Chapter 4

Evaluation of Task-Oriented Dialogue Generation Systems

Evaluation of chatbots refers to the process in which a chatbot agent is tested against a number of metrics in which its ability in different phases of language understanding and relevant response generation are tested. The chatbots are usually supposed to act like humans and create an engaging and context-relevant response, it is challenging and hard to evaluate these systems. Chatbots can be either evaluated automatically or manually. In the manual evaluation, the conversation history of a chatbot with real users (or a user simulator) is passed to evaluators (e.g., Amazon MT workers) and its quality is evaluated based on domain-specific metrics. In this study, we are more interested in automatic evaluation metrics, and specifically, we will focus on evaluation metrics for task-oriented chatbots as manual evaluation is costly and time consuming.

Regardless of the type of the chatbot (i.e., task-oriented or open-domain) and its architecture (i.e., end-to-end or pipelined), chatbots have to go through three phases for response generation. The first phase is understanding user utterance and extracting the necessary and relevant information from the user utterance. The second phase is processing all the information from the current state of the conversation and the information from the user utterance and choose an appropriate action (e.g., issuing an API call, query a database, ask the user more information, etc.). The third phase for the chatbot is the NLG part in which the chatbot creates an appropriate response in the form of natural language which is understandable by the user. It is important to note that the aforementioned three phases are more related to pipelined chatbots in which for each of these phases there is a trained module and the final chatbot is created by concatenating these modules (see Section 3.2.4). In the end-to-end chatbot architecture, these phases are implicitly learned by the model.

The evaluation metrics and methods used for the evaluation of task-oriented and open-domain chatbots have slight difference due to the fact that their final purpose is different. Task-oriented chatbots are supposed to complete a specific task in as few turns as possible and provide relevant responses while open-domain chatbots are intended to keep the user engaged and have a long conversation with the user. The evaluation methods and metrics used in sequence modeling tasks such as BLEU [82] which is usually used in machine translation tasks is also used in chatbot evaluation. The other methods that are usually in the evaluation of task-oriented chatbots are per-turn accuracy, per-dialogue accuracy, and entity F1 score. It is important to notice that the evaluation metrics of task-oriented chatbots do not model user error and characteristic in the evaluation process. Using a user-profile conditioned user simulator which is able to simulate a real user with different characteristics seems a promising approach in order to better evaluate the task-oriented chatbots. We create such an evaluation method using the user simulator, NLU, and NLG modules and evaluate the trained chatbots by measuring metrics such as the number of turns in the conversation in order to provide a proper evaluation method which models the user characteristic in the process of evaluation.

The outline of this chapter is as follows. In Section 4.1 we explain with detail the evaluation metrics that we will later use in our experiments. Later in Section 4.3 we explain the profile-conditioned user simulator evaluation method and explain its different components.

4.1 Evaluation Metrics for Task-oriented Dialogue Generation Systems

Evaluation of dialogue generation systems is a hard task [69][14] due to the fact that the generated dialogues have to be both structurally and semantically correct and coherent while the latter is a very hard task. For this study, we use four common metrics used in the evaluation of task-oriented chatbots [27]. Given the fact that end-to-end architectures of chatbots are language generation models, thus we can use the evaluation metrics used in the evaluation of language generation models for evaluation of chatbots. BLEU metric is one of the most popular evaluation methods for such systems and thus we will use it for the evaluation phase.

As explained in Section 3.1.2, one of the main tasks of task-oriented chatbots is to extract the domain-specific entities from the user utterance and thus it is important to evaluate the chatbot's ability for this task. Entity F1 score is the evaluation metric that we will use for the evaluation of end-to-end trained chatbots for this task. The other two methods that we will use for evaluation of chatbots are per-turn and per-dialogue accuracy which measure how chatbot can generate response according to the training corpus. In the following sections, we will dive more into these evaluation methods and also discuss their drawbacks.

4.1.1 BLEU

We use the BLEU (bilingual evaluation understudy) [82] metric which is commonly used in machine translation tasks. BLEU metric can be used to evaluate dialogue generation models as in [66][92][27]. BLEU metric is a word-overlap metric which computes the co-occurrence of N-grams in the reference and the generated response and also applies the brevity penalty which tries to penalize far too short responses which are usually not desired in task-oriented chatbots. We compute the BLEU score using all generated responses of our systems.

The formula for calculating BLEU score is shown in Equation (4.1):

$$BLEU = \min(1, \frac{\text{output length}}{\text{reference length}}) * (\prod_{i=1}^{4} \text{precision}_i)^{\frac{1}{4}} \tag{4.1}$$

where the first term, $\min(1, \frac{\text{output length}}{\text{reference length}})$, is brevity penalty and penalizes the answers which are far too short. The second term in Equation (4.1), $(\prod_{i=1}^{4} \text{precision}_i)^{\frac{1}{4}}$, is the product of precision for all N-grams of size 1 to 4.

BLEU metric was mainly designed for evaluation of machine translation tasks [125], but it has been used for evaluation almost all the NLP tasks which are in the form of language generation. Using BLEU score makes sure that the generated response is identical to the target response in the test dataset, but the model could generate a response which is entirely different from the target response though still a valid response. The usage of beam search can alleviate this problem [122]. In beam search decoding, the model considers the K most probable tokens and computes the multiplied log probabilities of tokens for each path and this process is repeated for each path until end of sentence token is reached and the output sentence (i.e. system response in case of a task-oriented chatbot) is generated (see 5.3 for more detail).

4.1.2 Per-Turn Accuracy

Per-turn accuracy measures the similarity of the system generated response versus the target response. Eric and Manning [27] used this metric to evaluate their systems in which they considered their response to be correct if all tokens in the system generated response matched the corresponding token in the target response. This metric is a little bit harsh, and the results may be low since all the tokens in the generated response have to be exactly in the same position as in the target response.

As an example, consider the target response "*The reservation for Pom restaurant at 2 pm is confirmed*" which is a turn in a conversation about reserving a restaurant. If the chatbot manage to create the exact same response as the target response then it is deemed to have created the correct response. Note that even if the chatbot creates the response with missed, additional, or reordered tokens which may have the same meaning as the target response it won't be deemed as the correct response.

4.1.3 Per-Dialogue Accuracy

We calculate per-dialogue accuracy as used in [27][9]. For this metric, we consider all the system generated responses and compare them to target responses. A dialogue is considered to be true if all the turns in the system generated responses match the corresponding turns in the target responses.

Consider a sample target conversation $X = \{x_1, x_2, ..., x_k\}$ in which x_i is the *ith* turn in the conversation with total of k turns. Consider the case in which the chatbot creates the response $X' = \{x'_1, x'_2, ..., x'_k\}$. The response, here the whole conversation X', is considered is to be correct if every token in x'_i is in the exact same position as in the x_i for i=1,...,k. Note that the order of turns and the number of turns do matter in this metric and that is why it is considered a very harsh metric.

4.1.4 Entity F1 Score

Datasets used in task-oriented tasks have a set of entities which represent user preferences. For example, in restaurant domain chatbots common entities are meal, restaurant name, date, time and the number of people. These are usually the required entities which are crucial for making reservations, but there could be optional entities such as location or rating. Each target response has a set of entities which the system asks or informs the user about. Our models have to be able to discern these specific entities and inject them into the generated response. To evaluate our models we could use named-entity recognition evaluation metrics [50]. F1 score is the most commonly used metric used for evaluation of named-entity recognition models which is the harmonic average of precision and recall of the model. We calculate this metric by micro-averaging over all the system generated responses.

$$F - Score = 2 * (\frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}) \tag{4.2}$$

4.2 User Simulators For Task-Oriented Chatbots

All the aforementioned evaluation methods of task-oriented chatbots try to assess the chatbot's ability in learning based on the training corpus, and the evaluation is done using the test dataset. However, given the functionality of a task-oriented chatbot in serving users for doing specific tasks, it makes sense to employ either real users or a user simulator for the evaluation of the chatbot. Utilizing real users for the task of chatbot evaluation can be costly, and in such cases, it may be useful to have a user simulator which can emulate the real users as much as possible. Building such a user simulator is a hard task due to the fact that the final user simulator has to be able to emulate the real user in a realistic way which consists of user behaviour and user characteristics. A user simulator is considered to be the alternative which tries to imitate the behaviour of chatbot users while conversing with the task-oriented chatbot.

There are generally two methods for creating user simulators for task-oriented chatbots which are agenda-based simulators and model-based simulators [31]. In agenda-based user simulator we use hand-crafted rules to simulate the interaction of a user with the task-oriented chatbot while in the model-based methods we rely on some corpus to learn the behaviour of the user. It is also possible to use a mix of both techniques for creating the user simulator.

Agenda-based user simulator [93] is one of the most widely used methods for implementation of user simulators as used in Microsoft's TC-bot [67]. The advantage of using an agenda-based user simulator is that there is no need for a dataset to train the user simulator and we can add as many hand-crafted rules as we need to the simulator to cover every possible case. This is needed for simulators used in industry chatbots since we need to handle every possible case (and corner cases) in order to make sure the chatbot works well. On the other hand, this process can be tedious and time-consuming since these rules are hand-crafted.

Model-based methods usually rely on learning the used behaviour based on some corpus [13]. End-to-end user simulators have recently been proposed which try to train end-to-end sequence learning models on the task-oriented corpus and emulate the user [60] [3]. Using end-to-end sequence learning models with later variable models has also been proposed to add user behaviour to a user simulator [40]. The advantage of using model-based user simulators is that we can learn the user behaviour from some training corpus without the need to create hand-crafted rules as used in agenda-based simulators. On the other hand, it is likely that some cases (most probably corner cases) are not available in the corpus and thus the chatbot is not evaluated on how it handles those situations.

We propose a new user simulator for task-oriented chatbots which has a pipelined architecture similar to Microsoft's user simulators used to train reinforcement learning agents in [67] but in our architecture we try to incorporate the user profile by conditioning the agenda-based user simulator on a user profile and thus try to make it easier to incorporate user behaviour and user characteristic into the user simulator.

4.3 Profile-Conditioned User Simulator

Our proposed method is based on a pipelined architecture which makes the task of adding user behaviour and user characteristic to the simulation easier. The architecture of our model is depicted in Figure 4.1.

There are five components in the profile-conditioned simulator. The user simulator, NLG, NLU, user goal, and user profile are implemented and trained independently and then concatenated together in order to create the user simulator. The mechanism of the proposed user simulator is as follows. The user simulator used utilizes a user profile and a user goal in order to create annotations. An annotation consists of a dialogue action and a list of entities with their values which are chosen randomly. An example of an annotation is *INFORM(cuisine=Persian,date=tomorrow)* which can be translated into natural language like "*I want to go to a Persian restaurant tomorrow*". The annotation is then passed to our template-based NLG and converted into nat-

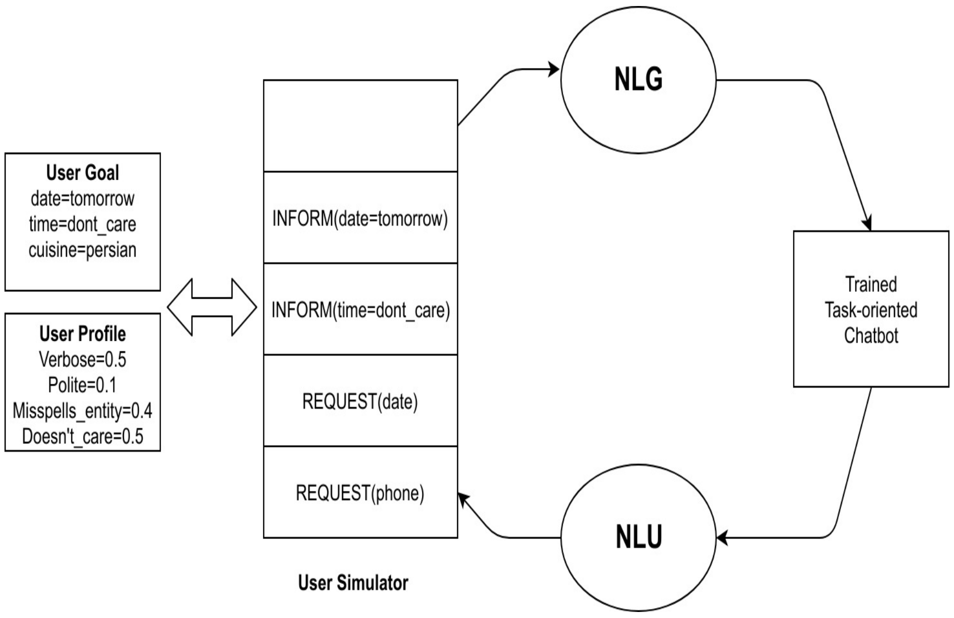

Figure 4.1: Architecture of a pipelined profile-conditioned user simulator

ural language. The generated sentence which conveys what the user simulator intends to convey to the chatbot is then passed to a trained task-oriented chatbot, and the response from the chatbot is passed to the NLU. The NLU component is the only component in our system that needs to be trained in order to identify the intent and the entities from the chatbot response. Based on the intent and list of entities extracted from the chatbot response, an annotation is created an passed to the user simulator in order to update its state. This process is repeated until the conversation ends. The end of the conversation is reached either when the user is satisfied and decides to end the conversation, or a maximum number of turns is reached. The latter is implemented in order to avoid infinite loops in the conversation in which a number of turns are repeated infinitely.

In the following sections, we will detail each of these components individually and explain the methods used for their implementation.

4.3.1 User Profile

User profile is a simple key-value data structure which stores the behavioural characteristics of the user. For instance, *verbosity*, *politeness*, *openness*, and *misspells_entities* are sample characteristics of a user which depict different ways in which the user can express its intention to the chatbot and also if he is likely to make mistakes in uttering the entities. These characteristics are completely domain-based and are intended to model different behaviours of real-user as much as possible in order to create a robust chatbot which can handle different types of users.

Verbosity, politeness, and misspelling are some examples of user characteristics which we can use in the user profile. If a user is verbose, then their utterance is longer compared to others and also it contains more dialogue actions and entities. As an example, a verbose user may say *"I want to book a table for two people tomorrow at Fancy Restaurant"* while a non-verbose one will only say *"I want to book a table for two people"*. A polite user tends to say greetings in the beginning of the conversation and also says goodbye when the conversations ends. The misspelling characteristic of the user makes them prone to misspell entities when they converse with the chatbot. Note that all these characteristics can be formulated into a probabilistic structure in which we can assign probability to each of the user characteristics. The higher the probability, the more the user shows that specific characteristic.

For example in Figure 4.1, the user profile shows the verbosity of user set to 0.5. This is a random number chosen from the uniform distribution on interval [0,1]. It is likely that this user creates long utterances with more than one dialogue action in the conversation. A verbosity 0 means that the user only pops one annotations from the agenda and a verbosity of 1 means that the user always pops the maximum allowed number of annotations from the agenda.

The user is also likely to misspell some entities when conversing with the chatbot with a chance of 0.4. When simulating this user, we misspell each entity with a chance of 0.4 by either adding extra characters or changing some random characters.

4.3.2 User Goal

User goal consists of a list of entities with their respective values. A sample user goal is shown in Figure 4.1. The user goal is a representation of user preference and can be used in the user simulator in order to either respond to some of the questions that the system asks the user in order to determine his preferences (*REQUEST* dialogue action from the chatbot).

In order to make the user goal more realistic, we add multiple entity types to it. The entity types that we use are *fixed*, *flexible*, *multiple-values*, and *open*. The fixed entities are the ones in which user only wants that specific value for that entity. The flexible entities means that the user preference is the value specified for that entity but is OK with other values in case that a specific value is not available. The multiple-value entity means that the user has a list of values for that entities which any of them can be used and the open entity means that the user does not care about the value of that entity.

In Figure 4.1, the user goal has three entities with their respective values. The time entity is of type open values since its value is "*dont_care*". The date and cuisine entities can be of type fixed or flexible since they only have on value (i.e. "*tomorrow*" for date and "*persian*" for cuisine). Note that when running the user simulator, all the entities will be assigned some type randomly and then some random values will be assigned to them based on their type.

4.3.3 Agenda-based User Simulator

The agenda-based user simulator [93] is one of the most widely used user simulators. Agenda is a stack-like data structure which stores annotations. The agenda is a representation of a user goal and its constraints. A sample of agenda-based user simulator is depicted in Figure 4.1. At the top of this agenda-based user simulator, the annotation *INFORM(date=tomorrow)* is available. If this annotation is popped from the user simulator, then it is passed to the NLG and then to the chatbot. We can also pop multiple annotations from the agenda based on the verbosity of the user. Also, it is important to notice that the agenda can be updated based on the answers from the chatbot. This is due to the fact that the agenda of a user simulator should contain all preferences and constraints that the user has in his mind at the current turn in the conversation.

When initializing the user simulator, we create an agenda for each intent that the user wants to cover. We then randomly sample the user goal for each intent and choose some of the entities with their respective values for each of them. Note that for those entities that are not available in the agenda we refer to the user goal in case we need their values during the conversation.

4.3.4 Rasa NLU

Rasa NLU[1] [8] is one of the most widely used open-source NLU frameworks. It provides two main functionalities for training an NLU model for task-oriented chatbots which are user intent classification and entity extraction. For user intent classification, one or a combination of different models such as keyword-based (similar to regex), SVM-based classification with Bag of Words (BOW) features, MITIE intent classification [58], and Facebook's StarSpace model for multilabel classification [124] are used.

For entity extraction, a number of named-entity recognition methods such as regex-based entity extraction, Facebook's duckling[2], CRF-based entity extraction [76], MITIE entity extractor [58], and Spacy's Named-entity Recognition (NER) model [46] is available to be used for entity extraction.

4.3.5 NLG

For NLG in our user simulator, we use a template-based NLG method in which the dialogue action and the list of the entities are converted into natural language. For example, an annotation such as *INFORM(date=tomorrow,time=2 pm)* will be converted into "*tomorrow at 2pm*".

We wanted to make it easier for the research community to use our profile-conditioned user simulator in their research and thus decided to create a framework, called ChatSim, for evaluation of task-oriented chatbots so that both

[1]https://github.com/RasaHQ

[2]https://github.com/facebook/duckling

researchers and practitioners can plug-in their task-oriented chatbot and get some preliminary evaluation of their chatbot in a few minutes.

We will finalize this chapter by explaining the ChatSim library and its components and use cases and will move into the experiments and results in the next chapter.

4.4 ChatSim; A Simulation-based Evaluation Library for Task-oriented Chatbots

ChatSim[3] is a architecture-agnostic evaluation library for task-oriented chatbots which uses different user simulators for the evaluation of task-oriented chatbots. The architecture-agnostic aspect of ChatSim means that it can be used for the evaluation of both end-to-end and also pipelined task-oriented chatbots. Its architecture is modular which consists of an NLU, an NLG, a user simulator, a moderator, and a task-oriented chatbot. The modulatiry of ChatSim allows the user to use different models for the NLG, NLU, and the user simulator components.

Since most of the components of the ChatSim are already explained in Section 4.3, we only briefly dive into its components in the next section. Then we explain the workflow of ChatSim and enumerate its use cases. Finally, we mention some future work on ChatSim and the components/features we could add to it in order to make it more useful for both researchers and practitioners.

4.4.1 ChatSim Components

As mentioned earlier, the components of ChatSim are an NLU, an NLG, a moderator, a user simulator, and the chatbot which we want to evaluate. The NLU component can be both a rule-based system or a machine learning based model. We plan to use Rasa NLU for the initial version of ChatSim, but users can train and add their custom NLU system to the framework if needed. The NLG component can also be both template-based (which we use for the initial version of ChatSim) or a neural sequence-to-sequence model.

[3]https://github.com/msaffarm/chatsim

The moderator is a new component we added to ChatSim in order to moderate the conversation flow and the interaction between the user simulator and the task-oriented chatbot. It makes sure that the conversation between the user simulator and the chatbot is sensible (e.g. the conversation does not turn into a conversation loop) and also controls the maximum number of turns in a single conversation between the user simulator and the chatbot. It also collects statistics about the performance of the chatbot with respect to different evaluation metrics. These metrics could be the average number of turns, and success rate of the task-oriented chatbot. User defined metrics can also be added to this component so that the moderator can keep track of them when running the conversation simulations.

The user simulator is probably the most important component in this framework which serves as the user chatting with the task-oriented chatbot. We use the profile-conditioned user simulator as our first simulator in this library, but like other components it can be replaced with custom simulators.

The chatbot which we want to evaluate can be both an end-to-end or a pipelined task-oriented chatbot. For end-to-end chatbots, we pass the response from the chatbot to the NLU system so that the dialogue action (see 3.3.1) and the entities can be extracted from the response and be passed to the user simulator. For the pipelined task-oriented chatbots, we bypass the NLU component since the response from the chatbot already contains the dialogue action and entities and thus can be directly passed to the user simulator. Note that we are using the profile-conditioned user simulator for this case and since the profile-conditioned user simulator takes dialogue action and entities as input to its agenda, we need to convert the response from the end-to-end chatbot into the format that is understandable by the user simulator. In case a different simulator is used, e.g. an end-to-end user simulator which takes a sequence as input rather than the dialogue action and entities, different strategies need to be implemented in order to make the whole simulation pipeline work. We consider this issue as a future work that needs to be addressed and integrated into the ChatSim.

4.4.2 ChatSim Workflow and Use Cases

The workflow of the simulation used in ChatSim follows the same workflow depicted in Figure 4.1. The moderator creates a user simulator with some user profile and simulates the conversation between the user simulator and the chatbot for a couple of times. It gets the utterance from the user simulator and passes it to the NLG module to create a natural language version of that utterance. That utterance is then given to the chatbot to process and the response from the chatbot is passed to the NLU in order to break it down to dialogue action and entities. The dialogue action and entities are passed to the user simulator so it can update its agenda based on that and create a response.

This cycle continues until either the chatbot can successfully fulfill the needs of the user simulator so that the user simulator ends the conversation or a maximum number of turns is reached and the conversation will be terminated. Note that the task-oriented chatbots are supposed to satisfy the needs of the user with as few turns as possible and therefore the intuition behind the latter criteria to end the conversation is to make sure that the conversation does no take much time to be handled by the task-oriented chatbot. The moderator runs this simulation for a couple of times by initializing the user simulator with different agendas, and the same personality, and collects statistics related to evaluation metrics.

We plan to make ChatSim a general evaluation platform for task-oriented chatbots so that users can easily plug in their chatbots into ChatSim and get evaluation metrics in a few minutes. Users are able to test their chatbots against different user personalities which makes it more realistic and enables them to find out about the possible weaknesses of their chatbots when specific user types are using their chatbot. The use cases of ChatSim would be all the possible use cases of task-oriented chatbots in different domain such a restaurant reservation chatbots, movie booking chatbots, flight booking chatbots, and so on.

4.4.3 ChatSim Desiderata

Our plan to further improve ChatSim is to add more models for different components such as NLU, NLG, and the user simulator and also add user custom evaluation metrics. For the NLU, we could add the feature that user can train neural models which can determine multiple intents in the utterance. For the NLG, we could give the user the ability to train a neural sequence-to-sequence model. We could also add an end-to-end user simulators to ChatSim so that users can train their own user simulator using their own corpus. This feature gives the users the ability to use specific training corpus (e.g., chat logs) for the user simulator which might be close to their target chatbot users.