# ONE CANNOT STAND FOR EVERYONE! LEVERAGING MULTIPLE USER SIMULATORS TO TRAIN TASK-ORIENTED DIALOGUE SYSTEMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

User simulators are agents designed to imitate human users; recent advances have found that Task-oriented Dialogue (ToD) systems optimized toward a user simulator could better satisfy the need of human users. However, this might result in a sub-optimal ToD system if it is tailored to only one *ad hoc* user simulator, since human users can behave differently. In this paper, we propose a framework called MUST to optimize ToD systems via leveraging **m**ultiple **u**ser **s**imula**t**ors.

The main challenges of MUST fall in 1) how to adaptively specify which user simulator to interact with the ToD system at each optimization step, since the ToD system might be over-fitted to some specific user simulators, and simultaneously under-fitted to some others; 2) how to avoid catastrophic forgetting of the adaption for a simulator that is not selected for several consecutive optimization steps. To tackle these challenges, we formulate MUST as a Multi-armed bandits (MAB) problem and provide a method called $\text{MUST}_{\text{adaptive}}$ that balances *i*) the *boosting adaption* for adaptive interactions between different user simulators and the ToD system and *ii*) the *uniform adaption* to avoid the catastrophic forgetting issue. With both automatic evaluations and human evaluations, our extensive experimental results on the restaurant search task from MultiWOZ show that the dialogue system trained by our proposed MUST achieves a better performance than those trained by any single user simulator. It also has a better generalization ability when testing with unseen user simulators. Moreover, our method $\text{MUST}_{\text{adaptive}}$ is indeed more efficient and effective to leverage multiple user simulators by our visualization analysis.

## 1 INTRODUCTION

Task-oriented dialogue systems aim to help users accomplish their various tasks (e.g., restaurant reservations) through natural language conversations. Training task-oriented dialogue systems in supervised learning (SL) approaches often requires a large amount of expert-labeled dialogues, however collecting these dialogues is usually expensive and time-consuming. Moreover, even with a large amount of dialogue data, some dialogue states may not be explored sufficiently for dialogue systems [1] (Li et al., 2016b). To this end, many researchers try to build user simulators to mimic human users for generating reasonable and natural conversations. By using a user simulator and sampled user goals, we can train the dialogue system from scratch by reinforcement learning (RL) algorithms. Previous works tend to design better user simulator models (Schatzmann et al., 2007; Asri et al., 2016; Gur et al., 2018; Kreyssig et al., 2018; Lin et al., 2021). Especially, Shi et al. (2019) builds various user simulators and analyzes the behavior of each user simulator in the popular restaurant search task from MultiWOZ (Budzianowski et al., 2018).

In real application scenarios, the deployed dialogue system needs to face various types of human users. A single *ad hoc* user simulator can only represent one or a group of users, while other users might be under-represented. Instead of choosing the best-performed one from many dialogue systems trained by different single user simulators, we believe that it is worth trying to train a dialogue system by leveraging all user simulators simultaneously.

[1] We use the dialogue systems to refer to the task-oriented dialogue systems for simplicity in this paper.

In this paper, we propose a framework called MUST to utilize **M**ultiple **U**ser **S**imula**T**ors simultaneously to obtain a better system agent. There exist several simple ways to implement the MUST framework, including a merging strategy, a continual reinforcement learning strategy, and a uniform adaption strategy, denoted as $\text{MUST}_{\text{merging}}$, $\text{MUST}_{\text{CRL}}$, and $\text{MUST}_{\text{uniform}}$ respectively (See Sec. 3.2). However, none of them could effectively tackle the challenges of MUST: 1) how to efficiently leverage multiple user simulators when training the dialogue system since the system might be easily over-fitted to some specific user simulators and simultaneously under-fitted to some others, and 2) it should avoid a catastrophic forgetting issue. Therefore, we first formulate the problem as a Multi-armed bandits (MAB) problem (Auer et al., 2002); similar to the exploration vs exploitation trade-off, specifying multiple user simulators should trade off a boosting adaption (tackling challenge *i*) and uniform adaption (tackling challenge *ii*), see Sec. 4.1 for more details. Then we implement a method called $\text{MUST}_{\text{adaptive}}$ which utilizes an adaptively-updated distribution among all user simulators to sample them in a real-time manner when training the dialogue system.

Extensive experimental results on the restaurant search task from MultiWOZ with both automatic evaluations and human evaluations show that the dialogue system trained by our proposed MUST achieves a better performance than those trained by any single user simulator. It also has a better generalization ability when testing with unseen user simulators and is more robust to the diversity of user simulators. Moreover, our method $\text{MUST}_{\text{adaptive}}$ is indeed more efficient for leveraging multiple user simulators by our visualization analysis.

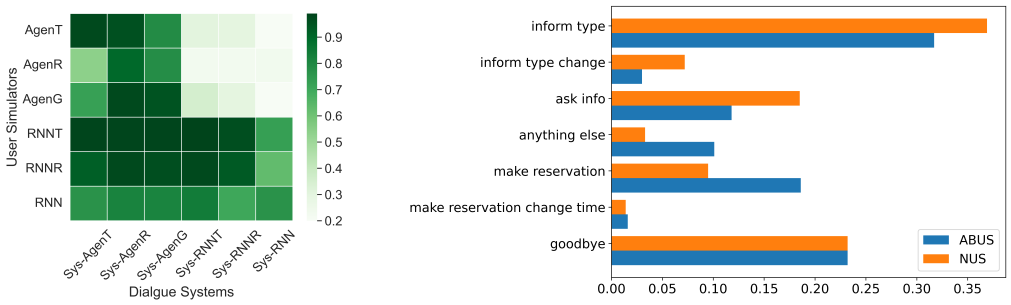
Our contributions are three-fold: (1) To the best of our knowledge, our proposed MUST is the first developed work to improve the dialogue system by using multiple user simulators simultaneously; (2) We design several ways to implement the MUST. Especially, we formulate MUST as a Multi-armed bandits (MAB) problem, based on which we provide a novel method $\text{MUST}_{\text{adaptive}}$; and (3) The results show that dialogue systems trained with MUST consistently outperform those trained with a single user simulator through automatic and human evaluations. Especially, it largely improves the performance of the dialogue system tested on out-of-domain evaluation. Furthermore, The proposed method $\text{MUST}_{\text{adaptive}}$ for MUST convergences faster than $\text{MUST}_{\text{uniform}}$.

## 2 BACKGROUND

**Dialogue System.** Task-oriented dialogue systems aim to help users accomplish various tasks such as restaurant reservations through natural language conversations. Researchers usually divide the task-oriented dialogue systems into four modules (Wen et al., 2017; Ham et al., 2020; Peng et al., 2021): Natural Language Understanding (NLU) (Liu & Lane, 2016) that first comprehends user's intents and extracts the slots-values pairs, Dialog State Tracker (DST) (Williams et al., 2013) that tracks the values of slots, Dialog Policy Learning (POL) (Peng et al., 2017; 2018) that decides the dialog actions, and Natural Language Generation (NLG) (Wen et al., 2015; Peng et al., 2020) that translates the dialog actions into a natural-language form. The DST module and the POL module usually are collectively referred to as the dialogue manager (DM) (Chen et al., 2017). These different modules can be trained independently or jointly in an end-to-end manner (Wen et al., 2017; Liu & Lane, 2018; Ham et al., 2020; Peng et al., 2021; Hosseini-Asl et al., 2020).

**User Simulator.** The user simulator is an agent but also plays a user role. Different from dialogue systems, 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 agent (Takanobu et al., 2020). Therefore, besides the modules of NLU, DM, and NLG like dialogue systems, the user agent should have another module called Goal Generator (Kreyssig et al., 2018), which is responsible for generating the user's goal. Building a user simulator usually could use an agenda-based approach (Schatzmann et al., 2007; Schatzmann & Young, 2009) designing handcrafted rules to mimic user behaviors or a model-based approach such as neural networks (Asri et al., 2016; Kreyssig et al., 2018; Gur et al., 2018) learned on a corpus of dialogues.

**Training Dialogue Systems with a User Simulator.** At the beginning of a dialogue, the user agent obtains its initial goal from the Goal Generator and then expresses its goal in natural languages. For the system agent, it does not know the user's goal and it 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




(a) Success rates of different systems.

(b) Dialog act distributions of different user simulators.

Figure 1: (a) is the heat map on the success rates of system agents tested by different user simulators on 200 dialogues. (b) shows the dialog act distributions of Agenda-based User Simulators (ABUS) and Neural networks-based User Simulators (NUS) provided by Shi et al. (2019). There exist seven user dialog acts annotated in the restaurant search task from MultiWOZ, as shown on the Y-axis.

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 with the help of Goal Generator (Kreyssig et al., 2018). When the dialogue ends, the user simulator will reward the system agent according to if the system agent accomplished the task. Then we could use the reward to update the system agent by reinforcement learning algorithms (Tseng et al., 2021).

## 3 MUST: LEVERAGE MULTIPLE USER SIMULATORS

### 3.1 MOTIVATIONS TO LEVERAGE MULTIPLE USER SIMULATORS

**User simulators behave differently.** Shi et al. (2019) implement six user simulators (AgenT, AgenR, AgenG, RNNT, RNNR, RNN [2]) with both agenda-based methods and neural networks-based methods on the popular restaurant search task from MultiWOZ (Budzianowski et al., 2018). From their experiments, we observed that the dialogue systems trained by different user simulators vary in their performances (i.e., the success rates tested by the same user simulators). For example, when interacting with the user simulator of AgenT, the success rates of the system agents trained by Agenda-based user simulators (i.e., AgenT, AgenR, AgenG) are much higher than the system agents trained by RNN-based user simulators (i.e., RNNT, RNNR, RNN), see Fig. 1(a). The reason might be that these user simulators (i.e., with either handcrafted rules or data-driven learning in their DM modules) have different user dialog act distributions [3] (see Fig. 1(b)) which determines the dialogue state space explored by the dialogue system.

**One cannot stand for everyone.** Users might behave differently, one could design different user simulators with specific user dialog act distributions, see Shi et al. (2019). A single user simulator learned on a task-oriented dialogue corpus can just represent one or a group of users, while the dialogue system needs to accomplish tasks from various human users in real scenarios. We argue that it is beneficial to utilize all different user simulators to train the dialogue system. By leveraging multiple user simulators that have different user dialog act distributions, the dialogue systems can explore a larger dialogue state space, which might improve the ability of learned dialogue system.

### 3.2 SOME PRELIMINARY PROPOSALS FOR MUST

We propose a framework to leverage **M**ultiple **U**ser **S**imula**T**ors (called 'MUST'), the core idea of which is to train a better dialogue system by using multiple user simulators. There are several simple ways to implement the MUST framework, including a *merging* strategy denoted as $\text{MUST}_{\text{merging}}$, a

[2]Here we rename the user simulators of SLT, SLR, and SLE in Shi et al. (2019) as RNNT, RNNR, RNN for emphasizing the model structure of their DM modules.

[3]The dialogue policy learning module is essential in both dialogue systems and user simulators. A policy module corresponds to a dialog act distribution since it decides to take which dialog act to respond to the current dialogue state. The user dialog act distribution behind a user simulator determines the diversity of the dialogue state space explored by dialogue system; therefore it might affect the performance of system agent.

Table 1: The comparison of different strategies for leveraging multiple user simulators.

| | dynamic adaption | avoid catastrophic forgetting | efficiency |
|---|---|---|---|
| $\text{MUST}_{\text{merging}}$ | × | × | × |
| $\text{MUST}_{\text{CRL}}$ | × | × | × |
| $\text{MUST}_{\text{uniform}}$ | × | ✓ | × |
| $\text{MUST}_{\text{adaptive}}$ | ✓ | ✓ | ✓ |

*Continual Reinforcement Learning* strategy denoted as $\text{MUST}_{\text{CRL}}$, and a uniform adaption strategy denoted as $\text{MUST}_{\text{uniform}}$.

**(I) MUST**$_{\text{merging}}$ first sample some dialogues from each user simulator and the corresponding dialogue system trained by this simulator. Then it combines the collected dialogues to train a new user simulator for ensembling different user dialog act distributions. Finally, it uses this new user simulator to train the dialogue system by RL algorithms.

**(II) MUST**$_{\text{CRL}}$ [4] treats each user simulator as an independent RL environment. It moves the trained system agent to another environment (i.e., let the system agent interact with another user simulator) if the system converges in the current RL environment.

**(III) MUST**$_{\text{uniform}}$ allows the system agent have chances to interact with all user simulators simultaneously. Different with $\text{MUST}_{\text{CRL}}$, $\text{MUST}_{\text{uniform}}$ puts all user simulators in a single RL environment and adopts the simplest way to specify different user simulators to train the dialogue system, which is to pick a user simulator among all user simulators with a uniform distribution for each iteration in the RL training.

**Challenges to leverage multiple user simulators.** The problem with $\text{MUST}_{\text{merging}}$ is that it becomes difficult to adjust the weights of each user simulator adaptively in the training process. In $\text{MUST}_{\text{merging}}$, the proportions of dialogues from each user simulator are fixed; however, with the fixed proportions of user simulators, some user simulators might be not well-adapted and some might be overfitted. Because $\text{MUST}_{\text{CRL}}$ has a problem of catastrophic forgetting (Khetarpal et al., 2020), it would be sensitive to the order of different user agents interacting with the dialogue system. Like Shi et al. (2019) shows, the system agents trained by different user simulators have different convergence speeds and converged performances. Namely, the system agent might be easily adapted to some user simulators but might be much harder to be adapted to others. A static distribution for user simulator selection (like in $\text{MUST}_{\text{uniform}}$) under our proposed MUST will result in inefficient training since some unnecessary efforts will be costed for easily-adapted user simulators. Overall, the challenging problems under the MUST framework are i) how to efficiently leverage multiple user simulators when training the system agent, and ii) avoiding the catastrophic forgetting issue.

## 4 MUST AS A MULTI-ARMED BANDIT PROBLEM

Under the setting of $\text{MUST}_{\text{uniform}}$, we put multiple user simulators into the same RL environment and allow the dialogue system to learn from all of them. To make its training process more efficient, we first formulate the proposed MUST as a Multi-armed bandit (MAB) problem, see Sec. 4.1. In Sec. 4.2, we propose an adaptively-updated distribution used to replace the uniform distribution for accelerating the MUST training, which is denoted as $\text{MUST}_{\text{adaptive}}$. We briefly compare different implementations of MUST in Tab. 1.

### 4.1 FORMULATE MUST AS A MULTI-ARMED BANDIT PROBLEM

Adaptively specifying different user simulators recalls us a similar thought in machine learning, i.e., the *boosting* strategy (Zhou, 2012). From a *boosting* point of view, one should increase the weights of weakly-performed data examples and decrease the weights for well-performed ones. In MUST, we accordingly assume that it should reduce the interactions with user simulators that dialogue system has performed well and allocate more interactions with those user simulators that dialogue system has not performed well yet . We refer to this strategy as *boosting adaption*.

[4] Continual Reinforcement Learning (CRL) Khetarpal et al. (2020) is a sequential learning paradigm for training an agent with RL algorithms.

Meanwhile, we should also give some chances to all user simulators to relieve the catastrophic forgetting issue. We refer to this as *uniform adaption*. Such a trade-off between *boosting adaption* and *uniform adaption* is similar to the Multi-armed bandit (MAB) problem (Auer et al., 2002) that mainly studies *the exploration vs exploitation trade-off* in Reinforcement Learning policies.

Here, we interpret MUST as a MAB problem. We treat each user simulator as an arm. Suppose there are $K$ arms (simulators), and each arm $i$ has a fixed but unknown reward distribution $R_i$ with an expectation $\mu_i$. At each time step $t = 1, 2, ..., T$, one must choose one of these $K$ arms. We denote the arm pulled at time step $t$ as $i_t \in \{1, ..., K\}$. After pulling an arm, it will receive a reward $x_{i_t}$ drawn from the arm's underlying reward distribution. The decision masker's objective is to maximize the cumulative expected reward over the time horizon

$$\sum_{t=1}^{T} \mathbb{E}[x_{i_t}] = \sum_{t=1}^{T} \mu_{i_t}. \tag{1}$$

In MUST, the reward received in each arm-pulling step refers to the possible performance gain of the dialogue system after it interacts with a selected user simulator. A significant *difference* between the standard MAB problem and MUST is that the reward expectation of a user simulator (arm) in MUST is not static; it changes over time. For example, by consecutively interacting with the same user simulator, the performance gain (reward) of the system will decay since it might be in saturation or overfitting to this simulator. Moreover, the performance gain of the system after interacting with a simulator might increases if the simulator has not been selected for a period. To deal with this *difference*, we should tailor the solution of MAB to the MUST framework, e.g., considering boosting adaption and uniform adaption.

### 4.2 Training the dialogue system with $\text{MUST}_{\text{adaptive}}$

To solve this MAB problem in MUST, we implement a method (called '$\text{MUST}_{\text{adaptive}}$') with a two-phase procedure, as presented in Algorithm 1. Similar to the UCB1 [5] algorithm, $\text{MUST}_{\text{adaptive}}$ specifies user simulators in a uniform distribution to interact with the system $S$ in the first $T_{\text{warmup}}$ steps (i.e., in the *warm-up phase*). After that, the *adaptive phase* will balance the boosting adaption and uniform adaption by introducing an adaptively-updated distribution $\boldsymbol{p}$, which is used to specify different user simulators to train the system agent $S$ in later RL training. To accelerate the RL training, intuitively, $\boldsymbol{p}$ is expected to *assign lower weights to user simulators that the system agent $S$ already performs well* and *higher weights to those user simulators that $S$ performs not well*.

(1) **Warm-up phase**: in the first $T_{\text{warmup}}$ dialogues, we use a uniform distribution to sample all user simulators to train the system agent $S$ (lines 4-7). This phase is mainly used to warm up the dialogue system $S$ and make it have little ability to converse with all user simulators.

(2) **Adaptive phase**: in this phase, the distribution $\boldsymbol{p}$ used to sample all user simulators will be adaptively updated, which is why we call this phase **adaptive phase**. When this phase begins, that is $t = 0$, we will first evaluate the performance (i.e., the success rate $\bar{x}_j, j \in \{1, \cdots, K\}$) of the dialogue system $S$ trained after the **warm-up phase**. The success rate $\bar{x}_j$ is obtained by letting $S$ interact $d$ times with the simulator $U_j$ (e.g., $j \in \{1, ..., K\}$) and calculating the success rates.

Inspired by UCB1 Auer et al. (2002), we design a calibrated **performance expectation** $\hat{x}_j$ of the system agent $S$ interacting with each user simulator $U_j$ taking exploration (the latter term) into consideration beyond purely exploitation (the former term):

$$\hat{x}_j = \bar{x}_j + \sqrt{\frac{2 \ln t}{T_{j,t}}}, j \in \{1, ..., K\}; \tag{2}$$

where $\bar{x}_j$ is the success rate tested with user simulator $U_j$ , $T_{j,t}$ is the number of times user simulator $U_j$ has been interacted so far. Then we normalize $\hat{x}_j$ into

$$z_j = 1/\left(\hat{x}_j - \tau \min(\{x_1, \cdots, x_K\})\right), \tag{3}$$

[5] One simple algorithm designed to solve the MAB problem is called the Upper Confidence Bound 1 (UCB1) algorithm (Auer et al., 2002). This policy first pulls each arm once in the first $K$ steps. Then the index of the arm will be played from $t = K + 1$ to $T$ is the sum of two terms: $i_t = \arg\max_i \left(\bar{x}_i + \sqrt{\frac{2 \ln t}{T_{i,t}}}\right)$.

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

**Input:** K fixed **U**ser simulators $\mathbf{U} = \{U_1, U_2, \cdots U_K\}$ and the values of hyperparameters $T_{\text{warmup}}, T, e, d, s$;

1 **Initialization**: randomly-initializing **S**ystem agent $S$;
2 **Initialization**: initializing the simulator sampling distribution $\boldsymbol{p}$ as a uniform distribution.
3 (1) **Warm-up phase:**
4 **for** $t = 0, ..., T_{warmup} - 1$ **do**
5  sampling a simulator $U_i$ in $\mathbf{U}$ w.r.t. the distribution $\boldsymbol{p}$;
6  synthesizing a new dialogue using the system agent $S$ and the sampled $U_i$ ;
7  using the reward obtained for the dialogue to update $S$ with a RL algorithm;
8 (2) **Adaptive phase:**
9 **for** $t = 0, ..., T - 1$ **do**
10  **if** $t\%e == 0$ **then**
11  **for** $j = 1, ..., K$ **do**
12  evaluating the performance i.e. the success rate $\bar{x}_j$ of the agent $S$ by letting it interact $d$ times with the simulator $U_j$;
13  updating $\boldsymbol{p}$ based on these success rates $\{\bar{x}_1, ..., \bar{x}_K\}$ (see Eq. 2, Eq. 3, and Eq. 4);
14  **else**
15  sampling a simulator $U_i$ in $\mathbf{U}$ w.r.t. the distribution $\boldsymbol{p}$;
16  synthesizing a new dialogue using the system agent $S$ and the sampled $U_i$ ;
17  using the reward obtained for the dialogue to update $S$ with a RL algorithm;

**Output:** The learned dialogue system $S$.

Eq. 3 penalizes the user simulators that the dialogue system already performs well in the expectation term. Where the hyperparameter $\tau$ is the smooth factor for distribution $\boldsymbol{p} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_K\}$ – the larger $s$ is, the sharper $\boldsymbol{p}$ is. Each probability $\boldsymbol{p}_j$ in $\boldsymbol{p}$ is calculated as

$$\boldsymbol{p}_i = \frac{z_j}{\sum_{j=1}^{K} z_j}. \tag{4}$$

In the following $T - 1$ dialogues, we will specify all user simulators to train the system agent $S$ with this distribution $\boldsymbol{p}$ (lines 15-18). And we will also evaluate the RL model $S$ for every $e$ episodes (line 10-12) and update the distribution $\boldsymbol{p}$ with the new $K$ success rates (line 13).

**Difference with the original UCB1.** The main differences between our modified UCB1 algorithm and the original UCB1 algorithm are twofold. First, we tailor the original UCB1 in our scenario using Eq. 3, to penalize the user simulators that the dialogue system has performed well. Secondly, we adopt a sampling schema based on a well-designed distribution (see Eq. 4), instead of taking the arm with the highest expectation. This is to increase the diversity and flexibility of arm selection.

## 5 EXPERIMENTS

To verify the effectiveness of MUST, we benchmark the system agents trained either with a single user simulator or multiple user simulators (including $\text{MUST}_{\text{merging}}$, $\text{MUST}_{\text{uniform}}$, $\text{MUST}_{\text{adaptive}}$).

### 5.1 EXPERIMENTAL SETUP

**Available user simulators.** There are six user simulators provided by Shi et al. (2019), which are Agenda-Template (**AgenT**), Agenda-Retrieval (**AgenR**), Agenda-Generation (**AgenG**), RNN-Template (**RNNT**), RNN-Retrieval (**RNNR**), RNN-End2End (**RNN**) trained with different dialog planning and generation methods. The NLU modules of all six user simulators are using the RNN model. The DM modules of **AgenT**, **AgenR**, and **AgenG** are rule-based methods. For the NLG module, these three simulators are using the template, retrieval, and generation methods respectively. The DM modules of **RNNT**, **RNNR** are using Sequicity (Lei et al., 2018) as their backbones which is an RNN-based seq2seq model with copy mechanism. The NLG modules of these two simulators

are using the template and retrieval methods respectively. The user simulator of **RNN** uses Sequicity as its backbone in an end-to-end manner.

**Baselines.** The baselines are the dialogue systems trained by each user simulator, including **Sys-AgenT**, **Sys-AgenR**, **Sys-AgenG**, **Sys-RNNT**, **Sys-RNNR**, and **Sys-RNN**. For a fair comparison, all system agents (including the systems trained by our MUST) have the same architecture described in Shi et al. (2019). See basic modules of user simulators and dialogue systems in App. B.1.

**MultiWOZ Restaurant Domain Dataset.** The original task in MultiWOZ (Budzianowski et al., 2018) is 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. Moreover, we randomly simulated 2,000 dialogues from each rule-based simulator AgenT, AgenR, AgenG, and their corresponding system agents respectively, and processed these dialogues to have the same annotation format as the MultiWOZ restaurant domain dataset. We denoted this dataset as **Simulated Agenda Dataset**.

**Evaluation Measures.** The direct automatic metric to evaluate the dialogue system is **the success rate** tested by each user simulator. We calculate the success rate between a user simulator and a system agent by sampling 200 dialogues. We exclude some user simulators in MUST training and test the system by them as **out-of-domain evaluation**. According to the previous study Gunasekara et al. (2020), there usually has a gap between automatic evaluations and human evaluations of dialogue systems. Therefore, we ask human users to converse with dialogue systems. Each dialogue system has conversed with 5 different users; each user has 10 dialogues. In total, we collect 50 dialogues for each dialogue system to calculate its success rate. See more details in App. B.2.

## 5.2 Implementations

### 5.2.1 Two new User Simulators

We believe Pre-trained Language Models (PLMs) might improve the capacity of user simulators since they have recently shown remarkable success in building task-oriented dialogue systems (Ham et al., 2020; Peng et al., 2021; Hosseini-Asl et al., 2020). Here we implement another two user simulators using GPT (Radford et al., 2018; 2019). Building a user simulator using GPT is similar to building a ToD system with GPT. See more details in App. C.

**GPT Simulator.** It is first fine-tuned on the *simulated agenda dataset* and then fine-tuned on the *MultiWOZ restaurant domain dataset* by leveraging GPT. This user simulator will be used to help implementing MUST.

**GPT$_{\text{IL}}$ Simulator.** Similar to Imitation Learning (IL), we first train a new user simulator with dialogue sessions collected from different user simulators and their corresponding dialogue systems. Like the GPT simulator, we learn this new user simulator based on GPT model and denote it as GPT$_{\text{IL}}$. GPT$_{\text{IL}}$ is first fine-tuned 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 fine-tuning GPT$_{\text{IL}}$.

### 5.2.2 Dialogue Systems

**Sys-GPT** is trained with the *single* user simulator GPT. **Sys-MUST**$_{\text{merging}}$ is trained by **GPT**$_{\text{IL}}$ for implementing MUST$_{\text{merging}}$ strategy. **Sys-MUST**$_{\text{uniform}}$ is trained by user simulators AgenT, AgenR, RNNT, and GPT [6] with uniform sampling distribution. In **Sys-MUST**$_{\text{adaptive}}$ [7], the distribution $\boldsymbol{p}$ is adaptively updated using our modified UCB1 algorithm.

[6] According to Shi et al. (2019), system agents trained purely by AgenG, RNNR, or RNN underperforms those trained by AgenT, AgenR, or RNNT. Our preliminary experiments also show that including AgenG, RNNR, and RNN is not beneficial in terms of performance in MUST but takes longer time for RL training. Therefore, we use only AgenT, AgenR, RNNT (excluding AgenG, RNNR, and RNN) and GPT to implement MUST (including MUST$_{\text{uniform}}$ and MUST$_{\text{adaptive}}$).

[7] See implementations of dialogue systems in App. B.3 and policy gradient in App. B.4.

Table 2: The success rates of the system agents were tested against various user simulators. Each column represents a user simulator, each row represents a dialogue system trained with a specific simulator, e.g., Sys-AgenT means the system trained with AgenT. Each entry shows the success rate on 200 dialogues collected from a user simulator and a system agent. We use four user simulators: AgenT, AgenR, RNNT, and GPT simulator to implement $\text{MUST}_{\text{uniform}}$ and $\text{MUST}_{\text{adaptive}}$.

| Dialogue Systems | | In-domain evaluation | | | | Out-of-domain evaluation | | | | | All | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AgenT | AgenR | RNNT | GPT | AgenG | RNNR | RNN | Avg.↑ | Std.↓ | Avg.↑ | Std.↓ |
| **single** | Sys-AgenT | 97.5 | 54.0 $\downarrow_{36.9\%}$ | 98.5 $\downarrow_{0.5\%}$ | 78.0 $\downarrow_{4.1\%}$ | 72.5 | 92.5 | 77.0 | 80.7 | 8.6 | 81.4 | 14.8 |
| | Sys-AgenR | 96.0 $\downarrow_{1.5\%}$ | 90.0 | 98.5 $\downarrow_{0.5\%}$ | 80.5 $\downarrow_{1.5\%}$ | 97.5 | 97.5 | 82.0 | 92.3 | 7.3 | 91.7 | 7.1 |
| | Sys-RNNT | 30.5 $\downarrow_{68.7\%}$ | 23.0 $\downarrow_{68.7\%}$ | 99.0 | 75.5 $\downarrow_{6.7\%}$ | 35.5 | 97.5 | 84.0 | 72.3 | 26.6 | 63.6 | 30.5 |
| | **Sys-GPT** | 60.5 $\downarrow_{37.9\%}$ | 51.5 $\downarrow_{39.5\%}$ | 97.0 $\downarrow_{2.0\%}$ | 82.0 | 59.5 | 94.0 | 92.0 | 81.8 | 15.8 | 76.6 | 17.6 |
| **MUST** | **Sys-MUST**$_{\text{merging}}$ | 97.5 $\uparrow_{0.0\%}$ | 83.5 $\downarrow_{6.7\%}$ | 94.5 $\downarrow_{4.6\%}$ | 80.5 $\downarrow_{1.5\%}$ | 97.5 | 94.0 | 82.5 | 91.3 | 6.4 | 90.0 | 6.9 |
| | **Sys-MUST**$_{\text{uniform}}$ | 97.5 $\uparrow_{0.0\%}$ | 89.0 $\downarrow_{1.0\%}$ | 97.5 $\downarrow_{1.5\%}$ | 82.5 $\uparrow_{0.5\%}$ | 96.5 | 96.0 | 87.5 | 93.4 | 4.2 | 92.4 | 5.6 |
| | **Sys-MUST**$_{\text{adaptive}}$ | 97.5 $\uparrow_{0.0\%}$ | 89.5 $\downarrow_{0.5\%}$ | 97.0 $\downarrow_{2.0\%}$ | 82.5 $\uparrow_{0.5\%}$ | 96.5 | 97.5 | 90.0 | **94.7** | **3.3** | **92.9** | **5.3** |

[1] The underlined number represents the success rate between a user simulator and its corresponding dialogue system trained by this user simulator. The increasing and decreasing percentages (in red and green colors) use the underlined numbers as the base success rates.
[2] ↓ (↑) indicates by what percentages the success rate has decreased (increased) compared with the base success rate by interacting with the same user simulator.

### 5.3 Experimental Results

**Automatic Evaluation.** As seen in Tab. 2, Sys-$\text{MUST}_{\text{uniform}}$ and Sys-$\text{MUST}_{\text{adaptive}}$ outperform the dialogue systems (Sys-AgenT, Sys-AgenR, Sys-RNNT, and Sys-GPT) trained by a single user simulator in the overall performance, demonstrating the superior of leveraging multiple user simulators. Especially, **Sys-MUST**$_{\text{adaptive}}$ has a 1.2 absolute value improvement (92.9 vs. 91.7) over the previous SOTA system Sys-AgenR. Observing that Sys-$\text{MUST}_{\text{merging}}$ is not as competitive as Sys-$\text{MUST}_{\text{uniform}}$ and Sys-$\text{MUST}_{\text{adaptive}}$, this might be because the merging strategy cannot effectively leverage multiple user simulators.

Table 3: Human evaluation results on dialogue systems.

| Dialogue Systems | | human evaluation |
|---|---|---|
| **single** | Sys-AgenT | 76.0 |
| | Sys-AgenR | 84.0 |
| | Sys-RNNT | 34.0 |
| | **Sys-GPT** | 58.0 |
| **MUST** | **Sys-MUST**$_{\text{merging}}$ | 90.0 |
| | **Sys-MUST**$_{\text{uniform}}$ | 92.0 |
| | **Sys-MUST**$_{\text{adaptive}}$ | 92.0 |

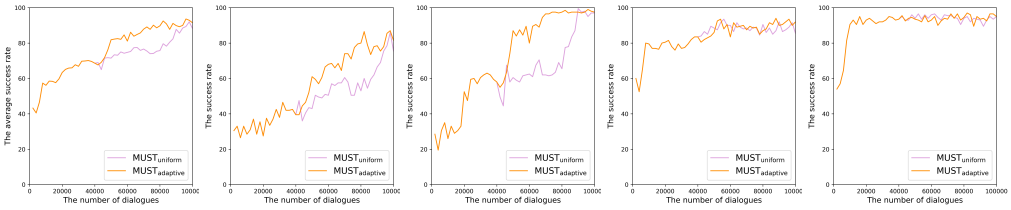
In **in-domain evaluation**, the performances of systems (Sys-AgenT, Sys-AgenR, Sys-RNNT, and Sys-GPT) trained by a single user simulator drop a lot when testing with a different user simulator. It requires us to delicately select a suitable user simulator for obtaining a good dialogue system. However, human users might be multi-facet or even unknown, which makes the selection difficult. Therefore, it is essential to leverage multiple user simulators when training dialogue systems. At least, the performance gap of dialogue systems trained by MUST becomes smaller than that without MUST, see the percentages labeled in green and red colors.

In **out-of-domain evaluation** where the user simulators used for testing are unseen by our MUST, Sys-$\text{MUST}_{\text{uniform}}$ and Sys-$\text{MUST}_{\text{adaptive}}$ achieve 2.4 absolute value improvements. This evidences that MUST has a better generalization ability for interacting with unseen user simulators. Moreover, MUST approaches (Sys-$\text{MUST}_{\text{merging}}$, Sys-$\text{MUST}_{\text{uniform}}$, and Sys-$\text{MUST}_{\text{adaptive}}$) have a lower standard deviation, which indicates that they are more robust to the diversity of user simulators.

**Human Evaluation.** The human evaluation results in Tab. 3 show that our Sys-$\text{MUST}_{\text{uniform}}$ and Sys-$\text{MUST}_{\text{adaptive}}$ largely outperforms the other dialogue systems when interacting with real users. The consistency between automatic evaluations and human evaluations evidences the effectiveness of our proposed MUST.

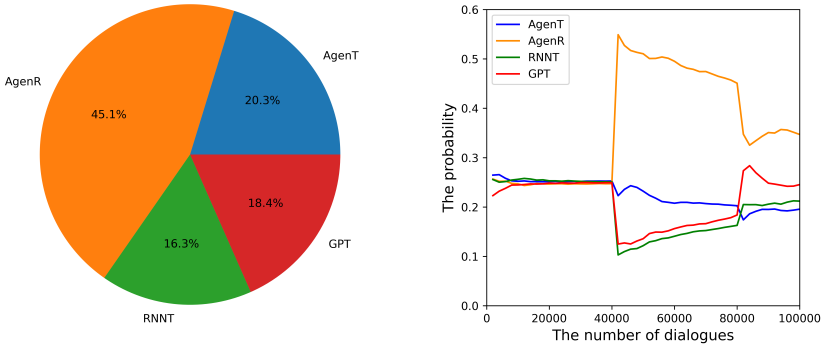
### 5.4 Analysis and Discussions

**Convergences between $\text{MUST}_{\text{uniform}}$ and $\text{MUST}_{\text{adaptive}}$.** In Fig. 2, we show the learning curves of Sys-$\text{MUST}_{\text{uniform}}$ and Sys-$\text{MUST}_{\text{adaptive}}$ in 100,000 steps; the first 40,000 steps are in the **warm-up phase** for Sys-$\text{MUST}_{\text{adaptive}}$. From Fig. 2(a), we see that training the dialogue system with AgenT, AgenR, RNNT, and GPT by $\text{MUST}_{\text{adaptive}}$ converges faster than $\text{MUST}_{\text{uniform}}$. We further plot the performances of the dialogue system tested by each user simulator in the RL training, which are shown in Fig. 2(b)-2(e).




(a) The performance (b) Tested by AgenR (c) Tested by AgenT (d) Tested by GPT (e) Tested by RNNT

Figure 2: The learning curves of Sys-MUST$_{\text{uniform}}$ and Sys-MUST$_{\text{adaptive}}$. (a) shows their average success rates tested by all user simulators (AgenT, AgenR, RNNT, and GPT). Success rates tested by each user simulator are in (b)-(e).




(a) The sampling proportion of simulators. (b) Variations of the sampling proportion (in every 2000 steps) of simulators.

Figure 3: The sampling proportion of user simulators in average (a) and in time horizon (b).

**Visualization of the patterns learned by MUST$_{\text{adaptive}}$.** Let us define the adaptation difficulty of a user simulator as the steps that the dialogue system should take to converge when trained by this user simulator. The adaptation difficulty of all user simulators could be ranked like AgenR > AgenT > GPT > RNNT according to Fig. 2(b)- 2(e). To check whether MUST$_{\text{adaptive}}$ tend to sample harder-to-adapt user simulators in the **adaptive phase**, as assumed in Sec. 4.2, Fig. 3(a) visualizes the sampling proportions of all user simulators. We could observe that AgenR was sampled with 45.1% (the biggest proportion) and it is indeed the hardest user simulator to be adapted; GPT has the smallest sampling proportion and it is the easiest to be adapted. The consistency between the adaptation difficulty and sampling proportions for these four user simulators evidences our assumption in Sec. 4.2. Interestingly, Fig. 3(b) visualizes the variations of the sampling distribution of user simulators, which shows that AgenR and AgenT are *competitive* with GPT simulator; while RNNT and GPT are *cooperative* with each other. This might be because RNNT and GPT simulator are learned from the dialogue corpus and share some similar behaviors.

## 6 Conclusion

In this paper, we propose a framework named MUST to improve the system agent by using multiple user simulators simultaneously. We discuss several simple methods to implement MUST, which are either inflexible or inefficient. Therefore, we formulate MUST as a Multi-armed bandits (MAB) problem, based on which we propose a novel implementation for MUST. The experimental results on the restaurant search task from MultiWOZ demonstrate that our proposed MUST can largely improve the system agent upon the baseline methods, especially when the tested user simulators are unseen. Moreover, MUST$_{\text{adaptive}}$ is robust to the diversity of user simulators and its training is more efficient. The main limitation of this work is mainly that we only conduct our experiments on the restaurant domain of the MultiWOZ since we can only find multiple user simulators from Shi et al. (2019) and they build these simulators only on the restaurant search task. In future work, we plan to apply our proposed methods to multi-domain scenarios.

## REFERENCES

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

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002. URL http://homes.dsi.unimi.it/~cesabian/Pubblicazioni/ml-02.pdf.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 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*, pp. 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1547. URL https://aclanthology.org/D18-1547.

Yun-Nung Chen, Asli Celikyilmaz, and Dilek Hakkani-Tür. Deep learning for dialogue systems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pp. 8–14, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL https://aclanthology.org/P17-5004.

R. Chulaka Gunasekara, Seokhwan Kim, Luis Fernando D'Haro, Abhinav Rastogi, Yun-Nung Chen, Mihail Eric, Behnam Hedayatnia, Karthik Gopalakrishnan, Yang Liu, Chao-Wei Huang, Dilek Hakkani-Tür, Jinchao Li, Qi Zhu, Lingxiao Luo, Lars Liden, Kaili Huang, Shahin Shayandeh, Runze Liang, Baolin Peng, Zheng Zhang, Swadheen Shukla, Minlie Huang, Jianfeng Gao, Shikib Mehri, Yulan Feng, Carla Gordon, Seyed Hossein Alavi, David R. Traum, Maxine Eskénazi, Ahmad Beirami, Eunjoon Cho, Paul A. Crook, Ankita De, Alborz Geramifard, Satwik Kottur, Seungwhan Moon, Shivani Poddar, and Rajen Subba. Overview of the ninth dialog system technology challenge: DSTC9. *CoRR*, abs/2011.06486, 2020. URL https://arxiv.org/abs/2011.06486.

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

Donghoon Ham, Jeong-Gwan Lee, Youngsoo Jang, and Kee-Eung Kim. 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*, pp. 583–592, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.54. URL https://aclanthology.org/2020.acl-main.54.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 20179–20191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/e946209592563be0f01c844ab2170f0c-Paper.pdf.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *CoRR*, abs/2012.13490, 2020. URL https://arxiv.org/abs/2012.13490.

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*, pp. 60–69, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5007. URL https://aclanthology.org/W18-5007.

Wenqiang Lei, Xisen Jin, Min-Yen Kan, Zhaochun Ren, Xiangnan He, and Dawei Yin. 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)*, pp. 1437–1447, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1133. URL https://aclanthology.org/P18-1133.

Jiwei Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1192–1202, Austin, Texas, November 2016a. Association for Computational Linguistics. doi: 10.18653/v1/D16-1127. URL https://aclanthology.org/D16-1127.

Xiujun Li, Zachary C Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. A user simulator for task-completion dialogues. *arXiv preprint arXiv:1612.05688*, 2016b.

Hsien-chin Lin, Nurul Lubis, Songbo Hu, Carel van Niekerk, Christian Geishauser, Michael Heck, Shutong Feng, and Milica Gasic. Domain-independent user simulation with transformers for task-oriented dialogue systems. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 445–456, Singapore and Online, July 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.sigdial-1.47.

Bing Liu and Ian Lane. End-to-end learning of task-oriented dialogs. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 67–73, New Orleans, Louisiana, USA, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-4010. URL https://aclanthology.org/N18-4010.

Bing Liu and Ian R. Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. *CoRR*, abs/1609.01454, 2016. URL http://arxiv.org/abs/1609.01454.

Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. Composite task-completion dialogue policy learning via hierarchical deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2231–2240, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1237. URL https://aclanthology.org/D17-1237.

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)*, pp. 2182–2192, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1203. URL https://aclanthology.org/P18-1203.

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. Few-shot natural language generation for task-oriented dialog. *CoRR*, abs/2002.12328, 2020. URL https://arxiv.org/abs/2002.12328.

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

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

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

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

Jost Schatzmann and Steve Young. The hidden agenda user simulation model. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):733–747, 2009. doi: 10.1109/TASL.2008.2012071.

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

Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. 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)*, pp. 1990–2000, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1206. URL https://aclanthology.org/D19-1206.

Ryuichi Takanobu, Runze Liang, and Minlie Huang. 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*, pp. 625–638, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.59. URL https://aclanthology.org/2020.acl-main.59.

Bo-Hsiang Tseng, Yinpei Dai, Florian Kreyssig, and Bill Byrne. Transferable dialogue systems and user simulators. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 152–166, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.13. URL https://aclanthology.org/2021.acl-long.13.

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*, pp. 1711–1721, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1199. URL https://aclanthology.org/D15-1199.

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pp. 438–449, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://aclanthology.org/E17-1042.

Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. The dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pp. 404–413, Metz, France, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/W13-4065.

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. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

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

Zhi-Hua Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

# A 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 $\mu_j$. 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 $\sum_{t=1}^{T} \mathbb{E}[X_{j_t}] = \sum_{t=1}^{T} \mu_{j_t}$.

# B MORE DETAILS ABOUT TRAINING DIALOGUE SYSTEMS.

## B.1 THE ARCHITECTURES OF USER SIMULATORS AND DIALOGUE SYSTEMS.

The basic modules of user simulators and dialogue systems are detailed in Tab. 4

Table 4: The architectures of user simulators and dialogue systems. Modules with † are trainable.

| **Agent Types** | **Agents** | **NLU** | **DM** | **NLG** |
|---|---|---|---|---|
| User Simulators | **AgenT** (Shi et al., 2019) | **RNN**† | Agenda | Template |
| | **AgenR** (Shi et al., 2019) | **RNN**† | Agenda | Retrieval |
| | **AgenG** (Shi et al., 2019) | **RNN**† | Agenda | **RNN**† (Generation) |
| | **RNNT** (Shi et al., 2019) | **RNN**† | | Template |
| | **RNNR** (Shi et al., 2019) | **RNN**† | | Retrieval |
| | **RNN** (Shi et al., 2019) | **RNN**† (NLU + NLG) | | |
| | GPT (ours) | **Transformer**† (NLU + DM + NLG) | | |
| | $\text{GPT}_{\text{IL}}$ (ours) | **Transformer**† (NLU + DM + NLG) | | |
| Dialogue Systems | All | **RNN**† | **RNN**† | Template |

## B.2 HUMAN EVALUATION ON DIALOGUE SYSTEMS

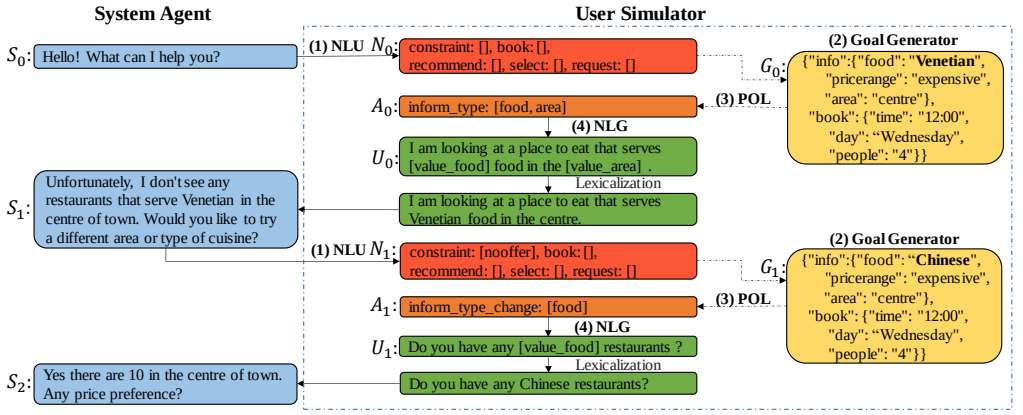
We find 5 volunteers to conduct the human evaluations on dialogue systems. They all have good English skills and are unpaid. Before the experiments, we introduced task-oriented dialogue systems and user simulators to them and tell them how to judge if the generated dialogue is successful. Then we prepare 50 user goals from **MultiWOZ Restaurant Domain Dataset**: 20 of them are simple, and 30 of them are a little bit complex. We specify 10 user goals for each volunteer and let the volunteer converse with all dialogue systems for each same user goal. In total, we collect 50 dialogues for each dialogue system to calculate its success rate.

## B.3 THE IMPLEMENTATIONS OF DIALOGUE SYSTEMS

The NLU modules of all system agents are a 2-layer bidirectional-GRU with 200 hidden units. The NLG modules of them are using the template-based method. The DM modules of them are a simple MLP. The input of the DM module is a state representation, which consists of the traditional dialog state and word count vector of the current utterance same as Shi et al. (2019). We mainly use the policy gradient method to train the DM modules of dialogue systems from scratch.

## B.4 THE DETAILS OF RUNNING POLICY GRADIENT

For training the DM modules of dialogue systems with the policy gradient method, we also apply the $\epsilon$-greedy exploration strategy. We let $\epsilon$ be 0.5 in the beginning, and it will decrease to 0 linearly within the RL training. The dialogue ends either when the user simulators say "goodbye" or when




(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 4: The overview of our U-GPT which consists of Natural Language Understanding (NLU), Goal Generator, Dialog Policy Learning (POL), and Natural Language Generation (NLG). It 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).

the number of turns of the dialogue exceeds 10. The reward will be given $+1$ for task success, -1 for task failure, and -0.1 for each additional turn to encourage the RL-based policy module to finish the task fast. Also, a discounted factor of 0.9 is applied to all the experiences.

### B.5 The parameters of training Sys-MUST

Table 5: The hyperparameters used for training the Sys-MUST$_{\text{adaptive}}$.

| Hyperparameter | Value |
|---|---|
| $T$ | 100,000 |
| $T_0$ | 40,000 |
| $e$ | 2,000 |
| $d$ | 200 |
| $s$ | 0.75 |

The hyperparameters used to train the Sys-MUST$_{\text{adaptive}}$ are listed in the Tab. 5.

## C Modeling User Simulator with GPT

We name the model of building a user simulator based on GPT as U-GPT. In this section, we will illustrate its details and conduct experiments to prove that it is a better model for building a user simulator.

### C.1 The architecture of U-GPT

As Fig. 4(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 "no-offer" 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 4(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 $\Theta$.

### C.2 Evaluations on U-GPT

To evaluate our proposed U-GPT, we adopt both **indirect** evaluations and **direct** evaluations as in Shi et al. (2019). We 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 & 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 goodness of a user simulator. For direct evaluations, we adopt six evaluation measures to evaluate the diversity of user simulators automatically: average utterance length, vocabulary size, Dist-1, Dist-2 (Li et al., 2016a) and Entropy (Zhang et al., 2018). We also ask human users to rate the simulated dialogues [8] to assess the user simulators directly. We use five same metrics as Shi et al. (2019) which are Fluency, Coherence, Goal Adherence, Diversity, and Overall quality to assess user simulators from multiple aspects.

### C.3 Training details of 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 the validation set through hyperparameters search of learning rate and batch size. The best models were fine-tuned with a batch size of 64 and a learning rate of 1e-3 over the corresponding dataset. We use the greedy decoding strategy for generating word-tokens in the inference phrase.

### C.4 Experiments

**GPT-RNN.** Because the implementation of user simulator RNN mainly consists of NLU and NLG, we remove the POL module from U-GPT and use the same annotated data as RNN to fine-tune it to compare our U-GPT with the RNN-based methods fairly and name it as GPT-RNN.

[8] 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.

As Tab. 6, Tab. 7, Tab. 8 show, GPT-RNN outperforms the user simulator RNN. It proves the power of leveraging GPT.

Our GPT-RNN performs better than the user simulator RNNT, which can be seen from the cross-model evaluation results in Tab. 6, the automatic evaluation results in Tab. 7, and the Hu.Div score in the human evaluation results in Tab. 8. However, as Tab. 8 shows, RNNT performs better than our GPT-RNN in the overall performance from the human evaluation. We think this might be because (1) the third-party system also has an impact on the generated dialogues and (2) the NLG module of RNNT is the template-based method which leads to the generated dialogues from RNNT being easy for the third-party system to understand and interact with.

The automatic evaluation results in Tab. 7 and the Hu.Div score in the human evaluation results in Tab. 8 show that RNNR can generate more diverse language than our GPT-RNN. We think it is because the user utterances generated by 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 RNNR are more diverse, the dialogues generated by our GPT-RNN are more fluent and coherent. Also, the cross-model evaluation results in Tab. 6 show that GPT-RNN can help to learn a more robust system agent than RNNR, but the Hu.All score in the human evaluation in Tab. 8 gives the opposite result.

Table 6: Cross study results. Each entry shows the success rate obtained by having the user simulator interacting with the RL system for 200 times.

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

Table 7: Automatic evaluation results of RNNT, RNNR and GPT-RNN. The metrics include average utterance length (Utt), vocabulary size (Vocab), distinct-n (DIST-n) and entropy (ENT-n).

| User Simulators | Utt ↑ | Vocab ↑ | DIST-1 ↑ | DIST-2 ↑ | ENT-4 ↑ |
|---|---|---|---|---|---|
| RNNT | 9.83 | 192 | 0.77% | 1.51% | 4.24 |
| RNNR | 11.06 | 346 | 2.45% | 9.59% | 6.59 |
| RNN | 10.95 | 205 | 1.17% | 3.14% | 4.98 |
| GPT-RNN | 14.00 | 262 | 1.13% | 3.53% | 5.62 |

Table 8: Human evaluation results of RNNT, RNNR and GPT-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).

| User Simulators | Hu.Fl ↑ | Hu.Co ↑ | Hu.Go ↑ | Hu.Div ↑ | Hu.All ↑ |
|---|---|---|---|---|---|
| RNNT | 4.60 | 4.68 | 4.96 | 3.34 | 4.70 |
| RNNR | 3.92 | 3.88 | 4.72 | 3.94 | 4.16 |
| RNN | 2.80 | 2.30 | 2.86 | 2.74 | 2.30 |
| GPT-RNN | 4.10 | 4.04 | 4.30 | 3.70 | 4.00 |