BotSIM: An End-to-End Bot Simulation Toolkit for Commercial Task-Oriented Dialog Systems

Guangsen Wang  Shafiq Joty  Junnan Li  Steven C.H. Hoi
Salesforce Research
{guangsen.wang, sjoty, junnan.li, shoi}@salesforce.com

Abstract

We introduce BotSIM, a modular, open-source Bot SIMulation environment with dialog generation, user simulation and conversation analytics capabilities. BotSIM aims to serve as a one-stop solution for large-scale data-efficient end-to-end evaluation, diagnosis and remediation of commercial task-oriented dialog (TOD) systems to significantly accelerate commercial bot development and evaluation, reduce cost and time-to-market. BotSIM adopts a layered design comprising the infrastructure layer, the adaptor layer and the application layer. The infrastructure layer hosts key models and components to support BotSIM’s major functionalities via a streamlined “generation-simulation-remediation” pipeline. The adaptor layer is used to extend BotSIM to accommodate new bot platforms. The application layer provides a suite of command line tools and a Web App to significantly lower the entry barrier for BotSIM users such as bot admins or practitioners. In this report, we focus on the technical designs of various system components. A detailed case study using Einstein BotBuilder is also presented to show how to apply BotSIM pipeline for bot evaluation and remediation. The detailed system descriptions can be found in our system demo paper (Wang et al., 2022). The toolkit is available at: https://github.com/salesforce/BotSIM.

1 Introduction

The typical dialog system development cycle consists of dialog design, pre-deployment testing, deployment, performance monitoring, model improvement and iteration. As in any production software system, effective and comprehensive testing at all stages is of paramount importance. Unfortunately, evaluating and troubleshooting production TOD systems is still a largely manual process requiring large amount of real human conversations with the bots. This process is time-consuming, expensive, and inevitably fails to capture the breadth of language variation present in the real world (Tan et al., 2021). The time- and labor-intensive nature of such an approach is further exacerbated when the developer significantly changes the dialog flows, since new sets of test dialogs will need to be collected (Benvie et al., 2020). Performing comprehensive end-to-end evaluation of both NLU and dialog-level performance (e.g., task success rate) is highly challenging due to the need for further annotated efforts. Finally, there is a lack of analytical tools for interpreting test results and troubleshooting underlying bot issues. To address these limitations, we present BotSIM, an open source Bot SIMulation environment with dialog generation, user simulation (Schatzmann et al., 2007) and conversation analytics capabilities. BotSIM aims to serve as a one-stop solution for bot practitioners to perform large-scale data-efficient end-to-end evaluation, diagnosis and remediation of commercial TOD systems. The modular design also allows bot developers to extend the framework to more bot platforms.

The toolkit design is depicted in Figure 1. It consists of three layers: the infrastructure layer, the adaptor layer and the application layer. The infrastructure layer supports the upper layers through the key modules (NLU and NLG) and components (the generator, simulator and remediator). They are the “engines” to power BotSIM’s “generation-simulation-remediation” pipeline for end-to-end diagnosis, evaluation and remediation of commercial TOD bots. More details regarding the functionalities of these modules and components are given in (Wang et al., 2022). BotSIM currently supports two bot platforms, including Salesforce Einstein BotBuilder and Google DialogFlow CX. The adaptor layer is designed for bot developers to extend BotSIM to new bot platforms. In addition to defining a set of unified parser and client interfaces, the adaptor layer also provides some reference implementations for developers to start their
2 Related Work

We present the related work of BotSIM from both toolkit design and application perspectives.

2.1 Toolkit Design

To the best of our knowledge, there are no open source toolkits designed for simulation, evaluation and remediation of commercial chat bots. Therefore, we review some of the research toolkits/libraries designed for TOD systems and are equipped with user simulators. They include ConvLab-2 (Zhu et al., 2020), TC-Bot (Li et al., 2016, 2017) and user-simulator (Shi et al., 2019).

- **ConvLab-2**[^1] is a research toolkit for building TOD systems with state-of-the-art dialog models. It was used in DSTC9 Track 2[^2] for the multi-domain TOD challenge. In addition, it also can perform end-to-end evaluation, and diagnose the weakness of the research systems. The essential differences to BotSIM include 1) the toolkit is limited to research TOD systems to support research models and datasets; 2) the evaluation is based on the annotated dialogs.

- **TC-Bot**[^3] is another research-oriented framework for building end-to-end neural-based TOD systems. TC-Bot utilises a dialog-act level agenda-based user simulator to interact with the neural-based end-to-end dialog agent for policy training. Some of our simulator implementations of the platform-dependent parsers and chat API clients. The application layer offers a suite of command line tools and an easy-to-use Web App. The Web app is designed for bot admins to directly apply BotSIM to their bots for evaluation and remediation without diving into the technical details of various system components. On the other hand, the command line tools are provided to bot practitioners for a better understanding of the toolkit to customize their own evaluation and remediation pipeline. They can learn more about how different BotSIM components work in terms of the required inputs, expected outputs and their functionalities. We have provide detailed tutorials of how to use these tools in our code documentation.

![Figure 1: BotSIM System Design.](image)

[^1]: https://github.com/thu-coai/ConvLab-2
[^2]: https://convlab.github.io/index.html
[^3]: https://github.com/MiuLab/TC-Bot
### 2.2 Commercial Bot Evaluation

The overall comparison of different platforms in terms of testing capabilities is given in Table 1. The detailed system reviews can be found in the system demo paper (Wang et al., 2022). Most current commercial platforms only focus on the regression testing rather than NLU or task-completion metrics. While regression testing is important to ensure correct and consistent system behaviours, it is also vital to perform pre-deployment evaluation to avoid poor user adoption and retention rate. Although some of the platforms offer turn-level NLU performance metrics, they require human efforts in curating or annotating a large number of test cases. In addition, the NLU metrics do not directly translate to goal completion performance. On the other hand, BotSIM can help circumvent these limitations via large scale automatic dialog generation and simulation.

#### 3 BotSIM Design

In this section, we present more details of the BotSIM design in Figure 1. The key design principles of BotSIM include modularity, extensibility and usability. These principles allow BotSIM to be adopted both by developers as a framework and bot end-users as an easy-to-use application. To achieve these, BotSIM adopts a layered design comprising the infrastructure layer, the adaptor layer and the application layer.

##### 3.1 Infrastructure Layer

As the name suggests, the infrastructure layer is designed to offer fundamental model support for the framework. BotSIM’s “generation-simulation-remediation” pipeline is powered by the models and components that reside in this layer. The models include the natural language understanding (NLU), natural language generation (NLG) models and the key components include the generator, the simulator and the remediator.

botsim.models package hosts BotSIM’s NLU and NLG models. From a dialogue system perspective, BotSIM can be viewed as a counterpart to a chatbot as shown in Figure 2: it needs to “understand” chatbot messages (NLU) and “respond” in natural languages (NLG) to carry on the conversation. Currently, fuzzy matching-based NLU and template-based NLG models are provided for efficiency reasons. More advanced NLU and NLG models can also be incorporated by the developers by following the recipes given in the code documentation.

botsim.modules consists of the three key components to power BotSIM’s “generation-simulation-remediation” pipeline.

---

**Table 1: Comparison of bot evaluation capabilities of the reviewed commercial bot platforms**

| Methods     | Stages                      | Automation | Metrics       |
|-------------|-----------------------------|------------|---------------|
|             | Regression                  | End-to-end | Pre-deployment | Monitoring | Test case curation | User Simulation | NLU | Task Completion |
| CX          | ✓                           | ✓          |               |            | ✓                  | ✓              | ✓   | ✓              |
| Watson       | ✓                           |            |               |            |                    | ✓              | ✓   | ✓              |
| Botium      | ✓                           | ✓          | ✓             |            |                    | ✓              | ✓   | ✓              |
| BotSIM      | ✓                           | ✓          | ✓             | ✓          |                    | ✓              | ✓   | ✓              |
The adaptor layer is designed for bot developers to unify bot definitions in the form of either metadata (Einstein BotBuilder) or API (DialogFlow CX) to infer the dialog-act maps (BotSIM’s NLU); 2) the large pre-trained language model based paraphraser generates paraphrases from the input intent utterances. These paraphrases are used as intent queries in the simulation goals to probe bots’ intent models, which allows BotSIM to perform large-scale data-efficient bot evaluation even before bots are deployed.

**botsim.modules.remediator** implements the dialog-act level agenda-based user simulation in abus. It also defines a simulation API client interface `simulation_client_base`.

**botsim.modules.simulator** implements a simulation API for bots. It provides two major functionalities: 1) the parser parses the input bot definitions (e.g. conversation flows, intents/tasks) from different platforms to a common representation of dialog act maps. The dialog act maps are used as BotSIM NLU to map bot messages to dialog acts. The rational is that most commercial TOD bots follow a “rule-action-message” design scheme and there exist clear mappings from system messages to rules/actions. For example, according to bot definitions, “May I get your email?” (bot message) is used to “Collect” (action) the “Email” (slot) with entity type “Email” from the user. One of the most important parser function is to parse input bot definitions (e.g., MetaData, API) to capture such mappings, infer the “request_Email” dialog act and add the bot message to the mapping candidates of the dialog act. An example of the dialog act map is given in Figure 3. The major parser functions are listed in the code snippet below:

```python
class Parser:
    def __init__(self, config):
        # The aggregated dialog act maps to be used
        self.dialog_act_maps = {}
        # mapping from dialogs to their entities
        self.dialog_ontology = {}
        # conversion graph data to support path explorer
        self.conv_graph_visualisation_data = {}

    def extract_local_dialog_act_map(self):
        # Model local dialog act maps and their transitions as a graph
        def extract_local_dialog_act_map(self):
            return: dialog_act_maps: aggregated dialog act maps obtained via graph traversal. These maps need to be reviewed/revised by users before used as BotSIM NLU
            conv_graph_visualisation_data: conversation graph data to support dialog path visualisation

    def extract_ontology(self):

Note the implementations of the parsers are highly platform-dependent. They require the developers to have access to bot platform, design and API documents to understand how bots are designed and how user inputs are elicited by the bots. There is also need to constantly revisit the implementations to incorporate new features that might be missed in the current implementations. We have provided our current implementations of BotBuilder and DialogFlow CX parsers for references under botsim.platforms.botbuilder and botsim.platforms.dialogflow_cx.

**simulation_client** is the other platform-dependent component for BotSIM to exchange conversations with bots via API calls.

```python
from botsim.modules.simulator.user_simulator import UserSimulator

class UserSimulatorClientInterface:
    def __init__(self, config):
        self.config = config

    def prepare_simulation(self):
        ***Prepare simulation according to the simulation configuration***

    def perform_batch_simulation(self, simulation_goals, dialog_name, start_episode, config, batch_size=25):
        ***Perform a batch of dialog user simulations using simulation goals***
        def perform_batch_simulation(self, simulation_goals, dialog_name, start_episode, config, batch_size=25):
            user_simulator = UserSimulator(test_goals, config)
```

3.2 **Adaptor Layer**

The adaptor layer is designed for bot developers to extend BotSIM to new bot platforms. To cover the new bot platforms, the following two most important platform-specific modules of the layer must be implemented. We also provided more details in the code documentation. To accommodate a new bot platform, create a new package under botsim.platforms and implement the following classes.

**parser** acts as an “adapter” to unify bot definitions (e.g., conversation flows, intents/tasks) from different platforms to a common representation of dialog act maps. The dialog act maps are used as BotSIM NLU to map bot messages to dialog acts. The rational is that most commercial TOD bots follow a “rule-action-message” design scheme and there exist clear mappings from system messages to rules/actions. For example, according to
Figure 2 depicts how a dialogue turn between the bot and BotSIM is conducted via bot APIs. Using dialog act maps as NLU, a rule-based dialog state manager (policy implemented in botsim.modules.simulator) takes in the bot dialog acts and produces the corresponding user dialog acts. The user dialog acts are converted to natural language responses by the template-based NLG. The natural language responses are sent back via API. The conversation ends when the task has been successfully finished or an error has been captured. Similar to the parser, the implementation of the client is also highly platform-dependent. Developers can refer to our implementations for BotBuilder and DialogFlow CX when extending BotSIM to new bot platforms.

3.3 Application Layer

The application layer is designed to significantly flatten the learning curves of BotSIM for both bot developers/practitioners and end-users.

Command Line Tools botsim.cli contains a set of command line tools for practitioners to learn more about the major BotSIM components. The “generation-simulation-evaluation” pipeline has been split into multiple stages to expose the required inputs and expected outputs for each stage. They serve as basic building blocks for bot practitioners to build their customized pipelines or apply only certain tasks rather than the whole BotSIM pipeline.

1 python botsim/cli/run_generator_paraphraser.py \
2   --platform platform \
3   --test_name $name
4 python botsim/cli/run_generator_paraphraser.py \
5   --platform platform \
6   --test_name $name
7 python botsim/cli/run_generator_goal_generation.py \
8   --platform platform \
9   --test_name $name
10 python botsim/cli/run_simulator.py \
11   --platform platform \
12   --test_name $name
13 python botsim/cli/run_remediator.py \
14   --platform $platform \
15   --test_name $name

Streamlit Web App botsim.streamlit_app is a multi-page easy-to-use Web app for end users such as bot admins without diving into technical details. The app can be built as a docker image and deployed to various cloud platforms (e.g., GCP) for access to more computation resources. We use Streamlit 6 to build the front-end pages. Flask is used to implement the backend APIs for Streamlit to invoke BotSIM functionalities. The app is also equipped with a SQL database to keep track of simulation stages and simulation performance. BotSIM supports two types of SQL databases including Sqlite3 and Postgres.

4 Einstein BotBuilder Case Study

In this section, we show how to perform end-to-end evaluation and remediation of the pre-built “Template Bot” from the Salesforce Einstein BotBuilder platform. We follow BotSIM’s “generation-simulation-remediation” pipeline as detailed in (Wang et al., 2022). The “Template Bot” has six dialog intents. Each intent has a set of hand-crafted training utterances. For controllable experiments, we sample 150 utterances per dialog as the training set (train-original) and use the remaining for evaluation (eval-original). The six intents are: “Transfer to agent (TA)”, “End chat (EC)”, “Connect with sales (CS)”, “Check issue status (CI)”, “Check order status (CO)” and “Report an issue (RI)”. The dataset information is given in Table 2.

4.1 Parse bot metadata

The required inputs for BotSIM include 1) bot design metadata containing the bot designs (e.g., intents/dialogs, entities) 2) intent utterance metadata 3) LiveAgent API information. They can be retrieved from users’ Salesforce org. BotSIM starts by parsing the input metadata and generates the NLU (Dialog Act Maps) and NLG (Response) templates needed for dialog simulation.

4.2 Revise dialog act maps and ontology

The generated NLU dialog act maps convert bot messages to dialog acts via fuzzy matching during dialog simulation. An example of dialog act maps is shown in Figure 3. The aggregated dialog act map is inferred automatically by the parser by modelling the bot design as graphs: 1) individual dialog acts are firstly parsed to get their “local” dialog act maps; 2) individual dialog acts (including sub-dialogs such as “Case Lookup”) are modelled as vertices associated with their “local” dialog act maps; 3) dialog transitions of the entire bot are modelled as graph edges. The final aggregated dialog act map of a dialog is created by collecting all the “local” dialog act maps along the paths starting from the dialog node to the successful dialogs (e.g., “End_Chat”). Meanwhile, the parser also extracts the bot entity ontology. It lists all entities and their randomly initialized values for each dialog.

—

6https://streamlit.io/
As the only “human-in-the-loop” step in the BotSIM pipeline, review or revision of the automatically inferred dialog act maps and ontology are required by BotSIM users. For dialog act maps, the two special dialog acts, “dialog_success_message” and “intent_success_message”, are the golden labels indicating a successful task completion and a correct intent classification, respectively. They are inferred heuristically by regarding the first dialog message as the “intent_success_message” and the last message as the “dialog_success_message”. Users are required to verify these two dialog acts for each evaluation dialog to ensure their correctness. Note the review of dialog act maps is a one-time effort unless there are significant changes made to the bot design. Since the entity values of the ontology are mostly related to users’ products and services, the randomly initialised values in the ontology file may be replaced with real ones in order to evaluate the named entity recognition (NER) model. The revised dialog act maps and the ontology are subsequently used to create the simulation goals as in Figure 4.3 for dialog simulation.

### 4.3 Paraphrase and goal generation

Agenda-based user simulation requires pre-defined goals to ensure that the simulated user behaves in a consistent, goal-directed manner (Schatzmann et al., 2007). The goals consist of a set of constraints (dialog acts and entity slot-value pairs) needed to complete the task. An exemplar simulation goal for “check the status of an existing issue” is given below:

```json
{
  "Goal": {
    "Check_the_status_of_an_existing_issue": {
      "inform_slots": {
        "Has_case_number@_Boolean": "no",
        "Case_Number@Case_Number_Entity": "P94RMU",
        "Email_for_Look_Up@Email_Address_Entity": "andrewadams@example.org",
        "intent": "I would like to know about my issue.",
        "Needs_transfer_to_agent@_Boolean": "no",
        "Needs_something_else@_Boolean": "no",
        "Goodbye@Goodbye_Entity": "goodbye",
        "Needs_to_add_case_comment@_Boolean": "no",
        "Case_Comments@_Text": "P94RMU"
      },
      "request_slots": {
        "Check_the_status_of_an_existing_issue": "UNK"
      },
      "name": "Check_the_status_of_an_existing_issue"
    }
  }
}```
Note how the entity slots are taken from the dialog act maps in Figure 3. In particular, the “intent” slot contains the intent query to test the intent models. To enable pre-deployment testing without any human-written test cases, we apply the paraphrasing models to the “train-original” utterances to get the “train-paraphrases” dataset. Compared to the number of original utterances, the “train-paraphrases” dataset is roughly ten times larger as shown in Table 2. Simulation goals are created by taking the “train-paraphrases” as the intent queries to simulate the variations in real user intent queries. The “train-paraphrases” goals are subsequently used as the development set to perform end-to-end evaluation of the dialog system via dialog simulation.

4.4 Dialog simulation results

In this study, we focus on the intent model for two reasons. Firstly, the intent model can be retrained. Secondly, since we do not have any customer data, the entity values in the goals are randomly generated and may not reflect the real-world values. After dialog simulation with “train-paraphrases” goals, the original intent training sets are augmented with the wrongly classified intent queries (“train-augmented”) to re-train the intent model according to the remediation suggestions. We then compare the performance before and after retraining on the goals created from the human-written “eval-original” utterances in Table 3 in terms of intent F1 scores.

We observe consistent improvements for all intents after model retraining. The more challenging the intents (lower F1s) are, e.g., “Report an issue” and “Connect with sales”, the larger performance
gains are observed compared to the easier intents such as “End Chat” (higher F1s). This demonstrates the efficacy of BotSIM in intent model improvement. We conjecture the improved performance of the more challenging intents is due to more paraphrases being selected for retraining to better cover the language variation.

4.5 Diagnosis and Remediation Dashboard

The remediator generates health diagnosis reports, performs analyses, and provides actionable recommendations to troubleshoot and improve dialog systems. The major dashboard components are presented in Figure 4, 5, 6, 7.

Bot health reports. The bot “health” dashboard consists of a set of multi-scale performance reports. At the highest level, users can have a historical view of most recent simulation/test sessions (e.g., after each major bot update). The historical performance comparison can help users evaluate the impacts of bot changes quantitatively, from which they can make decisions like whether or not keep certain changes. In the test session performance summary report, users can have the details of a selected test session including the data distribution, overall dialog performance metrics across all dialog intents of the test session. The dialog-specific performance report provides detailed detailed intent and NER performance of the selected dialog intent. Through the dialog-specific performance report, one can quickly identify the most confusing intents and entities. This saves significant efforts and helps better allocation of resources for troubleshooting and bot improvement.

Remediation recommendations. In addition to the diagnosis reports, the remediator also provides actionable insights/suggestions for users to remedy some of the identified issues. The root causes of the failed conversations are identified via backtracking of the simulation agenda. The recommendation dashboards (Figure 5 and Figure 6) allow detailed investigation of all intent or NER errors along with their corresponding simulated chat logs. For intent models, the paraphrase intent queries that lead to intent errors are grouped by the original intent utterances. These original intent utterances are sorted by the number of intent errors of their paraphrases in descending order (drop-down list of the Remediation suggestions in Figure 5). Depending on the wrongly classified intents, the remediator would suggest some follow-up actions. For example, 1) augmenting the intent training set with the queries deemed to be out-of-domain by the current intent model, 2) moving the intent utterance to another intent if most of paraphrases of the former intent utterance are classified to the latter intent. Note the suggestions are meant to be used as guidelines rather than strictly followed. More importantly, they can always be extended by users to include domain expertise in troubleshooting bots related to their products/services.

Conversation analytics. Another useful component of the Remediator is the suite of conversation analytical tools as shown in Figure 7. They further help bot practitioners gain more insights for troubleshooting and improving their dialog systems. The confusion matrix analysis breaks down the intent model performance into (sortable) recall, precision and F1 accuracies to help identify the worse performing intents. It also detects potential intent overlaps using the clustering algorithms in (Thoma, 2017) based on the performance metrics. Another useful analytical tool is the tSNE (van der Maaten and Hinton, 2008) clustering of the intent utterances using sentence transformer (Reimers and Gurevych, 2019) embeddings. The tSNE visualisation enables users to gauge the training data quality. It is also an effective tool in identifying overlapping intents and can potential benefit new intent discovery as well.

Dialog path explorer. Lastly, powered by parsers’ conversation graph modelling capability,
the dialog path explorer can be used to visualise different dialog paths of the current bot design. For example, users can select the “source” and “target” dialogs and explore the generated dialog paths. Not only is the tool valuable for comprehensive testing coverage of conversation paths, it also offers a controllable approach to troubleshooting dialog design related errors or even helping improve the current bot design.

5 Conclusion

We presented the design of BotSIM, an open source, modular end-to-end bot simulation toolkit for evaluation, diagnosis and remediation of commercial TOD systems. Through the streamlined “generation-simulation-remediation” pipeline, BotSIM can be adopted to accelerate commercial bot development and evaluation, reduce cost and time-to-market. Thanks to the layered design, not only can it be used as a framework for bot developers to extend BotSIM to new bot platforms, it also offers a suite of easy-to-use command line tools and Web App for bot practitioners to directly apply BotSIM to their bots. We have open-sourced the toolkit at https://github.com/salesforce/BotSIM including the Streamlit Web App. We also provides detailed documentations to accompany the code. We welcome contributions from the community to help improve and extend BotSIM further.

6 Limitations

For efficiency reasons, BotSIM adopts a template-based NLG model for converting user dialog acts to natural languages. Although the template-NLG is more controllable and flexible compared to the model-based NLG, they may lack naturalness. One possible future improvement includes a combination of template-based NLG and the model-based NLG. For example, we can train a model-based NLG to generate templates (Wiseman et al., 2018) for BotSIM’s response templates. In this way, both efficiency and naturalness can be achieved. Currently, BotSIM is trained and evaluated utilizing English text. We leave multi-lingual bot simulation capability as one of our future works.

7 Broader Impact

The pretrained language-model based paraphrasers (T5-base and Pegasus) used in this study are pretrained and finetuned with large scale of text corpora scraped from the web, which may contain biases. These biases may even be propagated to the generated paraphrases, causing harm to the subject of these stereotypes. Although the paraphrasing models are only applied to generate the testing intent queries, BotSIM users are advised to take into consideration these ethical issues and may wish to manually inspect or otherwise filter the generated paraphrases. It is also noteworthy that to prevent any data privacy leakage, the information produced in the simulation (the entity values in the BotSIM ontology) is randomly generated, and therefore fake. This includes the email addresses, names.

References

Adam Benvie, Eric Wayne, and Matthew Arnold. 2020. Watson assistant continuous improvement best practices.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Inigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. Multiwoz – a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling.

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

Xiujun Li, Zachary C. Lipton, Bhuwan Dhingra, Lihong Li, Jianfeng Gao, and Yun-Nung Chen. 2016. A user simulator for task-completion dialogues. CoRR, abs/1612.05688.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Celikyilmaz. 2017. End-to-end task-completion neural dialogue systems. In Proceedings of The 8th International Joint Conference on Natural Language Processing.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics: Companion Volume, Short Papers, pages 149–152, Rochester, New York. Association for Computational Linguistics.
Weiyan Shi, Kun Qian, Xuewei Wang, and Zhou Yu. 2019. How to build user simulators to train rl-based dialog systems. arXiv preprint arXiv:1909.01388.

Samson Tan, Shafiq Joty, Kathy Baxter, Araz Taeihagh, Gregory A. Bennett, and Min-Yen Kan. 2021. Reliability testing for natural language processing systems. 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), pages 4153–4169, Online. Association for Computational Linguistics.

Martin Thoma. 2017. Analysis and optimization of convolutional neural network architectures.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. Journal of Machine Learning Research, 9(86):2579–2605.

Guangsen Wang, Samson Tan, Shafq Joty, Guang Wu, Jimmy Au, and Steven Hoi. 2022. Botsim: An end-to-end bot simulation framework for commercial task-oriented dialog systems.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2018. Learning neural templates for text generation. CoRR, abs/1808.10122.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. ConvLab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.
One of the major applications of BotDR is to perform end-to-end bot performance evaluation to understand NLU and the goal-completion metrics of an NLU bot. This dashboard presents bot health reports and remediation recommendations generated from bot evaluation via large scale dialog simulation. Insights can be drawn from a collection of reports and analytical tools to troubleshoot and improve bot systems.

| test_id | updated_at | total_episodes | success_rate(%) | intent_error_rate(%) | NER_error_rate(%) | other_error_rate(%) |
|---------|-------------|----------------|------------------|----------------------|-------------------|---------------------|
| 0       | 2022-07-20 13:28:13 | 9674 | 54 | 13 | 10 | 1 |
| 1       | 2022-07-20 13:28:13 | 9677 | 56 | 12 | 10 | 2 |
| 2       | 2022-07-20 13:28:13 | 1112 | 60 | 11 | 10 | 1 |
| 3       | 2022-07-20 13:28:13 | 1115 | 63 | 7 | 17 | 1 |

(1) Historical performance comparison of testing sessions

Bot health reports

The bot health reports consist of a summary report and per-intent reports to show both the goal-completion and NLU performance.

Performance summary for selected test [test_id=4]:

Intent dataset distribution and overall performance

(2) Test session performance summary

Performance report for selected dialog "Check the status of an existing issue"

| #Sessions | F1 score | Goal-completion Rate | Intent Error Rate | NER Error Rate | Other Error Rate |
|-----------|----------|-----------------------|-------------------|----------------|-----------------|
| 2172      | 0.912    | 0.61                  | 0.14              | 0.25           | 0.0             |

Intent and NER performance breakdown for Check the status of an existing issue

(3) Detailed dialog performance

Figure 4: Remediator dashboard: bot health reports.
Figure 5: Remediator dashboard: intent model remediation suggestions.
Figure 6: Remediator dashboard: NER remediation suggestions.
Figure 7: Remediator dashboard: Conversational analytical tools.