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

Guangsen Wang[♣] **Samson Tan**[◇*] **Shafiq Joty**[♣] **Gang Wu**[♣] **Jimmy Au**[♣] **Steven Hoi**[♣]
[♣]Salesforce Research
[◇]AWS AI Research & Education
{guangsen.wang, sjoty, jimmy.au, gang.wu, shoi}@salesforce.com

Abstract

We present BotSIM, a data-efficient end-to-end **Bot SIM**ulation framework for commercial text-based task-oriented dialog (TOD) systems. BotSIM consists of three major components: 1) a *Generator* that can infer semantic-level dialog acts and entities from bot definitions and generate user queries via model-based paraphrasing; 2) an agenda-based dialog user *Simulator* (ABUS) to simulate conversations with the dialog agents; 3) a *Remediator* to analyze the simulated conversations, visualize the bot health reports and provide actionable remediation suggestions for bot troubleshooting and improvement. We demonstrate BotSIM's effectiveness in end-to-end evaluation, remediation and multi-intent dialog generation via case studies on two commercial bot platforms. BotSIM's "generation-simulation-remediation" paradigm accelerates the end-to-end bot evaluation and iteration process by: 1) reducing manual test cases creation efforts; 2) enabling a holistic gauge of the bot in terms of NLU and end-to-end performance via extensive dialog simulation; 3) improving the bot troubleshooting process with actionable suggestions. A demo of our system can be found at https://tinyurl.com/mryu74cd and a demo video at https://youtu.be/qLi5iSoly30.

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 human conversations with the systems. This process is time-consuming, expensive, and inevitably fails to capture the breadth

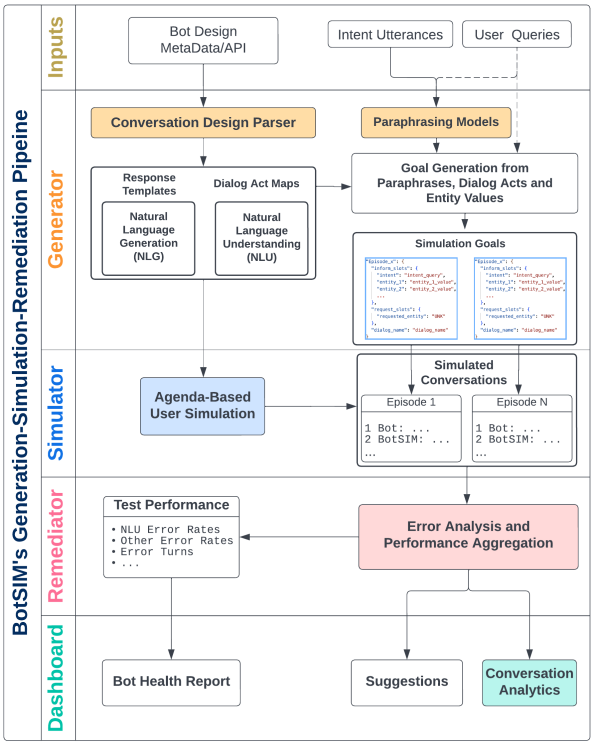

Figure 1: BotSIM overview including the generator, simulator, and remediator. The dotted (optional) paths from users can be used for bot performance monitoring: they can provide production chat logs or manually crafted utterances when creating evaluation goals.

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 created (Benvie et al., 2020). Performing comprehensive end-to-end bot evaluation is highly challenging due to the need for additional annotation 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*, a **Bot SIM**ulation environment for data-efficient end-to-end commercial bot evaluation, remediation via multi-intent dialog generation and agenda-

[*]Work done at Salesforce Research.

Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 178 - 190

based dialog user simulation (Schatzmann et al., 2007). BotSIM consists of three major modules, namely Generator, Simulator, and Remediator (Figure 1). We use a pretrained sequence-to-sequence T5 model (Zhang et al., 2019; Raffel et al., 2020) in the Generator to simulate lexical and syntactic variations in user queries via paraphrasing. The Generator is also responsible to generate various templates needed by the Simulator. To make BotSIM more platform- and task- agnostic, we adopt dialog-act level ABUS to simulate conversations with bots via APIs. The dialog acts are automatically inferred by the Generator via a unified interface to convert bot designs of different platforms to a universal graph representation. The graph has all dialogs as nodes and their transitions as edges. Through graph traversal, BotSIM offers a principled and scalable approach to generating and exploring multi-intent conversations. Not only can the conversation path generation greatly increase evaluation coverage for troubleshooting dialog errors caused by faulty designs (*e.g.,* unexpected dialog loops), it is also valuable for bot design improvements. The Remediator summarizes bots' health status in a dashboard for easy comprehension. It also analyzes the simulated conversations to identify any issues and further provides actionable suggestions to remedy them.

BotSIM's "generation-simulation-remediation" paradigm can significantly accelerate bot development and evaluation, reducing human efforts, cost and time-to-market. Our contributions include:

- We propose BotSIM, a modular, data-efficient bot simulation framework. To the best of our knowledge, this is the first work focused on end-to-end evaluation, diagnosis and remediation of commercial bots via ABUS.
- BotSIM offers a principled approach to generating and simulating multi-intent dialogs for comprehensive evaluation coverage and better bot design.
- We finetuned a T5 paraphrasing model on par with the state-of-the-art performance to generate diverse user responses for greater test coverage of language variation.
- An easy-to-use Streamlit[1] Web App with Flask back-end and SQL database is developed for bot practitioners. The App can be deployed as a docker container or to Heroku[2].

[1] https://streamlit.io/

[2] https://www.heroku.com

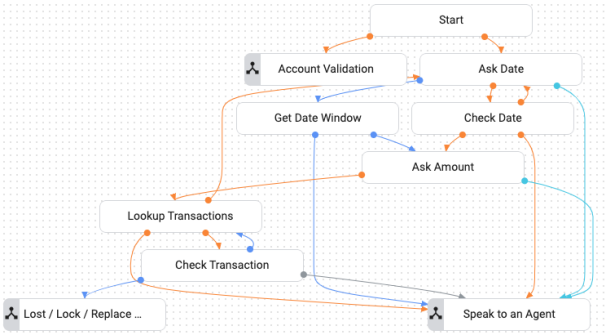

Figure 2: "Investigate Charges" flow of the DialogFlow CX pre-built "Financial Service Agent" mega-agent

2 Related Work

There are two main categories of dialog user simulators, namely the agenda-based user simulator (ABUS) (Schatzmann et al., 2007; Li et al., 2016; Shi et al., 2019; Zhu et al., 2020; Liu et al., 2021; Shah et al., 2018) and recent neural-based user simulator (NUS) (Asri et al., 2016; Crook and Marin, 2017; Kreyssig et al., 2018; Gur et al., 2018; Liu et al., 2017). Since BotSIM is designed to support commercial bot evaluation and remediation, we focus on the review of the testing capacities offered by some existing bot platforms rather than the simulators. Recently, ABUS is also used in Amazon's Alexa conversation (Acharya et al., 2021) for training an end-to-end dialog agent, which is also beyond the scope of our discussion.

2.1 IBM Watson Assistant

IBM Watson assistant offers a suite of open-source Python libraries and notebooks to help analyze customer bots using manually created or annotated test cases (Benvie et al., 2020). An exemplar test case used for the standard regression testing is given in Table 2. Given the annotated conversations, the notebooks offer some analytical functions to compute two metrics, namely coverage (NLU) and effectiveness (task completion) to monitor the bot performance. However, the manual annotation and analysis still require significant expertise and involvement of bot teams. Large scale automatic pre-deployment performance evaluation and analysis are also infeasible since there may not be enough chat logs.

2.2 Google DialogFlow CX

CX offers a built-in regression testing environment for users to create test dialogs and perform regression testing. To create "golden" test cases, users

	Methods		Stages		Automation		Metrics	
	Regression	End-to-end	Pre-deployment	Monitoring	Test case curation	User Simulation	NLU	Task Completion
CX	✓			✓				
Watson	✓			✓			✓	✓
Botium	✓						✓	
BotSIM	✓	✓	✓	✓	✓	✓	✓	✓

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

User:	May I book a flight to New York?
Labels	**Intent: #flight**
	Entity: @Destination
Bot:	When would you like to depart?

Table 2: Example of IBM Watson assistant test case

need to manually chat with the bot and annotate each system turn with correct intent, entity and dialog transitions. During regression testing, each bot response is matched against the golden labels to detect regressions. To achieve good regression testing coverage, users have to "design" testing dialogs to cover as many conversation paths as possible. However, the number of paths grows exponentially with the number of intents and dialog branches, making it almost impossible to craft testing cases for all paths (Figure 2). For end-to-end performance evaluation, even more annotated dialogs are needed to cover the language variation in user responses, which will greatly intensify the manual efforts.

2.3 Botium: Bots Testing Bots

Botium[3] offers a unified platform for regression testings of various bot platforms via platform-specific connectors and "Botium scripts (test cases)" (Appendix A). The core component, dubbed as "Botium Box", analogous to a bot testing "IDE", can be used to connect different bot platforms, create testing cases and conduct regression testings. However, the testing capability is constrained by the underlying platforms. For example, users may be still required to design testing dialogs manually. Therefore, Botium cannot perform large scale end-to-end pre-deployment testing.

The overall comparison of different platforms is given in Table 1. Most current platforms only focus on regression testing. 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 platforms are capable of computing turn-level NLU metrics, they require significant manual efforts in curating or annotating test cases. In addition, the NLU metrics do not directly translate to the end-to-end goal completion performance. We will show how BotSIM can help circumvent these limitations via large scale automatic dialog generation and simulation.

[3] https://www.botium.ai/

3 BotSIM

BotSIM system overview is shown in Figure 1.

3.1 Generator

The generator takes bot designs and intent utterances as input and produces the required configuration files and dialog goals for dialog simulation.

Dialog act maps. Most commercial TOD bots follow a "rule-action-message" design scheme and there exist clear mappings from system messages to rules/actions. For example, the utterance "May I get your email?" (message) is used to "Collect" (action) the "Email" (slot) with entity type "Email" from the user. Therefore, this message can be mapped to the "request_Email" dialog act by the generator parser. As the only platform-specific component, the parser acts as an "adaptor" to unify bot definitions from different platforms to a common representation of dialog act maps (example in Figure 3) from bot messages to dialog acts. Such (local) dialog acts are automatically inferred by the parser for each dialog. Furthermore, the parser unifies the entire bot design as a graph, where individual dialogs are vertices and their transitions are edges. Each graph node is initially associated with its "local" dialog act map. The dialog act map of a "mega" dialog containing references to other dialogs will be updated by including all the "local" dialog act maps of the dialog nodes along the paths starting from the mega dialog to the terminating dialogs (*e.g.,* "End_Chat"). The algorithm is detailed in Appendix 1. The graph modeling not only enables BotSIM to naturally support the simulation of mega-agents with multiple intents/dialogs

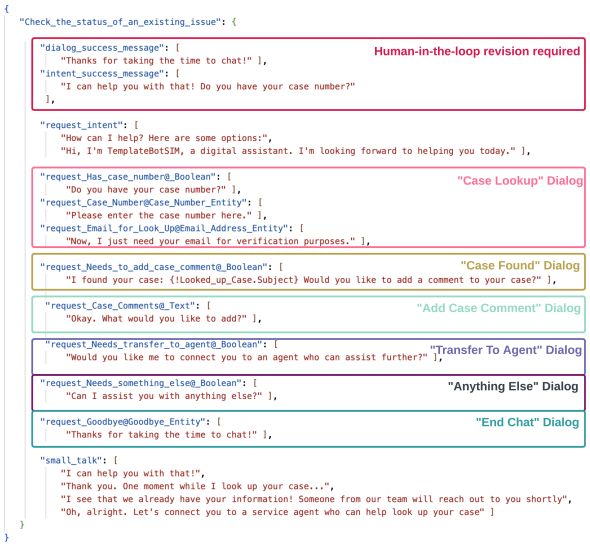

Figure 3: Automatically generated dialog act maps for the mega dialog "Check the status of an existing issue" from the Salesforce Einstein BotBuilder Template Bot.

(Figure 2), it also offers a scalable and controllable approach to exploring conversation paths for greater test coverage and potentially benefiting the conversation design as well.

The generated dialog act maps serve as the BotSIM NLU module to map system messages to dialog acts via fuzzy matching.[4] In particular, the two dialog acts, "dialog_success_message" and "intent_success_message" are the golden labels indicating a successful dialog and a correct intent classification, respectively. They are inferred heuristically by taking the first message as "intent_success_message" and last message as "dialog_success_mesage". BotSIM users are required to review or revise these two dialog acts for each evaluation dialog to confirm their correctness.

Simulation goals. For agenda-based dialog simulation, a user goal comprises a set of dialog acts and entity slot-value pairs needed to complete the task defined by the goal. The dialog acts and slots are from the parsed dialog act maps and the entity values are randomly initialised according to some heuristics. As the entity values are mostly related to products/services, to better test bot NER capabilities, users can replace these random values to real ones when generating simulation goals. Below is a snippet of a simulation goal. The goal is generated by collecting the entity-value pairs in the dialog act map and the ontology. The "inform_slots" contains entities to be "informed" to the bot, whereas the "request_slots" comprises entities to be "requested" from the bot.

```
Check_the_status_of_an_existing_issue_0:
  goal: Check_the_status_of_an_existing_issue
  inform_slots:
    Email_for_Look_Up: andrews@ms-mail.com
    Case_Number: C379870
    Intent: Can I check the latest status of
            my reported issue?
  request_slots:
    Check_the_status_of_an_existing_issue: UNK
  ...
```

All the entity-value pairs in "inform_slots" of the goals are used to test bots' NLU capabilities. The special "intent" slot contains the intent queries generated by the paraphrasing models for pre-deployment testing or user-provided evaluation utterances for performance monitoring.

T5 paraphrasing model. As a core model component, we fine-tune a T5-base (Raffel et al., 2020) model for paraphrasing. To further improve the diversity, model ensemble with an off-the-shelf Huggingface Pegasus (Zhang et al., 2019) model[5] is adopted. The paraphrasing models take intent utterances as input and output their top N paraphrases by beam search. The paraphrases are subsequently filtered by discarding candidates with low semantic similarity scores and edit distances. The filtered paraphrases serve as the "intent" slot values of the goals as intent queries for pre-deployment testing. Our T5-base paraphrasing model has very competitive performance on par with the state-of-the-art HVQ-VAE model as shown in Table 3. It is worth noting that the T5 model yields significantly lower self-BLEU scores, which means the generated paraphrases share less lexical similarities with the source sentences, a merit desirable for BotSIM in generating dialogs to cover greater breadth of language variation. More details of the T5 model are discussed in Appendix B.

3.2 Simulator

We use a dialog-act-level ABUS rather than NUS for the following reasons. First, BotSIM targets commercial use cases and simulation duration and computation are crucial non-functional considerations. NUS inference usually requires GPUs, which can significantly increase the barrier to entry and operational cost. Second, NUS requires large amounts of annotated data to train and are prone to overfitting. Finally, dialogue-act-level simulation is more platform- and task-agnostic. The user

[4] https://github.com/seatgeek/thefuzz

[5] https://huggingface.co/tuner007/pegasus_paraphrase

	WIKI-Answers			QQP		
	Target (↑)	Self(↓)	iBLEU (↑)	Target(↑)	Self(↓)	iBLEU(↑)
HVQ-VAE	39.5	33.0	24.9	30.5	40.2	16.4
T5-base	33.9	23.9	23.9	29.1	35.2	16.3

Table 3: Paraphrasing model comparison. BLEUs are computed from the top one paraphrase with the reference (Target-BLEU)/input (Self-BLEU). iBLEUs are obtained by a weighted sum of target (0.8) and self BLEUs (-0.2).

simulator can be viewed as a dialog agent with its NLU, NLG and dialog state manager.

NLU. BotSIM uses dialog act maps to map bot messages to dialog acts via fuzzy matching.

NLG. For efficient end-to-end dialog simulation, template-based NLG is adopted to convert user dialog acts to natural language responses. Given a dialog act, *e.g.,* "request_Email", a response is randomly chosen from a set of pre-defined templates with a "Email" slot, which is replaced by the value in the goal during dialog simulation. The plug-and play user response templates can be constantly updated to include more language variation as encountered in real use cases.

Dialog state manager. Rule-based dialog manager is used for its simplicity and robustness. The dialog states are maintained as a stack-like structure called agenda. During simulation, user dialog acts are popped from the agenda to respond to different system dialog acts. The two most important rules are for responding to "request" and "inform" dialog acts. While most of the bot behaviours/messages can be converted to these two dialog acts, BotSIM allows users to implement new rules to accommodate novel dialog acts that may only exist in their own bot designs. Figure 4 illustrates an API-based conversation turn between BotSIM and the bot during dialog simulation: Based on the dialog acts matched by the NLU, the state manager applies the corresponding rules to generate the user dialog acts. They are then converted to natural language responses by the NLG and sent back to the bot. The conversation ends when the task has been successfully finished or an error has been captured.

3.3 Remediator

The remediator generates health reports, performs analyses, and provides actionable insights to troubleshoot and improve dialog systems. The reports are presented in a dashboard in Figure 5. More detailed introduction is given in Appendix C.

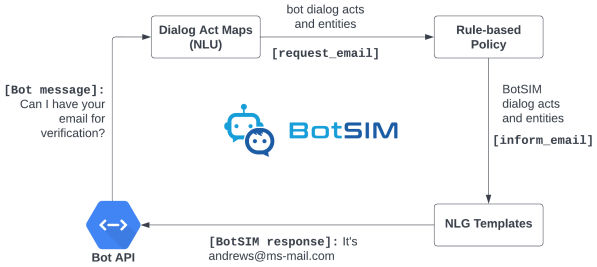

Figure 4: A conversation turn between BotSIM and the bot during a dialog simulation.

Bot health reports. The bot health dashboard consists of a set of multi-level 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) to evaluate the impacts of bot changes from the performance trend in Figure 5(1). Users can also investigate a selected test session as in Figure 5(2). Given a test session, users can select a dialog/intent to check the detailed performance in Figure 5(3). From the detailed intent and NER plots, one can easily identify the most confusing intents and entities.

Actionable remediation suggestions. The outputs of the Remediator comprises actionable suggestions from analysing the simulated dialogs with errors in Figure 5(4). The dashboard allows detailed investigation of all intent or NER errors together with the simulated conversation. The root causes of the failed conversations are identified via backtracking of the simulation agenda. For intent models, the intent queries/paraphrases that lead to intent errors are grouped by the original intent utterances sorted by the number of errors in descending order (drop-down list of Figure 5(4)). Depending on the classified intent labels, the remediator would suggest some follow-up actions (Figure 5(5)). For example, augmenting the intent training set with the queries deemed to be out-of-domain by the current intent model, moving the intent utterance to another intent if most of paraphrases of the former intent utterance are classified to the latter intent.

Conversation analytics. Another useful component of the Remediator is the suite of conver-

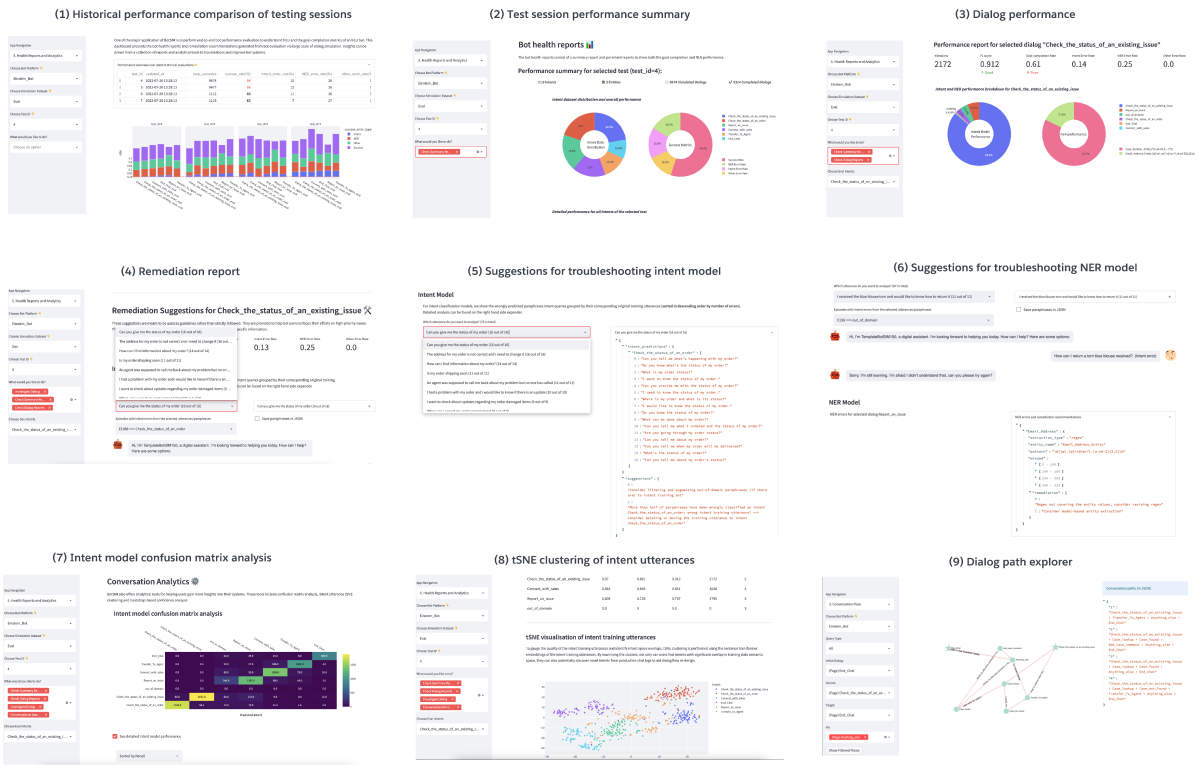

Figure 5: Remediator dashboard including bot health reports, actionable suggestions and conversation analytics.

sation analytical tools to gain more insights for troubleshooting and improving their dialog systems. They include: confusion matrix analysis (Figure 5(7)) for identifying confusion among intents and potential intent clusters (Thoma, 2017), tSNE (van der Maaten and Hinton, 2008) clustering of the sentence embeddings of intent utterances (Figure 5(8)) to help evaluate the training data quality and detect intent overlaps.

Conversation graph modelling Powered by the underlying conversation graph model, the conversation flow visualisation tool (Figure 5(9)) helps users explore their current dialog designs. For example, users can select the "source" and "target" dialogs to investigate 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 improving the bot design.

4 Case Studies

4.1 Salesforce Einstein Bot

The "Template Bot" is the pre-built bot of the Salesforce Einstein BotBuilder platform. It has six intents with hand-crafted training utterances.

Experimental setup. We sample 150 utterances per intent as the training set (train-original) and use the rest 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)". We show how BotSIM can be used to perform data-efficient end-to-end evaluation through dialog user simulation. To probe the baseline system, we apply the paraphrasing models to the "train-original" utterances to get the "train-paraphrases" dataset and use it as the development set. Simulation goals are created by taking the "train-paraphrases" as the intent queries to capture the variations in real user intent queries. The "train-paraphrases" goals are then used to evaluate the dialog system via dialog simulation. After simulation, the Remediator produces the performance reports and remediation suggestions. Although the Remediator provides suggestions for remedying both intent and NER errors, we focus on the intent model since it can be retrained (NER model has not supported retraining yet). Another reason is that the entity values in the goals are randomly generated and may not reflect the real-world values. Since the impact of the NER is removed, the improvement of intent performance directly trans-

Model	Eval.	TA	EC	CS	CI	CO	RI
Baseline	original	0.92±0.03	0.95±0.02	0.89±0.03	0.93±0.03	0.94±0.02	0.82±0.04
	paraphr.	0.88±0.01	0.93±0.01	0.85±0.01	0.91±0.01	0.93±0.01	0.77 ±0.02
Retrained	original	0.92±0.03	0.97±0.02	0.93±0.03	0.95±0.02	0.96±0.02	0.87±0.04
	paraphr.	0.89±0.01	0.94±0.01	0.90±0.01	0.94±0.01	0.94±0.01	0.80±0.02

Table 4: Results for the Einstein Bots case study, before and after retraining the intent model with the augmented training set (F1 with 95% confidence interval computed with 10K bootstrapped samples).

	CB (86)	MP (66)	LC (139)	IC (224)	CC (142)	Acc
Baseline	0.84±0.06	0.83±0.07	0.88±0.04	0.95±0.02	0.96±0.02	0.90
Retrained	0.91±0.04	0.89±0.06	0.94±0.03	0.95±0.02	0.95±0.03	0.92

Table 5: F1 (95% confidence interval) comparison of intent models before and after retraining for CX case study

lates to the improvement of dialog success rate. It is also important to note that the suggestions are meant to be used as guidelines rather than strictly followed. They can also be extended by users to include domain expertise. To validate the effectiveness of the remediation suggestions, we augment the recommended misclassified paraphrases to the "train-original" set to form the "train-augmented" set and retrain the intent model. We then compare the performance before and after retraining on the goals created from the "evaluation-original".

Results and analytics. We observe consistent improvements for all intents on the human-written "eval-original" set after model retraining. More challenging intents (lower F1s), *e.g.,* "RI" and "CS", saw larger performance gains compared to the easier ones such as "EC" (higher F1s). This demonstrates the efficacy of BotSIM and is likely due to more paraphrases being selected for retraining the model on the more challenging intents. We applied the paraphrasing models to the "eval-original" set to get the "eval-paraphrases" set to further increase the test coverage. In Table 9, the second row (✗) shows the number of misclassified "eval-original" utterances. Out of the remaining correctly classified "eval-original" utterances in the first row(✓), substantially larger number of them have at least one of their paraphrases in "eval-paraphrases" wrongly classified by the same intent model. This indicates that the diversity introduced by the paraphrasing models potentially expands the test coverage by a large margin.

4.2 Google DialogFlow CX

We use the pre-built financial service mega-agent for the flow-based evaluations. Even for a single flow in Figure 2, it is non-trivial to manually design conversations to cover all paths. Through BotSIM's conversation graph modeling, the flow-based conversations can be simulated by generating goals consisting of the dialog acts of all dialog nodes along a traversal path. On top of these flow-based dialog paths, paraphrases of the intent utterances can be used as the intent queries to probe the NLU performance via dialog simulation. To simulate pre-deployment testing, we choose five flows and split the intent utterances into train and evaluation sets. The intent F1 scores are given in Table 5. The flows are "Check Balance (CB)", "Make Payment (MP)", "Lost Card", "Investigate Charges(IC)", "Compare Cards (CC)". Since the financial bot has only ~30 utterances for training each intent, to obtain a more reliable test set, we use the "eval-paraphrases" set together with the "eval-original". The total number of evaluation intent queries are inside the parentheses of the Table 5 header. Similar to the previous study, retrained intent model outperforms the baseline in terms of both F1 and accuracy (Table 5), especially for the challenging flows such as "CB", "MP".

5 Conclusion

We presented BotSIM, a modular end-to-end bot simulation framework for multi-intent dialog generation and evaluation of commercial TOD systems via agenda-based dialog user simulation. Our case studies show that BotSIM can save substantial manual effort in bot evaluation, troubleshooting and improvement. BotSIM can be easily extended to support new platforms by implementing a set of well-defined parser functions to convert bot messages to dialog acts. We are in the midst of open-sourcing the codes including the Web App. We also plan to support more platforms as future work.

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.

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.

References

Anish Acharya, Suranjit Adhikari, Sanchit Agarwal, Vincent Auvray, Nehal Belgamwar, Arijit Biswas, Shubhra Chandra, Tagyoung Chung, Maryam Fazel-Zarandi, Raefer Gabriel, Shuyang Gao, Rahul Goel, Dilek Hakkani-Tür, Jan Jezabek, Abhay Jha, Jiun-Yu Kao, Prakash Krishnan, Peter Ku, Anuj Goyal, Chien-Wei Lin, Qing Liu, Arindam Mandal, Angeliki Metallinou, Vishal Ishwar Naik, Yi Pan, Shachi Paul, Vittorio Perera, Abhishek Sethi, Minmin Shen, Nikko Strom, and Eddie Wang. 2021. Alexa conversations: An extensible data-driven approach for building task-oriented dialogue systems. In *Proceedings of NAACL-HLT 2021*, pages 125–132.

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. *CoRR*, abs/1607.00070.

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

Paul A Crook and Alex Marin. 2017. Sequence to Sequence Modeling for User Simulation in Dialog Systems. In *Proceedings of the 18th Annual Conference of the International Speech Communication Association (INTERSPEECH 2017)*, pages 1706–1710. ISCA - International Speech Communication Association.

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

Tom Hosking, Hao Tang, and Mirella Lapata. 2022. Hierarchical sketch induction for paraphrase generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2489–2501, Dublin, Ireland. Association for Computational Linguistics.

Florian Kreyssig, Iñigo Casanueva, Pawel Budzianowski, and Milica Gasic. 2018. Neural user simulation for corpus-based policy optimisation for spoken dialogue systems. *CoRR*, abs/1805.06966.

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.

Bing Liu, Gökhan Tür, Dilek Hakkani-Tür, Pararth Shah, and Larry P. Heck. 2017. End-to-end optimization of task-oriented dialogue model with deep reinforcement learning. *CoRR*, abs/1711.10712.

Jiexi Liu, Ryuichi Takanobu, Jiaxin Wen, Dazhen Wan, Hongguang Li, Weiran Nie, Cheng Li, Wei Peng, and Minlie Huang. 2021. Robustness testing of language understanding in task-oriented dialog. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer.

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.

Pararth Shah, Dilek Hakkani-Tür, Gökhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry P. Heck. 2018. Building a conversational agent overnight with dialogue self-play. *CoRR*, abs/1801.04871.

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*.

Hong Sun and Ming Zhou. 2012. Joint learning of a dual SMT system for paraphrase generation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 38–42, Jeju Island, Korea. Association for Computational Linguistics.

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.

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

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2019. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization.

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*.

A Regression test cases for Botium

Below is a Botium regression testing case (script):

```
T01_Check_Card_Balance
#me
What is the due amount for my card?
#bot
Can I have the last 4 digit card number?
INTENT balance_enquiry
#me
5789.
#bot
You have $50.8 due for this card.
INTENT balance_enquiry
ENTITY_VALUES 5789
```

It is important to note that the Botium "end-to-end" testing is different from BotSIM as it aims to ensure the bot interface and operation work across different devices without regression errors. In other words, they are still for detecting regressions rather performance evaluation. For BotSIM's end-to-end evaluation, we refer to the setup that BotSIM takes in bot designs and automatically 1) parses bot designs, 2) generates test cases, 3) performs large-scale end-to-end dialog simulations, 4) analyzes outputs and provides actionable remediation suggestions for troubleshooting and improvement.

B T5-base paraphrasing model

Pytorch (Paszke et al., 2019) is used to fine-tune the T5-base model with the adam optimizer. The effective batch size is 128. The learning rate is 1e-4 and no warm-up is applied. The maximum sequence length is 128. Four NVIDIA A100 GPUs are used. After each epoch, we compute the iBLEU scores on the development set according to (Hosking et al., 2022; Sun and Zhou, 2012) to decide whether to keep the current checkpoint or not. The best model was obtained after 88 epochs.

B.1 Paraphrase datasets

We use the datasets released in the HVQ-VAE paper (Hosking et al., 2022) for finetuning the paraphrasing model. The datasets are "Wiki-Answers", "QQP" and "MSCOCO". Instead of training a separate model for each dataset as in (Hosking et al., 2022), we fintune a single paraphrasing model from the pooled dataset of the three. Additionally, we also curate our own set of paraphrasing datasets and use them together with the other three datasets. The dataset information is given in Table 6. Note some of the datasets are not initially designed for paraphrasing tasks. Therefore, they may also contain noisy or trivial labels for paraphrasing. Therefore, a filtering process is applied to select the final training sentence pairs. The filtering process applies thresholds on semantic and lexical scores to strike a good balance between lexical variation and semantic similarity between the paraphrasing sentence pairs. In particular, we use sentence transformer (Reimers and Gurevych, 2019) score to measure the semantic similarity and FuzzyWuzzy ratio (based on Levenshtein distance [6]) for lexical diversity. Sentence pairs with low semantic

[6] https://pypi.org/project/python-Levenshtein/

Algorithm 1 Dialog act maps inference from bot designs (MetaData/API)

```
local_dialog_act_maps = {}
for dialog ∈ all_dialogs do
    local_dialog_act_maps [dialog] = {}
    for message ∈ all_messages do
        dialog_act = infer_dialog_act_from_message(message)
        if dialog_act not in local_dialog_act_maps [dialog] then
            local_dialog_act_maps [dialog][dialog_act] = []
        end if
        local_dialog_act_maps [dialog][dialog_act].append(message)
    end for
end for
global_dialog_act_maps = {}
for dialog ∈ all_non_end_dialogs do
    global_dialog_act_maps[dialog] = local_dialog_act_maps[dialog]
    for end_dialog ∈ all_end_dialogs do
        for node ∈ conv_graph.simple_paths(dialog, end_dialog) do
            global_dialog_act_maps[dialog].update(local_dialog_act_maps[node])
        end for
    end for
end for
```

similarities (noisy labels) or low Levenshtein distances (trivial labels with little lexical variation) are discarded. We also performed a benchmark of iBLEU scores in Table 3. The paraphrases are generated via beam search with the same beam size of 10. The results of HVQ-VAE are taken from the original paper. The discrepancy of the QQP results from the paper is due to some train/eval data overlap we found in their original setup. We contacted the authors and they provided the updated QQP results and the fixed datasets. We thus used the bug-fixed version for finetuning and evaluating our T5-base model. The off-the-shelf Pegasus model has the largest model size but performed the worst compared to the other two models across all scores. Since the author did not reveal anything about the model, we cannot finetune it with the same datasets as the T5-base. Using the same setup as HVQ-VAE, we finetuned a T5-base-Single-Task model for each task and it consistently outperformed the HVQ-VAE model on the task that it was trained on. On the contrary, The single-task models performed significantly worse on the task they were not trained on (see the numbers with *), indicating the negative impacts of domain mismatch. In addition, it is impractical to finetune a new model for each new dataset or task. Therefore, we pooled all datasets together and finetuned a single model "T5-base-Multi-Task". Although the multi-task model performs slightly worse than the single-task ones on each individual task, the overall performance is still on par with the state-of-the-art HVQ-VAE model, especially on the QQP task. Therefore, we choose the "T5-base-Multi-Task" as the BotSIM paraphrasing model.

We apply the same filtering principle for the training data preparation to the generated paraphrases to keep the ones with high semantic and low lexical similarities. In Figure 6, we show the number of candidates before and after filtering when generating the "train-paraphrases" set. For simple intents like "TA" and "EC", almost half of the original candidates are discarded. More challenging intents have more surviving paraphrases due to larger variation in their training utterances. The filtered candidates are then used as the intent queries for creating the simulation goals.

B.2 Investigation into misclassified paraphrases

From Table 9, we can see the paraphrasing models help increase the testing coverage as some correctly classified original intent queries (prediction ✓) have misclassified paraphrase intent queries (prediction ✗). Below we show two successfully classified original utterances with their wrongly classified paraphrases of the "Report an issue (RI)" intent.

	Task	Sent-Transformer score	FuzzRatio	No. Final Pairs
SNLI	NLI	[0.70, 0.99]	-	13,635
MNLI	NLI	[0.80, 0.99]	-	32.398
PAWS-Wiki	Paraphrasing	-	-	19,004
tapaco-en	Paraphrasing	[0.50, 0.99]	70	14,735
WIKI-Answers	Paraphrasing	-	-	79,6679
QQP	Paraphrasing	-	-	16,3621
MSCOCO	Paraphrasing	-	-	47,3210

Table 6: Datasets for T5-base paraphrasing model finetuning. "-" means no filtering applied.

	WIKI-Answers			QQP		
	Target (↑)	Self(↓)	iBLEU (↑)	Target(↑)	Self(↓)	iBLEU(↑)
Pegasus	31.4	55.3	14.0	23.8	46.0	9.9
HVQ-VAE	39.5	33.0	24.9	30.5	40.2	16.4
T5-base-Wiki	42.7	42.7	**25.6**	14.9	20.0	7.9*
T5-base-QQP	32.3	46.8	16.5*	31.9	42.7	**17.0**
T5-base-Multi-Task	33.9	23.9	23.9	29.1	35.2	16.3

Table 7: Performance benchmarking of different paraphrasing models. The BLEU scores are computed from the top one paraphrase candidate with respect to the reference (Target-BLEU) or the input (Self-BLEU) sentence. The iBLEU scores are calculcated using $\texttt{Target-BLEU} * 0.8 - \texttt{Self-BLEU} * 0.2$. Numbers with asterisk* denote the "zero-shot" performance.

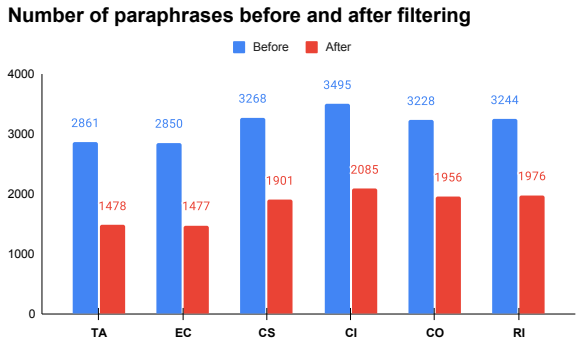

Figure 6: Number of paraphrase candidates before and after semantic and lexical filtering for "train-paraphrases"

```
{
   My order was damaged in
      shipping: [
    Why did my order get damaged
       during shipping?,
    Why was my order damaged when
       it was shipped?,
    Shipping damaged my order.,
    The order was damaged when it
       was shipped.
  ],
  The bottle of conditioner was
     open when it arrived i need
     a replacement: [
    I need a new bottle of
       conditioner because the
       one that arrived was open.
       ,
    I need a replacement for the
       opened bottle of
       conditioner.,
    I need a replacement for the
       open bottle of conditioner
       .,
    I need a new bottle of
       conditioner because the
       one I received was open.
  ]
}
```

As suggested by the Remediator, users can select some of the high-quality paraphrases to augment the original intent training set to refine the intent models. They can also filter from the evaluation (manual-crafted utterances or product chat logs) paraphrases to create a larger evaluation set for performance monitoring. This saves substantial hu-

Dataset	Intent enquiries	TA	EC	CS	CI	CO	RI
Train	train-original	150	150	150	150	150	150
	train-augmented	255	184	212	268	215	294
Dev	train-paraphrases	1465	1467	1754	1989	1895	1786
Eval	eval-original	182	145	183	222	205	178
	eval-paraphrases	1190	933	1648	2172	1936	1795

Table 8: Dataset information for the Einstein Template Bot case study.

eval-original	TA	EC	CS	CI	CO	RI
prediction ✓	9	17	27	33	34	61
prediction ✗	9	7	16	19	8	26

Table 9: Test coverage expansion via paraphrasing

man efforts in creating or annotating dialog testing data.

C Remediator reports and analytical tools

The Remdediator outputs are detailed in the bot health report dashboard shown in Figure 5. The left panel gives users options to navigate through the dashboard. For example, they can select different bot platforms, datasets, test ids and intents. The bot health report (the first row of Figure 5) offers a multi-scale view of simulation performance. At the highest level is the historical comparison of most recent testing sessions (Figure 5(1)). For example, a testing session may be needed after each major bot update. From the historical performance comparison, users can see how certain changes impact the overall bot performance and decide whether to keep or revert the update.

Given the historical performance, users may be interested in further investigating a particular testing session. They can do so by selecting one from the drop-list of all testing sessions and enable the "Check Summary Report" option in the multi-selection box. The resulting overall bot health report for the selected test session is shown in Figure 5(2). It summarizes the simulation information including number of intents, entities and simulation episodes. The two doughnut charts depict the dataset distribution and the overall success metrics.

To investigate the detailed performance report of each individual intent (Figure 5(3)), users can navigate to "Check Dialog Report" and select a dialog from the drop-list. The detailed dialog report presents the intent and NER performance. One can quickly identify the most confusing intents or entities and focus their efforts to investigate and resolve the confusions.

To help troubleshoot the identified errors, users can select "Investigate Dialog" to see the remediation suggestions. Figure 5(5) shows the misclassified intent query paraphrases and their corresponding original utterances, grouped by the wrongly predicted intent labels. Given the prediction results, suggestions are provided for possible further actions. For the given example, all paraphrases of the utterance "Can you give me the status of my order" have been classified as the "check order" intent, indicating an annotation error of the original utterance. Therefore, this utterance should be moved from the "check issue" intent to the "check order" intent.

To gain more insights into their bot systems, users can harness the conversation analytical tools for better comprehension of the simulation results. To understand more about the intent classifier, confusion matrix analysis is applied to the intent predictions of the simulated conversations(in Figure 5(7)). A detailed and sortable intent performance can be displayed by checking the checkbox, allowing users to quickly identify the worst performing intents in terms of recall, precision or F1 rates. This helps them plan and allocate resources to improve the poor-performing intents.

To gauge the quality of the intent training utterances and identify intent overlaps, tSNE clustering is performed based on the sentence transformer embeddings of the intent training utterances. By examining the clusters, not only can users find intents with significant overlaps in the training data semantic space, they can also potentially discover novel intents from production chat logs to aid dialog flow re-design. The dashboard can be easily extended to to support more analytical tasks.

D Streamlit web app

Finally, we give some brief discussions of the Streamlit Web App. The motivation is to offer BotSIM not just as a framework for developers but also as an easy-to-use app to end users such as bot admins without diving into technical details. The app can be deployed as a docker container or to the Heroku platform. We use Streamlit as the front-end and Flask as the backend. A set of API functions are designed to communicate with BotSIM. For multi-platform support, keeping track of the simulation status and historical performance, a SQL-based database is used. BotSIM supports two types of databases including Sqlite3 and Postgres. To support Heroku deployment, particularly its ephemeral file system, cloud storage such as AWS S3 is used to store the simulation logs and results.