of S-MUST interacting with each user simulator in the RL training shown in Figure 3(c)-(f) and find that S-MUST trained by the user simulator U-AgenR converges last. Combining Figure 3(b) which shows us the proportion of each user simulator that has been sampled by the *modified* UCB1 algorithm in the last 80,000 episodes of RL, we know that the *modified* UCB1 algorithm would sample the user simulator whose corresponding system agent is harder to converge more times and this could help the system agent converges faster. The overall experimental results prove that the proposed MUST is powerful and versatile to cope with a variety of user simulators to train a system agent effectively by the *modified* UCB1 algorithm.

4.4.2 Evaluations on User Simulators

For the indirect evaluation of user simulators, we use the average success rates of the system agents trained by them. The average success rates are calculated by letting each RL system interact with all eight user simulators that same as the eight user simulators used in Table 1. Table 3 shows the cross-model evaluation results. We can see that the average success rate of S-GPT$_{\text{SL-RNN}}$ far exceeds that of S-RNN. This represents that our U-GPT can perform more various behaviors than U-RNN to help the system agent explore the dialogue states.

System	Avg. ↑	Std. ↓
S-RNN	47.3	**22.2**
S-GPT$_{\text{SL-RNN}}$	**67.1**	24.1

Table 3: The cross-model evaluation results of user simulators. We put the detailed results in the table 7.

User Simulators	Utt ↑	Vocab ↑	DIST-1 ↑	DIST-2 ↑	ENT-4 ↑
U-RNN	10.95	205	**1.17%**	3.14%	4.98
U-GPT$_{\text{SL-RNN}}$	**14.00**	**262**	1.13%	**3.53%**	**5.62**

Table 4: Automatic evaluation metrics of user simulators include average utterance length (Utt), vocabulary size (Vocab), distinct-n (DIST-n) and entropy (ENT-n).

The automatic evaluation results in Table 4 and the language diversity score (Hu.Div in Table 5) evaluated by humans directly tell us that the dialogues generated by our U-GPT are more diverse than by U-RNN in the level of language. And the remaining metrics in Table 5 tell us that our U-GPT

User Simulators	Hu.Fl ↑	Hu.Co ↑	Hu.Go ↑	Hu.Div ↑	Hu.All ↑
U-RNN	2.80	2.30	2.86	2.74	2.30
U-GPT$_{\text{SL-RNN}}$	**4.10**	**4.04**	**4.30**	**3.70**	**4.00**

Table 5: Human evaluation results of user simulators. The metrics include sentence fluency (Hu.Fl), coherence (Hu.Co), goal adherence (Hu.Go), language diversity (Hu.Div) and an overall score (Hu.All).

can generate more fluent dialogues and is more like a real human that could express its goal completely. We give some examples in the appendix D.1.

4.5 Discussion

We also compare our U-GPT$_{\text{SL-RNN}}$ with U-RNNT and U-RNNR in the appendix B. Our U-GPT$_{\text{SL-RNN}}$ can generate more diverse language and more various behaviors than U-RNNT according to the cross-model evaluation and automatic evaluation. However, U-RNNT performs better than our U-GPT$_{\text{SL-RNN}}$ in the overall performance according to the human evaluation. We think it is because the NLG module of U-RNNT is the template-based method and the generated dialogues from them are easy for the third-party system to interact with. The user utterances generated by U-RNNR are retrieved from a corpus that is written by real humans. We think that the dialogues are written by humans usually have higher language diversity than the dialogues generated by models. Even though the dialogues generated by U-RNNR are more diverse, the dialogues generated by our U-GPT$_{\text{SL-RNN}}$ are more fluent and coherent. Also, the cross-model evaluation results show that U-GPT$_{\text{SL-RNN}}$ can help to learn a more robust system agent than U-RNNR, but the Hu.All score in the human evaluation gives an opposite result. The gap between cross-model evaluation metrics, automatic metrics, and real human evaluation represents that the evaluation of the user simulators is challenging.

5 Related Work

The ABUS (Schatzmann et al., 2007; Schatzmann and Young, 2009) represents the user state as a stack of user actions, called the agenda. The mechanism that generates the user response and updates the agenda does not require any data, though it can be improved using data. (Asri et al., 2016) modeled user simulation as a sequence-to-sequence task and user behavior is learned entirely from data but ignores the goal changes. (Kreyssig et al., 2018) introduces a Neural User Simulator (NUS) consisting of a goal generator that generates its own goal and possibly changes it during a dialogue, a feature extractor, and a neural network-based sequence-to-sequence model (Sutskever et al., 2014). Even though these simulators can generate successful conversations, they lack the necessary mechanisms to produce diverse responses. (Gur et al., 2018) develop a hierarchical seq2seq user simulator (HUS) to allow the model to capture undiscovered parts of the user goal without an explicit DST module and several variants by utilizing a latent variable model to inject random variations into user responses to generate diverse user responses.

A work similar to us in building the user simulator is (Mohapatra et al., 2021). It uses GPT-2 to build a user bot and uses the data collected from this user bot and an agent bot to achieve improvements in both low-resource settings as well as in the overall task performance of the agent bot. However, their user simulator mainly consists of Response Generator and Response Selector modules in a different way from us and Response Generator generates the response conditioned on the goal instruction expressed in a natural language, not a structured goal expressed in slot types, slots, and values. (Shi et al., 2019) implements six user simulators in both the ABUS and the RNN-based User Simulators and presents a comprehensive evaluation framework for user simulator study. These six simulators have very different behaviors, however, they do not use them together to train a more robust system agent. Our framework is the first work to consider training the system agent with different user simulators.

6 Conclusion

In this paper, we propose a framework named MUST to improve the system agent by using different user simulators simultaneously and a new method named U-GPT which leverages GPT to improve the user simulator. we conduct automatic evaluation, cross-evaluation, and human assessments on them. The experiment results demonstrate that our proposed MUST can significantly improve the robustness of the system agent upon the baseline methods and our U-GPT can generate more diverse language and more various behaviors than RNN-based user simulators to help the system agent explore the dialogue states. In the future, we plan to apply our proposed methods and training techniques to multi-domain scenarios.

References

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Izzeddin Gur, Dilek Hakkani-Tur, Gokhan Tur, and Pararth Shah. 2018. User modeling for task oriented dialogues.

Donghoon Ham, Jeong-Gwan Lee, Youngsoo Jang, and Kee-Eung Kim. 2020. End-to-end neural pipeline for goal-oriented dialogue systems using GPT-2. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 583–592, Online. Association for Computational Linguistics.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue. In *Advances in Neural Information Processing Systems*, volume 33, pages 20179–20191. Curran Associates, Inc.

Florian Kreyssig, Iñigo Casanueva, Paweł Budzianowski, and Milica Gašić. 2018. 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. Association for Computational Linguistics.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1437–1447, Melbourne, Australia. Association for Computational Linguistics.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Austin, Texas. Association for Computational Linguistics.

Biswesh Mohapatra, Gaurav Pandey, Danish Contractor, and Sachindra Joshi. 2021. Simulated chats for building dialog systems: Learning to generate conversations from instructions.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. 2021. Soloist: Building task bots at scale with transfer learning and machine teaching.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. 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 Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

Jost Schatzmann and Steve Young. 2009. The hidden agenda user simulation model. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):733–747.

Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. 2019. How to build user simulators to train RL-based dialog systems. 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 1990–2000, Hong Kong, China. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks.

Ryuichi Takanobu, Runze Liang, and Minlie Huang. 2020. Multi-agent task-oriented dialog policy learning with role-aware reward decomposition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 625–638, Online. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. 2018. Generating informative and diverse conversational responses via adversarial information maximization. In *NeurIPS*.

A Implementation Details

A.1 Training user simulators

We implement our GPT-based user simulators with DistilGPT2 (Sanh et al., 2020), a distilled version of GPT-2 by HuggingFace's Transformers (Wolf et al., 2020). We select the best performing models on validation set through hyperparameters search of learning rate and batch size. The best models was fine-tuned with batch size of 64 and the learning rate of 5e-3 over the corresponding dataset. And we use the greedy decoding strategy for U-GPT generating word-tokens in the inference phrase.

A.2 Training the S-MUST

Parameters	
T	200,000
T_0	120,000
e	2,000
d	200
s	0.75

Table 6: The hyperparameters used for training the S-MUST.

The hyperparameters used in our *modified* UCB1 algorithm to train the S-MUST are listed in the Table 6.

B The comparisons between U-RNNT, U-RNNR and our U-GPT$_{\text{SL-RNN}}$

We also compare our U-GPT$_{\text{SL-RNN}}$ with U-RNNT and U-RNNR.

Compared with U-RNNT, the cross-model evaluation results (Table 7), the automatic evaluation results (Table 8) and the Hu.Div score in the human evaluation results (Table 9) show that our U-GPT$_{\text{SL-RNN}}$ performs better. However, as Table 9 shown, U-RNNT performs better than our U-GPT$_{\text{SL-RNN}}$ in the overall performance according to the human evaluation. We think it is because the third-party system also has an impact on the generated dialogues and the NLG module of U-RNNT is the template-based method which leads to that the generated dialogues from U-RNNT are easy for the third-party system to understand and interact with.

The automatic evaluation results (Table 8) and the Hu.Div score in the human evaluation results (Table 9) show that U-RNNR can generate more diverse language than our U-GPT$_{\text{SL-RNN}}$. We think it is because the user utterances generated

System \User	U-AgenT	U-AgenR	U-AgenG	U-RNNT	U-RNNR	U-RNN	U-GPT$_{\text{SL}}$	U-GPT$_{\text{IL}}$	Avg. ↑	Std. ↓
S-RNNT	30.5	23.0	35.5	99.0	97.5	84.0	75.5	66.0	63.9	28.5
S-RNNR	30.0	23.0	30.0	96.5	93.5	70.5	68.5	56.0	58.5	26.7
S-RNN	20.0	23.5	20.0	73.0	63.0	77.0	56.5	45.0	47.3	**22.2**
S-GPT$_{\text{SL-RNN}}$	36.5	38.0	42.0	95.5	94.0	89.0	80.5	61.0	**67.1**	24.1

Table 7: Cross study results. Each entry shows the success rate obtained by having the user simulator interacting with the RL system for 200 times.

User Simulators	Utt ↑	Vocab ↑	DIST-1 ↑	DIST-2 ↑	ENT-4 ↑
U-RNNT	9.83	192	0.77%	1.51%	4.24
U-RNNR	11.06	346	2.45%	9.59%	6.59
U-GPT$_{\text{SL-RNN}}$	14.00	262	1.13%	3.53%	5.62

Table 8: Automatic evaluation results of U-RNNT, U-RNNR and U-GPT$_{\text{SL-RNN}}$. The metrics include average utterance length (Utt), vocabulary size (Vocab), distinct-n (DIST-n) and entropy (ENT-n).

User Simulators	Hu.Fl ↑	Hu.Co ↑	Hu.Go ↑	Hu.Div ↑	Hu.All ↑
U-RNNT	4.60	4.68	4.96	3.34	4.70
U-RNNR	3.92	3.88	4.72	3.94	4.16
U-GPT$_{\text{SL-RNN}}$	4.10	4.04	4.30	3.70	4.00

Table 9: Human evaluation results of U-RNNT, U-RNNR and U-GPT$_{\text{SL-RNN}}$. The metrics include sentence fluency (Hu.Fl), coherence (Hu.Co), goal adherence (Hu.Go), language diversity (Hu.Div) and an overall score (Hu.All).

by U-RNNR are retrieved from a corpus that is written by real humans and the sentences written by humans are usually more diverse than the sentences generated by generative models. Even though the dialogues generated by U-RNNR are more diverse, the dialogues generated by our U-GPT$_{\text{SL-RNN}}$ are more fluent and coherent. Also, the cross-model evaluation results (Table 7) show that U-GPT$_{\text{SL-RNN}}$ can help to learn a more robust system agent than U-RNNR, but the Hu.All score in the human evaluation (Table 9) gives an opposite result.

C Comparision between our U-GPT and the ABUS

U-GPT$_{\text{IL-AgenR}}$. Because building simulators using rule-based approaches is independent with the data set, we collect 2,000 dialogues sampled from interactions between U-AgenR and S-AgenR to fine-tune our U-GPT for comparing with U-AgenT, U-AgenR, U-AgenG. And we name this imitated user simulator as U-GPT$_{\text{IL-AgenR}}$.

We can see that the average success rate of S-GPT$_{\text{IL-AgenR}}$ is higher than S-AgenT, S-AgenG at least 4.9 points from Table 10. The automatic evaluation (Table 11) and the Hu.Div score in the human evaluation of Table 12 represent that U-AgenR has the highest language diversity. We think it is the reason why the average success rate of S-GPT$_{\text{IL-AgenR}}$ is slightly lower than S-AgenR. It is easily understood because U-GPT$_{\text{IL-AgenR}}$ is trained on the simulated dialogues from U-AgenR using IL and we know that it is more difficult to imitate when the training data is more diverse. But the difference of 0.9 points between S-AgenR and S-GPT$_{\text{IL-AgenR}}$ proves that our U-GPT has a good ability on learning from past experiences. The NLG module of U-AgenR uses Retrieval method. If we ignore it, U-GPT$_{\text{IL-AgenR}}$ will have the highest language diversity. Therefore, we can claim that our U-GPT can generate various user behaviors and diverse language. Also, U-AgenT performs best in the overall performance on the human evaluation (Table 12). And the performance of our U-GPT$_{\text{IL-AgenR}}$ is much better than U-AgenG. We also give some examples in the appendix D.2.

D Case Study

D.1 U-RNN vs U-GPT$_{\text{SL-RNN}}$

We give two examples to show the superiority of our U-GPT. Compared with U-GPT$_{\text{SL-RNN}}$, U-RNN is more likely to ignore the system response to state its goal when interacting with the system agent. As shown in Table 13, we highlight the user utterances generated not well by U-RNN in red color. U-RNN first ignore the request about the "food" type asked by the system agent and give its "book" information about the goal directly. And then U-RNN repeats three times to state that the restaurant it is looking for should be in the expensive price range. As a result, the dialogue generated by U-RNN fail as the user does not inform food type and book time. For the same goal, our U-GPT$_{\text{SL-RNN}}$ completes its goal in four turns and the generated dialogue are fluent and coherent.

System \User	U-AgenT	U-AgenR	U-AgenG	U-RNNT	U-RNNR	U-RNN	U-GPT$_{\text{SL}}$	U-GPT$_{\text{IL}}$	Avg. ↑	Std. ↓
S-AgenT	97.5	54.0	72.5	98.5	92.5	77.0	78.0	89.0	82.4	14.1
S-AgenR	96.0	90.0	97.5	98.5	97.5	82.0	80.5	96.0	**92.3**	6.8
S-AgenG	79.0	78.5	95.0	98.5	96.5	81.5	79.0	91.0	87.4	8.2
S-GPT$_{\text{IL-AgenR}}$ (ours)	94.0	85.0	94.5	97.0	98.0	85.0	81.0	96.5	91.4	**6.2**

Table 10: Cross study results on U-GPT$_{\text{IL-AgenR}}$ and the three ABUS. Each entry shows the success rate obtained by having the user simulator interacting with the RL system for 200 times.

User Simulators	Utt ↑	Vocab ↑	DIST-1 ↑	DIST-2 ↑	ENT-4 ↑
U-AgenT	9.65	180	0.76%	1.61%	4.51
U-AgenR	11.61	383	2.24%	10.06%	7.05
U-AgenG	8.07	159	0.59%	1.17%	4.15
U-GPT$_{\text{IL-AgenR}}$	9.70	298	1.15%	4.60%	5.47

Table 11: Automatic evaluation results of U-GPT$_{\text{IL-AgenR}}$ and the three ABUS. The metrics include average utterance length (Utt), vocabulary size (Vocab), distinct-n (DIST-n) and entropy (ENT-n).

User Simulators	Hu.Fl ↑	Hu.Co ↑	Hu.Go ↑	Hu.Div ↑	Hu.All ↑
U-AgenT	4.60	4.60	4.96	3.58	4.48
U-AgenR	3.88	3.56	4.62	4.02	3.78
U-AgenG	3.40	3.36	4.26	3.40	3.54
U-GPT$_{\text{IL-AgenR}}$	4.08	3.36	4.20	3.82	3.62

Table 12: Human evaluation results of U-GPT$_{\text{IL-AgenR}}$ and the three ABUS. The metrics include sentence fluency (Hu.Fl), coherence (Hu.Co), goal adherence (Hu.Go), language diversity (Hu.Div) and an overall score (Hu.All).

D.2 U-AgenR vs U-GPT$_{\text{IL-AgenR}}$

The NLG module of U-AgenR uses Retrieval method. As shown in Table 14, U-AgenR first wants to find a moderate restaurant. However, the retrieved template is not proper as the beginning of the conversation. And there has no constraint about the location of the restaurant, but U-AgenR uses the word "dontcare" to fill the retrieved template directly which leads to an influent user utterance. Then U-AgenR is satisfied with the recommendation of the system agent and decides to book a table on Tuesday. Also the retrieved template used to convey this user action is so weird. To this end, even though U-AgenR can complete its goal, but the fluency and coherence of the generated dialogue is poor. The dialogue generated by our U-GPT$_{\text{IL-AgenR}}$ are more fluent and coherent.

Goal	{'id': 'SNG02148.json', 'info': {'food': 'indian', 'pricerange': 'expensive', 'area': 'west'}, 'book': {'time': '18:00', 'day': 'tuesday', 'people': '8'}}		
Agent	Sentence	**Agent**	Sentence
U-RNN	I would like to find an expensive restaurant in the west part of town.	U-GPT$_{\text{SL-RNN}}$	I am looking for an expensive place to dine. i want it to be on the west side of town.
System	Do you have a preference on the food?	System	Do you have a preference on the food?
U-RNN	I would like to book a table for 8 people on tuesday.	U-GPT$_{\text{SL-RNN}}$	Yes, I would prefer indian food if necessary.
System	Tandoori palace is a indian restaurant on the west side of town. It's in the expensive price range. Is there anything else I can help you with?	System	Tandoori palace is an indian restaurant on the west side of town. It's in the expensive price range. Is there anything else I can help you with?
U-RNN	That is the restaurant to be in the expensive price range.	U-GPT$_{\text{SL-RNN}}$	Yes, that sounds good. could you reserve a table for 8 at 18:00 on tuesday?
System	Cocum is a indian restaurant on the west side of town. It's in the expensive price range. Is there anything else I can help you with?	System	I have booked the reservation for you. And you reference number is ABC.
U-RNN	That is the restaurant to be in the expensive price range.	U-GPT$_{\text{SL-RNN}}$	Great, that is all i will be needing today. thank you for your help.
System	Rice boat is a indian restaurant on the west side of town. It's in the expensive price range. Is there anything else I can help you with?		
U-RNN	That is the restaurant to be in the expensive price range.		
System	Maharajah tandoori restaurant is an indian restaurant on the west side of town. It's in the expensive price range. Is there anything else I can help you with?		
U-RNN	That is all i needed. Thank you for your help.		

Table 13: An example of dialogue session comparison between U-RNN and U-GPT$_{\text{SL-RNN}}$.

Goal	{'id': 'SSNG0179.json', 'info': {'food': 'turkish', 'pricerange': 'moderate'}, 'book': {'people': '7', 'day': 'tuesday', 'time': '12:30'}}		
Agent	Sentence	**Agent**	Sentence
U-AgenR	A moderate restaurant please	U-GPT$_{\text{IL-AgenR}}$	I am looking for a moderately priced restaurant.
System	Do you have a preference on the food and area?	System	Do you have a preference on the food and area?
U-AgenR	I would like some good turkish food . near the dontcare of town would be great.	U-GPT$_{\text{IL-AgenR}}$	I am looking for an inexpensive turkish restaurant.
System	Anatolia is a turkish restaurant on the centre side of town. It's in the moderate price range. Is there anything else I can help you with?	System	Anatolia is a turkish restaurant on the centre side of town. It's in the moderate price range. Is there anything else I can help you with?
U-AgenR	On tuesday please	U-GPT$_{\text{IL-AgenR}}$	There will be 7 of us and we would like to eat at 12:30.
System	Could you please tell me how many people and what time you would like for the reservation?	System	Could you please tell me what day you would like for the reservation?
U-AgenR	I would like a reservation at 12:30 for a party of 7 .	U-GPT$_{\text{IL-AgenR}}$	On tuesday please
System	I have booked the reservation for you. And you reference number is ABC.	System	I have booked the reservation for you. And you reference number is ABC.
U-AgenR	Wonderful , thanks . That s all i need .	U-GPT$_{\text{IL-AgenR}}$	Thank you for your help, good day.

Table 14: An example of dialogue session comparison between U-AgenR and U-GPT$_{\text{IL-AgenR}}$.