Bootstrapping a Neural Conversational Agent with Dialogue Self-Play, Crowdsourcing and On-Line Reinforcement Learning

Pararth Shah[1], **Dilek Hakkani-Tür**[1], **Bing Liu**[2]*, **Gokhan Tür**[1]
[1]Google AI, Mountain View, CA, USA
pararth@google.com, {dilek,gokhan.tur}@ieee.org
[2]Carnegie Mellon University, Pittsburgh, PA, USA
liubing@cmu.edu

Abstract

End-to-end neural models show great promise towards building conversational agents that are trained from data and on-line experience using supervised and reinforcement learning. However, these models require a large corpus of dialogues to learn effectively. For goal-oriented dialogues, such datasets are expensive to collect and annotate, since each task involves a separate schema and database of entities. Further, the Wizard-of-Oz approach commonly used for dialogue collection does not provide sufficient coverage of salient dialogue flows, which is critical for guaranteeing an acceptable task completion rate in consumer-facing conversational agents. In this paper, we study a recently proposed approach for building an agent for arbitrary tasks by combining dialogue self-play and crowd-sourcing to generate fully-annotated dialogues with diverse and natural utterances. We discuss the advantages of this approach for industry applications of conversational agents, wherein an agent can be rapidly bootstrapped to deploy in front of users and further optimized via interactive learning from actual users of the system.

1 Introduction

Goal-oriented conversational agents enable users to complete specific tasks like restaurant reservations, buying movie tickets or booking a doctor's appointment, through natural language dialogue via a spoken or a text-based chat interface, instead of operating a graphical user interface on a device. Each task is based on a database *schema* which defines the domain of interest. Developing an agent to effectively handle all user interactions in a given domain requires properly dealing with variations in the dialogue flows (what information the users choose to convey in each utterance), surface forms (choice of words to convey the same information), database states (what entities are available for satisfying the user's request), and noise conditions (whether the user's utterances are correctly recognized by the agent). Moreover, the number of potential tasks is proportional to the number of transactional websites on the Web, which is in the order of millions.

Popular consumer-facing conversational assistants approach this by enabling third-party developers to build dialogue "experiences" or "skills" focusing on individual tasks (e.g. DialogFlow[1], Alexa Skills (Kumar et al. (2017)), wit.ai[2]). The platform provides a parse of the user utterance into a developer defined *intent*, and the developer provides a *policy* which maps user intents to system *actions*, usually modeled as flow charts[3]. This gives the developer full control over how a particular task is handled, allowing her to incrementally add new features to that task. However, some limitations are that (i) the developer must anticipate all ways in which users might interact with the agent, and (ii) since the programmed dialogue flows are not "differentiable", the agent's dialogue policy cannot be improved automatically with experience and each improvement requires human intervention to add logic to support a new dialogue flow or revise an existing flow.

Recently proposed neural conversational models (Vinyals and Le (2015)) are trained with supervision over a large corpus of dialogues (Serban et al. (2016, 2017); Lowe et al. (2017)) or with reinforcement to optimize a long term reward (Li et al. (2016a,b)). End-to-end neural conversational models for task-oriented dialogues (Wen et al. (2016); Liu and Lane (2017a)) leverage annotated dialogues collected with an expert to embed the expert's dialogue policy for a given task in

* Work done while the author was an intern at Google.

[1]https://dialogflow.com
[2]https://wit.ai
[3]https://dialogflow.com/docs/dialogs

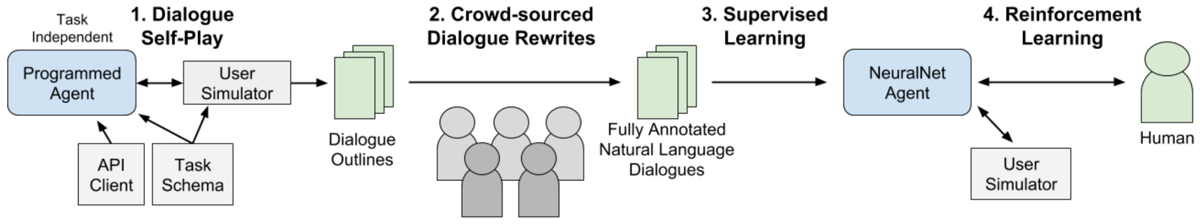


Figure 1: Bootstrapping a neural conversational agent.

the weights of a neural network. However, training such models requires a large corpus of annotated dialogues in a specific domain, which is expensive to collect. Approaches that use reinforcement learning to find the optimal policy also rely on a pre-training step of supervised learning over expert dialogues in order to reduce the exploration space to make the policy learning tractable (Fatemi et al. (2016); Su et al. (2016b, 2017); Liu and Lane (2017b)). A further issue with application of reinforcement learning techniques is that the user simulator used for the policy training step may not entirely mimic the behavior of actual users of the system. This can be mitigated by continuously improving the deployed agent from interactions with actual users via on-line learning (Gašić et al. (2011); Su et al. (2015, 2016a)).

The Wizard-of-Oz setup (Kelley (1984); Dahlbäck et al. (1993)) is a popular approach to collect and annotate task-oriented dialogues via crowd-sourcing for training neural conversational models (Wen et al. (2016); Asri et al. (2017)). However, this is an expensive and lossy process as the free-form dialogues collected from crowd-workers might contain dialogues unfit for use as training data, for instance if the crowd workers use language that is either too simplistic or too convoluted, or may have errors in dialogue act annotations requiring an expensive manual filtering and cleaning step. Further, the corpus might not cover all the interactions that the dialogue developer expects the agent to handle. In contrast, the recently proposed Machines Talking To Machines (M2M) approach (Shah et al. (2018)) is a functionality-driven process for training dialogue agents, which combines a dialogue self-play step and a crowd-sourcing step to obtain a higher quality of dialogues in terms of (i) diversity of surface forms as well as dialogue flows, (ii) coverage of all expected user behaviors, and (iii) correctness of annotations.

To apply these recent neural approaches to consumer-facing agents that must rapidly scale to new tasks, we propose the following recipe (Fig. 1): (1) exhaustively generate dialogue templates for a given task using *dialogue self-play* between a simulated user and a task-independent programmed system agent, (2) obtain natural language rewrites of these templates using crowd sourcing, (3) train an end-to-end conversational agent on this fully annotated dataset, achieving a reasonable task completion rate, and (4) deploy this agent to interact with users and collect user feedback, which serves as a reward value to continuously improve the agent's policy with on-line reinforcement learning updates. Consequently, a programmed dialogue agent's policy is distilled into a differentiable neural model which sustains a minimum task completion rate through guaranteed coverage of the interactions anticipated by the developer. Such an agent is safely deployable in front of actual users while also continuously improving from user feedback via lifelong learning.

The main contribution of this paper is two-fold:

1. an approach combining dialogue self-play, crowd-sourcing, and on-line reinforcement learning to rapidly scale consumer-facing conversational agents to new tasks.

2. discussion of practical solutions for improving user simulation and crowd-sourcing setups to guarantee coverage of salient dialogue flows and diversity of surface forms.

2 Approach

We present a brief overview of the Machines Talking To Machines (M2M) approach for bootstrapping a conversational agent. We direct the reader to the technical report Shah et al. (2018) for a detailed description of this approach.

Table 1: Sample dialogue outline and rewrite for movie ticket booking.

Outline		Rewrite
Annotations	**Template utterances**	**NL utterances**
S: greeting()	Greeting.	Hi, how can I help you?
U: inform(intent=book_movie, name=Inside Out, date=tomorrow, num_tickets=2)	Book movie with name is Inside Out and date is tomorrow and num tickets is 2.	I want to buy 2 tickets for Inside Out for tomorrow.
S: ack() request(time)	OK. Provide time.	Alright. What time would you like to see the movie?
U: inform(time=evening)	Time is evening.	Anytime during the evening works for me.
S: offer(theatre=Cinemark 16, time=6pm)	Offer theatre is Cinemark 16 and time is 6pm.	How about the 6pm show at Cinemark 16?
U: affirm()	Agree.	That sounds good.
S: notify_success()	Reservation confirmed.	Your tickets have been booked!

2.1 M2M

At a high level, M2M connects a *developer*, who provides the task-specific information, and a *framework*, which provides the task-independent information, for generating dialogues centered around completing the task. In this work we focus on database querying applications, which involve a relational database which contains entities that the user would like to browse and select through a natural language dialogue. The input to the framework is a task specification obtained from the developer, consisting of a *schema* of "slots" induced by the columns of the database and an *API client* which can be queried with a SQL-like syntax to return a list of matching candidate entities for any valid combination of slot values. For example, the schema for a movie ticket booking domain would include slots such as "movie name", "number of tickets", "date" and "time" of the show, etc. The API client would provide access to a database (hosted locally or remotely via the Web) of movie showtimes.

Outlines. With the task specification, the framework must generate a set of dialogues centered around that task. Each dialogue is a sequence of natural language utterances, i.e. dialogue *turns*, and their corresponding *annotations*, which include the semantic parse of that turn as well as additional information tied to that turn. For example, for the user turn "Anytime during the evening works for me", the annotation would be *"User: inform(time=evening)"*. The key idea in M2M is to separate the linguistic variations in the surface forms of the utterances from the semantic variations in the dialogue flows. This is achieved by defining the notion of a dialogue *outline* as a sequence of *template utterances* and their corresponding annotations. Template utterances are simplistic statements with language that is easy to generate procedurally. An outline encapsulates the semantic flow of the dialogue while abstracting out the linguistic variation in the utterances. The first two columns of Table 1 provide a sample dialogue outline for a movie ticket booking interaction, consisting of the annotations and template utterances, respectively.

Dialogue self-play. M2M proceeds by first generating a set of dialogue outlines for the specified task. A task-oriented dialogue involves the back and forth flow of information between a user and a system agent aimed towards satisfying a user need. Dialogue self-play simulates this process by employing a task-independent *user simulator* and *system agent* seeded with a task schema and API client. The user simulator maps a (possibly empty) dialogue history, a user profile and a task schema to a distribution over turn annotations for the next user turn. Similarly, the system agent maps a dialogue history, task schema and API client to a distribution over system turn annotations. Annotations are sampled from user and system iteratively to take the dialogue forward. The generated annotations consist of *dialogue frames* that encode the semantics of the turn through a *dialogue act* and a *slot-value map* (Table 1). For example "inform(date=tomorrow, time=evening)" is a dialogue frame that informs the system of the user's constraints for the date and time slots. We use the Cambridge dialogue act schema (Henderson et al. (2013)) as the list of possible dialogue acts. The process continues until either the user's goals are achieved and the user exits the dialogue with a "bye()" act, or a maximum number of turns are reached.

In our experiments we use an agenda-based user simulator (Schatzmann et al. (2007)) parameterized by a user goal and a user profile. The programmed system agent is modeled as a handcrafted finite state machine (Hopcroft et al. (2006)) which encodes a set of task-independent rules for constructing system turns, with each turn consisting of a *response* frame which responds to the user's previous turn, and an *initiate* frame which drives the dialogue forward through a predetermined sequence of sub-dialogues. For database querying applications, these sub-dialogues are: gather user preferences, query a database via an API, offer matching entities to the user, allow user to modify preferences or request more information about an entity, and finally complete the transaction (buying or reserving the entity) (Fig. 2). By exploring a range of parameter values and sampling a large number of outlines, dialogue self-play can generate a diverse set of dialogue outlines for the task.

Template utterances. Once a full dialogue has been sampled, a *template utterance generator* maps each annotation to a template utterance using a domain-general grammar (Wang et al. (2015)) parameterized with the task schema. For example, "inform(date=tomorrow, time=evening)" would map to a template *"($slot is $value) (and ($slot is $value))*"*, which is grounded as "Date is tomorrow and time is evening." The developer can also provide a list of templates to use for some or all of the dialogue frames if they want more control over the language used in the utterances. Template utterances are an important bridge between the annotation and the corresponding natural language utterance, as they present the semantic information of a turn annotation in a format understandable by crowd workers.

Crowd-sourced rewrites. To obtain a natural language dialogue from its outline, the framework employs crowd sourcing to paraphrase template utterances into more natural sounding utterances. The paraphrase task is designed as a "contextual rewrite" task where a crowd worker sees the full dialogue template, and provides the natural language utterances for each template utterances of the dialogue. This encourages the crowd worker to inject linguistic phenomena like coreference ("Reserve *that* restaurant") and lexical entrainment ("Yes, the *6pm show*") into the utterances. Fig. 5 in the Appendix provides the UI shown to crowd workers for this task. The same outline is shown to $K > 1$ crowd-workers to get diverse natural language utterances for the same dialogue. The third column of Table 1 presents contextual rewrites for each turn of an outline for a movie ticket booking task.

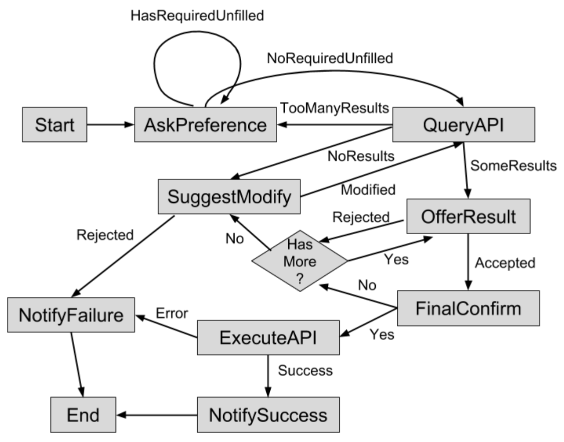


Figure 2: Finite state machine for a task-independent system agent for database querying applications.

Model training. The crowd sourced dataset has natural language utterances along with full annotations of dialogue acts, slot spans, dialogue state and API state for each turn. These annotated dialogues are sufficient for training end-to-end models using supervision (Wen et al. (2016)). Dialogue self-play ensures sufficient coverage of flows encoded in the programmed system agent in the crowd sourced dataset. Consequently, the trained agent reads natural language user utterances and emits system turns by encoding the FSM policy of system agent in a differentiable neural model.

2.2 On-line reinforcement learning

A limitation of training a neural agent on the dataset collected with M2M is that it is restricted to the flows encoded in the user simulator or the programmed system agent, and utterances collected from crowd-workers. When deployed to interact with actual users, the agent may find itself in new dialogue states that weren't seen during training. This can be mitigated by continually improving the agent's language understanding as well as dialogue policy by using a feedback score on each dialogue interaction of the neural agent as a reward value to optimize the end-to-end model using policy gradient reinforcement learning (RL). The RL updates can be done in two phases (which could be interleaved):

RL with user simulator. Since RL requires training for thousands of episodes, we construct a simulated environment in which the user simulator emits a user turn annotation, and a natural language utterance is sampled from the set of utterances collected for that dialogue frame from crowd sourcing. This enables the neural agent to discover dialogue flows not present in the programmed agent. The reward is computed based on successful task completion minus a turn penalty (El Asri et al. (2014)), and the model is updated with the on-policy REINFORCE update after each episode (Liu et al. (2017)).

RL with human feedback. For the agent to handle user interactions that are not generated by the user simulator, the agent must learn from its interactions with actual users. This is accomplished by applying updates to the model based on feedback scores collected from users after each dialogue interaction (Shah et al. (2016)).

3 User simulation and dialogue self-play

M2M hinges on having a generative model of a user that is reasonably close to actual users of the system. While it is difficult to develop precise models of user behavior customized for every type of dialogue interaction, it is easier to create a task-independent user simulator that operates at a higher level of abstraction (dialogue acts) and encapsulates common patterns of user behavior for a broad class of dialogue tasks. Seeding the user simulator with a task-specific schema of intents, slot names and slot values allows the framework to generate a variety of dialogue flows tailored to that specific task. Developing a general user simulator targeting a broad class of tasks, for example database querying applications, has significant leverage as adding a new conversational pattern to the simulator benefits the outlines generated for dialogue interfaces to any database or third-party API.

Another concern with the use of a user simulator is that it restricts the generated dialogue flows to only those that are engineered into the user model. In comparison, asking crowd workers to converse without any restrictions could generate interesting dialogues that are not anticipated by the dialogue developer. Covering complex interactions is important when developing datasets to benchmark research aimed towards building human-level dialogue systems. However, we argue that for consumer-facing chatbots, the primary aim is reliable coverage of critical user interactions. Existing methods for developing chatbots with engineered finite state machines implicitly define a model of expected user behavior in the states and transitions of the system agent. A user simulator makes this user model explicit and is a more systematic approach for a dialogue developer to reason about the user behaviors handled by the agent. Similarly, having more control over the dialogue flows present in the dataset ensures that all and only expected user and system agent behaviors are present in the dataset. A dialogue agent bootstrapped with such a dataset can be deployed in front of users with a guaranteed minimum task completion rate.

The self-play step also uses a programmed system agent that generates valid system turns for a given task. Since M2M takes a rule-based agent which works with user dialogue acts and emits a neural conversational agent that works with natural language user utterances, the framework effectively distills an expert dialogue policy combined with a language understanding module into a single learned neural network. The developer can customize the behavior of the neural agent by modifying the component rules of the programmed agent. Further, by developing a task-independent set of rules for handling a broad task like database querying applications (Fig. 2), the cost of building the programmed agent can be amortized over a large number of dialogue tasks.

4 Crowdsourcing

In the Wizard-of-Oz setting, a task is shown to a pair of crowd workers who are asked to converse in natural language to complete the task. The collected dialogues are manually annotated with dialogue act and slot span labels. This process is expensive as the two annotation tasks are difficult and therefore time consuming: identifying the dialogue acts of an utterance requires understanding the precise meaning of each dialogue act, and identifying all slot spans in an utterance requires checking the utterance against all slots in the schema. As a result, the crowd-sourced annotations may need to be cleaned by an expert. In contrast, M2M significantly reduces the crowd-sourcing expense by automatically annotating a majority of the dialogue turns and annotating the remaining turns with two simpler crowd-sourcing tasks: “Does this utterance contain this particular slot value?” and “Do these two utterances have the same meaning?”, which are easier for the average crowd worker.

Further, the lack of control over crowd workers' behavior in the Wizard-of-Oz setting can lead to dialogues that may not reflect the behavior of real users, for example if the crowd worker provides all constraints in a single turn or always mentions a single constraint in each turn. Such low-quality dialogues either need to be manually removed from the dataset, or the crowd participants need to be given additional instructions or training to encourage better interactions (Asri et al. (2017)). M2M avoids this issue by using dialogue self-play to systematically generate all usable dialogue outlines, and simplifying the crowd-sourcing step to a dialogue paraphrase task.

5 Evaluations

We have released[4] two datasets totaling 3000 dialogues collected using M2M for the tasks of buying a movie ticket (Sim-M) and reserving a restaurant table (Sim-R). We present some experiments with these datasets.

5.1 Dialogue diversity

First we investigate the claim that M2M leads to higher coverage of dialogue features in the dataset. We compare the Sim-R training dialogues with the DSTC2 (Henderson et al. (2013)) training set which also deals with restaurants and is similarly sized (1611 vs. 1116 dialogues) (Table 2). M2M compares favorably to DSTC2 on the ratio of unique unigrams and bigrams to total number of tokens in the dataset, which signifies a greater variety of surface forms as opposed to repeating the same words and phrases. We also measure the *outline diversity*, defined as the ratio of unique outlines divided by total dialogues in the dataset. We calculate this for sub-dialogues of length $k = \{1, 3, 5\}$ as well as full dialogues. This gives a sense of the diversity of dialogue flows in the dataset. M2M has fewer repetitions of sub-dialogues compared to DSTC2.

[4] https://github.com/google-research-datasets/simulated-dialogue

Table 2: Comparing DSTC2 and M2M Restaurants datasets on diversity of language and dialogue flows.

Metric	**DSTC2 (Train)**	**Sim-R (Train)**
Dialogues	1611	1116
Total turns	11670	6188
Total tokens	199295	99932
Avg. turns per dialogue	14.49	11.09
Avg. tokens per turn	8.54	8.07
Unique tokens ratio	0.0049	0.0092
Unique bigrams ratio	0.0177	0.0670
Outline diversity (k=1)	0.0982	0.2646
Outline diversity (k=3)	0.1831	0.3145
Outline diversity (k=5)	0.5621	0.7061
Outline diversity (full)	0.9243	0.9292

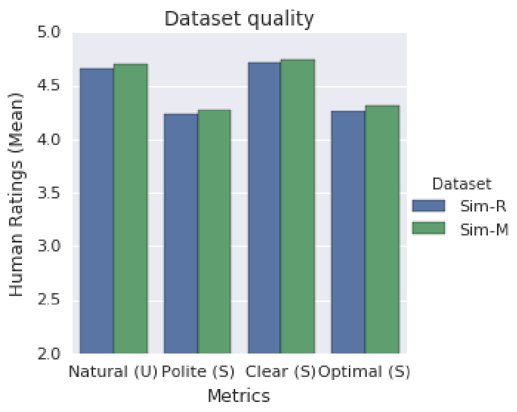


Figure 3: Crowd worker ratings for the quality of the user and system utterances of dialogues collected with M2M.

5.2 Human evaluation of dataset quality

To evaluate the subjective quality of the M2M datasets, we showed the final dialogues to human judges recruited via a crowd-sourcing service, and asked them to rate each user and system turn between 1 to 5 on multiple dimensions. Fig. 6 in the Appendix provides the UI shown to crowd workers for this task. Each dialogue was shown to 3 judges. Fig. 3 shows the average ratings aggregated over all turns for the two datasets.

5.3 Human evaluation of model quality

To evaluate the proposed method of bootstrapping neural conversational agents from a programmed system agent, we trained an end-to-end conversa-

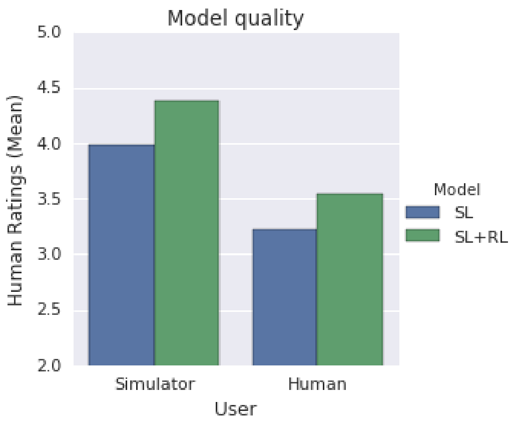


Figure 4: Average crowd worker ratings for the quality of the system utterances of neural conversational agents trained on Sim-M.

tional model (Liu et al. (2017)) using supervised learning (SL) on the Sim-M training set. This model is further trained with RL for 10K episodes with the user simulator as described in Section 2.2 (SL+RL). We performed two separate evaluations of these models:

Simulated user. We evaluate the neural agents in the user simulation environment for 100 episodes. We asked crowd-sourced judges to read dialogues between the agent and the user simulator and rate each system turn on a scale of 1 (frustrating) to 5 (optimal way to help the user). Each turn was rated by 3 different judges. Fig. 4 shows the average scores for both agents. End-to-end optimization with RL improves the quality of the agent according to human judges, compared to an agent trained with only supervised learning on the dataset.

Human user. We evaluate the neural agents in live interactions with human judges for 100 episodes each. The human judges are given scenarios for a movie booking task and asked to talk with the agent to complete the booking according to the constraints. After the dialogue finishes, the judge is asked to rate each system turn on the same scale of 1 to 5. Fig. 4 shows the average scores for both agents. End-to-end optimization with RL improves the agent's interactions with human users. The interactions with human users are of lower quality than those with the user simulator as human users may use utterances or dialogue flows unseen by the agent. Continual training of the agent with on-line reinforcement learning can close this gap with more experience.

6 Related work and discussion

We presented an approach for rapidly bootstrapping goal-oriented conversational agents for arbitrary database querying tasks, by combining dialogue self-play, crowd-sourcing and on-line reinforcement learning.

The dialogue self-play step uses a task-independent user simulator and programmed system agent seeded with a task-specific schema, which provides the developer with full control over the generated dialogue outlines. PyDial (Ultes et al. (2017)) is an extensible open-source toolkit which provides domain-independent implementations of dialogue system modules, which could be extended by adding dialogue self-play functionality. We described an FSM system agent for handling any transactional or form-filling task. For more complex tasks, the developer can extend the user simulator and system agents by adding their own rules. These components could also be replaced by machine learned generative models if available. Task Completion Platform (TCP) (Crook et al. (2016)) introduced a task configuration language for building goal-oriented dialogue interactions. The state update and policy modules of TCP could be used to implement agents that generate outlines for more complex tasks.

The crowd-sourcing step uses human intelligence to gather diverse natural language utterances. Comparisons with the DSTC2 dataset show that this approach can create high-quality fully annotated datasets for training conversational agents in arbitrary domains. ParlAI (Miller et al. (2017)), a dialogue research software platform, provides easy integration with crowd sourcing for data collection and evaluation. However, the crowd sourcing tasks are open-ended and may result in lower quality dialogues as described in Section 4. In M2M, crowd workers are asked to paraphrase given utterances instead of writing new ones, which is at a suitable difficulty level for crowd workers.

Finally, training a neural conversational model over the M2M generated dataset encodes the programmed policy in a differentiable neural model which can be deployed to interact with users. This model is amenable to on-line reinforcement learning updates with feedback from actual users of the system (Su et al. (2016a); Liu et al. (2017)), ensuring that the agent improves its performance in real situations with more experience.

References

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. 2017. Frames: A corpus for adding memory to goal-oriented dialogue systems. *arXiv preprint arXiv:1704.00057* .

Paul Crook, Alex Marin, Vipul Agarwal, Khushboo Aggarwal, Tasos Anastasakos, Ravi Bikkula, Daniel Boies, Asli Celikyilmaz, Senthilkumar Chandramohan, Zhaleh Feizollahi, et al. 2016. Task completion platform: A self-serve multi-domain goal oriented dialogue platform. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. pages 47–51.

Nils Dahlbäck, Arne Jönsson, and Lars Ahrenberg. 1993. Wizard of oz studieswhy and how. *Knowledge-based systems* 6(4):258–266.

Layla El Asri, Romain Laroche, and Olivier Pietquin. 2014. Task completion transfer learning for reward inference. *Proc of MLIS* .

Mehdi Fatemi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman. 2016. Policy networks with two-stage training for dialogue systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. pages 101–110.

Milica Gašić, Filip Jurčíček, Blaise Thomson, Kai Yu, and Steve Young. 2011. On-line policy optimisation of spoken dialogue systems via live interaction with human subjects. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, pages 312–317.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2013. Dialog state tracking challenge 2 & 3. http://camdial.org/~mh521/dstc/.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

John F Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems (TOIS)* 2(1):26–41.

Anjishnu Kumar, Arpit Gupta, Julian Chan, Sam Tucker, Bjorn Hoffmeister, and Markus Dreyer. 2017. Just ask: Building an architecture for extensible self-service spoken language understanding. In *NIPS Conversational AI Workshop*.

Jiwei Li, Alexander H Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. 2016a. Dialogue learning with human-in-the-loop. *arXiv preprint arXiv:1611.09823* .

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

Bing Liu and Ian Lane. 2017a. An end-to-end trainable neural network model with belief tracking for task-oriented dialog. In *Interspeech*.

Bing Liu and Ian Lane. 2017b. Iterative policy learning in end-to-end trainable task-oriented neural dialog models. In *Proceedings of IEEE ASRU*.

Bing Liu, Gokhan Tur, Dilek Hakkani-Tur, Pararth Shah, and Larry Heck. 2017. End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. In *NIPS Workshop on Conversational AI*.

Ryan Lowe, Nissan Pow, Iulian Vlad Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. 2017. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse* 8(1):31–65.

Alexander H Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. 2017. Parlai: A dialog research software platform. *arXiv preprint arXiv:1705.06476* .

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*. Association for Computational Linguistics, pages 149–152.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*. volume 16, pages 3776–3784.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville, and Yoshua Bengio. 2017. A hierarchical latent variable encoder-decoder model for generating dialogues. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Pararth Shah, Dilek Hakkani-Tür, and Larry Heck. 2016. Interactive reinforcement learning for task-oriented dialogue management. In *NIPS Deep Learning for Action and Interaction Workshop*.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. 2018. Building a conversational agent overnight with dialogue self-play. *arXiv preprint arXiv:1801.04871* .

Pei-Hao Su, Paweł Budzianowski, Stefan Ultes, Milica Gasic, and Steve Young. 2017. Sample-efficient actor-critic reinforcement learning with supervised data for dialogue management. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. pages 147–157.

Pei-Hao Su, Milica Gasic, Nikola Mrkšić, Lina M Rojas Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016a. On-line active reward learning for policy optimisation in spoken dialogue systems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. volume 1, pages 2431–2441.

Pei-Hao Su, Milica Gasic, Nikola Mrksic, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016b. Continuously learning neural dialogue management. *arXiv preprint arXiv:1606.02689* .

Pei-Hao Su, David Vandyke, Milica Gašíc, Dongho Kim, Nikola Mrkšíc, Tsung-Hsien Wen, and Steve Young. 2015. Learning from real users: Rating dialogue success with neural networks for reinforcement learning in spoken dialogue systems. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*. volume 2015, pages 2007–2011.

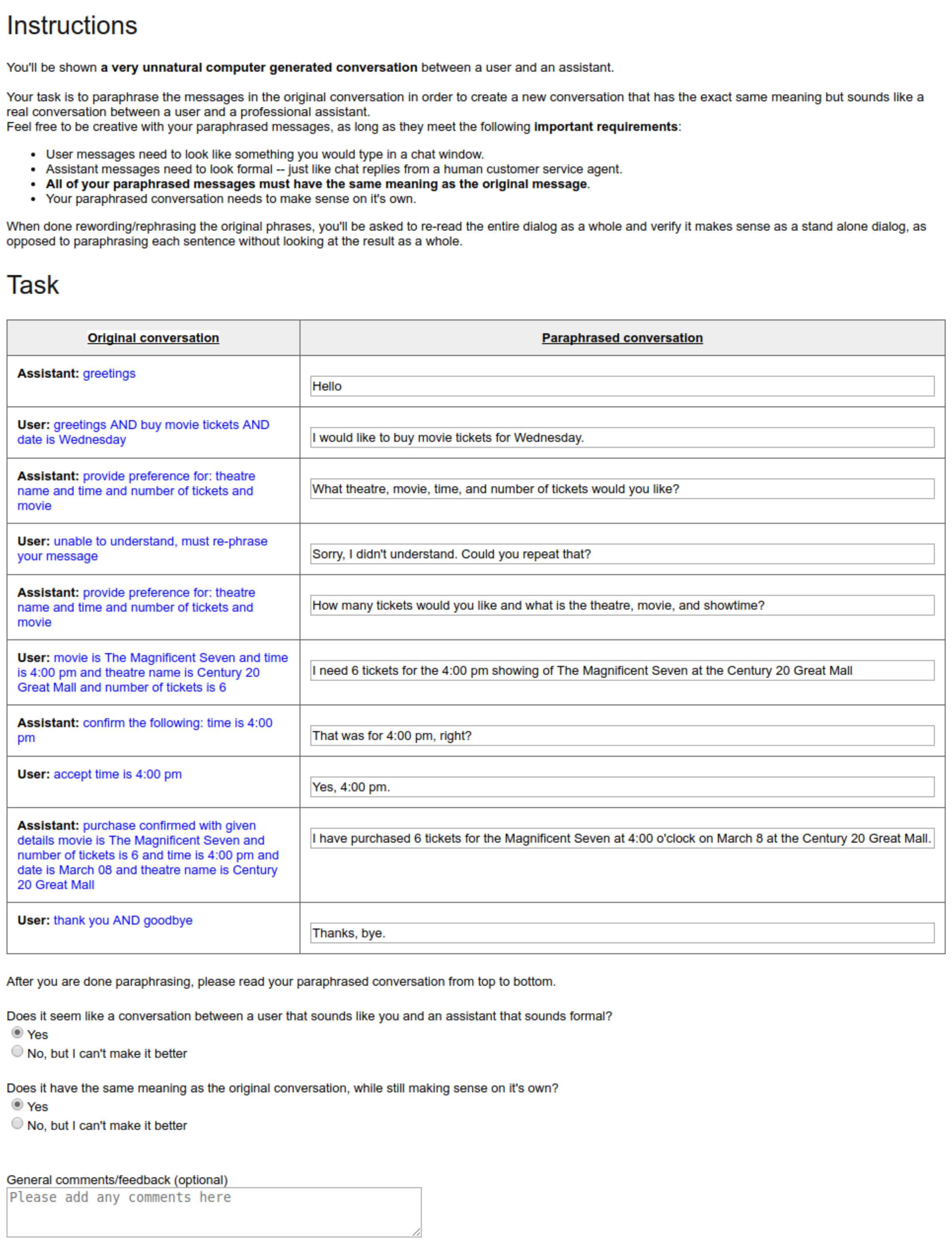
Stefan Ultes, Lina M Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Inigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, et al. 2017. Pydial: A multi-domain statistical dialogue system toolkit. *Proceedings of ACL 2017, System Demonstrations* pages 73–78.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869* .

Yushi Wang, Jonathan Berant, Percy Liang, et al. 2015. Building a semantic parser overnight. *ACL* .

Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2016. A network-based end-to-end trainable task-oriented dialogue system. *ACL* .

Figure 5: Contextual rewrite task interface for paraphrasing a dialogue outline with natural language.



Instructions

You'll be shown **a very unnatural computer generated conversation** between a user and an assistant.

Your task is to paraphrase the messages in the original conversation in order to create a new conversation that has the exact same meaning but sounds like a real conversation between a user and a professional assistant.
Feel free to be creative with your paraphrased messages, as long as they meet the following **important requirements**:

- User messages need to look like something you would type in a chat window.
- Assistant messages need to look formal -- just like chat replies from a human customer service agent.
- **All of your paraphrased messages must have the same meaning as the original message**.
- Your paraphrased conversation needs to make sense on it's own.

When done rewording/rephrasing the original phrases, you'll be asked to re-read the entire dialog as a whole and verify it makes sense as a stand alone dialog, as opposed to paraphrasing each sentence without looking at the result as a whole.

Task

Original conversation	Paraphrased conversation
Assistant: greetings	Hello
User: greetings AND buy movie tickets AND date is Wednesday	I would like to buy movie tickets for Wednesday.
Assistant: provide preference for: theatre name and time and number of tickets and movie	What theatre, movie, time, and number of tickets would you like?
User: unable to understand, must re-phrase your message	Sorry, I didn't understand. Could you repeat that?
Assistant: provide preference for: theatre name and time and number of tickets and movie	How many tickets would you like and what is the theatre, movie, and showtime?
User: movie is The Magnificent Seven and time is 4:00 pm and theatre name is Century 20 Great Mall and number of tickets is 6	I need 6 tickets for the 4:00 pm showing of The Magnificent Seven at the Century 20 Great Mall
Assistant: confirm the following: time is 4:00 pm	That was for 4:00 pm, right?
User: accept time is 4:00 pm	Yes, 4:00 pm.
Assistant: purchase confirmed with given details movie is The Magnificent Seven and number of tickets is 6 and time is 4:00 pm and date is March 08 and theatre name is Century 20 Great Mall	I have purchased 6 tickets for the Magnificent Seven at 4:00 o'clock on March 8 at the Century 20 Great Mall.
User: thank you AND goodbye	Thanks, bye.

After you are done paraphrasing, please read your paraphrased conversation from top to bottom.

Does it seem like a conversation between a user that sounds like you and an assistant that sounds formal?
[✓ Yes]
No, but I can't make it better

Does it have the same meaning as the original conversation, while still making sense on it's own?
[✓ Yes]
No, but I can't make it better

General comments/feedback (optional)
Please add any comments here

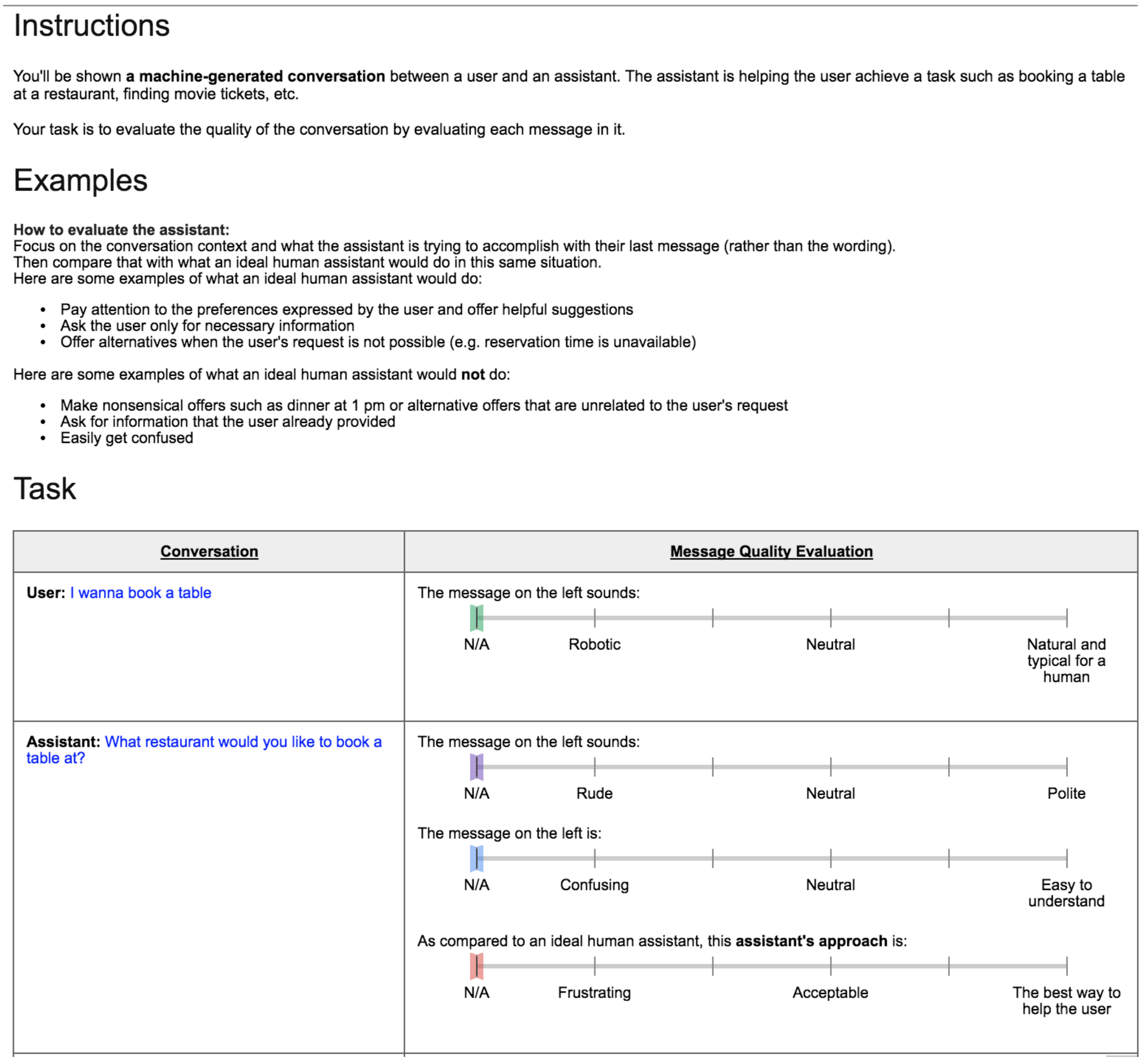


Figure 6: Dialogue quality evaluation task interface for rating the user and system turns of completed dialogues.

Instructions

You'll be shown **a machine-generated conversation** between a user and an assistant. The assistant is helping the user achieve a task such as booking a table at a restaurant, finding movie tickets, etc.

Your task is to evaluate the quality of the conversation by evaluating each message in it.

Examples

How to evaluate the assistant:
Focus on the conversation context and what the assistant is trying to accomplish with their last message (rather than the wording).
Then compare that with what an ideal human assistant would do in this same situation.
Here are some examples of what an ideal human assistant would do:

- Pay attention to the preferences expressed by the user and offer helpful suggestions
- Ask the user only for necessary information
- Offer alternatives when the user's request is not possible (e.g. reservation time is unavailable)

Here are some examples of what an ideal human assistant would **not** do:

- Make nonsensical offers such as dinner at 1 pm or alternative offers that are unrelated to the user's request
- Ask for information that the user already provided
- Easily get confused

Task

Conversation	Message Quality Evaluation
User: I wanna book a table	The message on the left sounds: N/A · Robotic · Neutral · Natural and typical for a human
Assistant: What restaurant would you like to book a table at?	The message on the left sounds: N/A · Rude · Neutral · Polite The message on the left is: N/A · Confusing · Neutral · Easy to understand As compared to an ideal human assistant, this **assistant's approach** is: N/A · Frustrating · Acceptable · The best way to help the user