TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

Graduation Project

to obtain the

National Engineering Diploma

in Applied Sciences and Technology

Specialty: Software Engineering

Chatbot Robustness Testing using Reinforcement Learning

Prepared by

Altaf Allah ABBASSI

Hosted by **SWAT Lab, Ecole Polytechnique Montréal**

Defended on the 6th of October 2022 in front of a jury composed of:

Dr. Hazar MLIKI	President of the jury
Dr. Ghada Guesmi	Reviewer
Dr. Foutse KHOMH	Supervisor at the Company
Dr. Sonia BOUZIDI	Supervisor at INSAT

School year: 2021 / 2022

TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

Graduation Project

to obtain the

National Engineering Diploma

in Applied Sciences and Technology

Specialty: Software Engineering

Chatbot Robustness Testing using Reinforcement Learning

Prepared by

Altaf Allah ABBASSI

Hosted by **SWAT Lab, Ecole Polytechnique Montréal**

Supervisor at the Company: **Foutse KHOMH**	Supervisor at INSAT: **Sonia BOUZIDI**
Date:	Date:

School year: 2021 / 2022

Acknowledgments

Before stating any description of this project, I have to thank all the people who participated in the realization of this work.

A profound thanks to my supervisor Mr. **Foutse Khomh** for for allowing me to work on such an enriching project, and for his precise and constructive advice throughout the internship.

I wish to express my sincere gratitude to my academic supervisor Mrs. **Sonia Bouzidi** for her insightful and concise advice and guidance throughout the internship, which enabled me to complete this work.

I am very grateful to Mr. **Houssem Ben Braiek** for the time and effort afforded, his continuous support, and his availability.

Many appreciations and gratitudes go to **Ahmed Haj Yahmed** for his helpful advice and encouragement in solving many challenges.

I am very grateful to the members of the honorable jury for taking the time to evaluate my work and for providing me with their valuable advice and comments.

Dedication

To **Ayoub**, my great brother, childhood friend, and life partner, who believed in me first.

To **Latifa**, my beloved supermama, who loved me the most and made many sacrifices for me. Thank you for your trust and encouragement.

To **Hassen**, my distinguished father, who taught me how to love and protect myself and still considers me his little girl.

to **Mohamed**, my legendary grandfather, was the first to take my ideas and opinions into consideration. Thank you for your unending assistance over the last 23 years.

To **Kamel**, my favorite uncle and my studies sponsor, thank you for your unconditional support and advice.

Abstract

Chatbots are computer programs designed to simulate conversations with human users. These systems are increasingly being adopted by multiple industries in order to automate some aspects of customer services. Bots are considered as reliable businesses partners with low-cost and 24/7 support. Meanwhile, poorly designed conversational systems cause customer dissatisfaction resulting in high turn rate.
With the rising demand and the use of chatbots in more critical tasks, ensuring the reliability of these systems have become crucial and of paramount importance. Traditional testing methods, using small and clean data samples, are not effective because they don't reflect the diversity and the large size of input space. As a result, proposing advanced testing strategies is an urgent need. To that end we propose a Reinforcement Learning (RL) Testing approach for Chatbot, RLTest4Chatbot, aiming to evaluate the Dialogue State Tracking (DST) capability. RLTest4chatbot extends the testing input space using an association of parametric semantically-preserving Natural Language Processing (NLP) transformations and employs RL to enhance vulnerability coverage.
We evaluate the performance of RLTest4Chatbot on state-of-the art DST models using the Multiwoz benchmark [1–5]. Our results show that, our framework, RLTest4Chatbot successfully challenges the robustness of chatbots and simulates worst case scenarios outperforming average case testing in terms of valid rate, i.e the rate of generated semantically-preserved inputs, and failure rate, i.e the rate of generated valid inputs that cause the chatbot to fail.

Keywords: Software Testing, Chatbots, Dialogue State Tracking, Reinforcement Learning, NLP Transformations

Contents

List of Figures

List of Tables

List of acronyms

- **INSAT** National Institute of Applied Sciences and Technology
- **SWAT Lab** SoftWare Analytics and Technologies Laboratory
- **NLP** Natural Language Processing
- **NLU** Natural Language Understanding
- **NLG** Natural Language Generation
- **DST** Dialogue State Tracking
- **E2E** End-to-End
- **RL** Reinforcement Learning
- **DRL** Deep Reinforcement Learning
- **PMDP** Parametric Markov Decision Process
- **DQN** Deep Q-networks
- **P-DQN** Parametric Deep Q-networks
- **H-PPO** Hybrid Proximal Policy Optimization
- **MPO** Maximum Posteriori Policy Optimisation
- **DDPG** Deep Deterministic Policy Gradient
- **DL** Deep Learning
- **AI** Artificial Intelligence
- **Q/A** Question/Answering
- **SQL** Structured Query Language
- **SQLI** SQL Injection
- **XSS** Cross Site Scripting

- **GCP** Google Cloud Platform
- **CC** Compute Canada
- **HTTP** Hypertext Transfer Protocol
- **Woz** Wizard-of-Oz
- **MultiWoz** Multi-domain Wizard-of-Oz
- **RQ** Research Question
- **DSTC** Dialogue State Tracking Challenge Wizard-of-Oz
- **TRADE** DTRAnsferable Dialogue StatE Generator
- **Simple-TOD** Simple Language Model for Task-Oriented Dialogue

Introduction

Chatbots are an Artificial Intelligence (AI) programs that communicate with users. They have been around for a while. The first bot was developed in the 1960s. It was a psychotherapist simulator called ELIZA [8]. In recent years, with the development of AI and Machine Learning (ML), as well as the rising popularity of messaging applications, the use of chatbots is increasing across a wide range of industries. More and more businesses are opting to automate some aspects of the customer experience. Since 2019, chatbots have seen 92% growth as a brand communication channel, and it is estimated, that by the end of 2022 bots will handle 75-90% of healthcare and banking systems [9].

Although chatbots appear to be incredibly popular and efficient, not everything about them is perfect. Poorly designed conversational systems are miserably failing to engage with users or carry out basic tasks leaving a bad impression that outlasts [10] resulting in 73% turn rate [11] and additional expenses. Furthermore, if a patient asks an AI bot for medical treatment but the chatbot doesn't fully recognise his health condition, it could result in serious injury or even cause the patient to die [12].

Since chatbots are widely being used for crucial tasks, where failure would be extremely expensive and possibly endanger people's lives, it is an urgent need to design and implement chatbot testing frameworks. Assessing bots as softwares only is not sufficient; their AI components have to be investigated. To do this traditionally, just like other AI system, they are evaluated using global metrics like accuracy, i.e the rate of correctly classified instance, on clean and small test set. This results in a magnificent performance. Sadly, the employed test sets do not accurately reflect the diversity and richness of the test input space, making it unable to describe the real performance. Additionally, since chatbots converse with humans informally, the input space is even larger due to typos and abbreviations. Increasing the data can be helpful, but doing so in the early stages of development is challenging and expensive.

To that end, the scientific community and businesses engaged in AI are constantly making investments in creating and putting into use new techniques and technologies to enable more precise chatbot evaluation.
Recent testing techniques have additional restrictions that chatbots must take into account. They did not consider the context of the conversation and would test each question as a separate input. Additionally, they only support a limited number of sentence variations and use arbitrary dialogue simulations to assess only average-case robustness. Finally, despite the fact that chatbots are composed of 3 main modules: Natural Language Understanding (NLU), Dialogue State Tracking (DST), and Natural Language Generation (NLG), the previous testing techniques mainly take into account only NLU. To fill this gap we propose a Reinforcement Learning Testing framework for Chatbot (RLTest4Chatbot), that assesses the robustness of DST module under worst case scenarios. In order to do this, RL is used to enhance vulnerability coverage by selecting a Natural Language (NLP) transformation which systematically generates perturbed inputs.
Our main contributions can be summarized as follows: (i) enriching the action space through a combination of parametric NLP transformations, (ii) using RL as a search strategy, and (iii) proposing the first framework for chatbot's DST module testing.

The remaining of this work is organized as follows:

- Chapter 1: Project Context
- Chapter 2: Background and Related Word
- Chapter 3: RLTest4Chatbots: A Robustness Chatbot Testing Framework
- Chapter 4: Empirical Evaluation
- Conclusion and Future Work

1

Project Context

Introduction

This chapter defines the scope of this internship. It introduces the host laboratory, the motivations and problem statement of the internship project, the general timeline, and, finally, the project management method.

1 Presentation of the hosting Laboratory

The internship took place in the department of Computer and Software Engineering at Polytechnique Montréal [13], Canada within the SoftWare Analytics and Technologies (SWAT) [14] Laboratory, from March 1 to August 26, 2022, under the supervision of Pr. Foutse Khomh, a full Professor at Polytechnique Montréal and the leader of the SWAT team, and Dr. Houssem Ben Braiek, a postdoctoral researcher at Polytechnique Montréal and a member of the SWAT team.
Polytechnique Montréal is a leading engineering educational and research institution in

Canada. It is an important player in the engineering and innovation sectors in Quebec, as well as a preferred partner for several innovative firms in Canada and around the world. Polytechnique newline SWAT laboratory is a business-oriented research lab that is actively exploring and developing approaches and techniques to assist in the development, maintenance, and deployment of AI-intensive and cloud-based software systems. SWAT researchers use analytical techniques to provide insight and actionable data about development teams' activities. They also implement tools for evaluating and improving the quality of software systems. Early models and tools developed by SWAT members are already in use in the business, and the lab is currently working on projects with Canadian and foreign companies, as well as several other research projects.

2 Motivations and Problem Statement

In this section, we discuss the motivations that led us to conduct this research and present the problem statement, demonstrating why chatbot testing is still a topic worth more investigation.

2.1 Motivations

Chatbots have recently become ubiquitous in our increasingly digitized society. It is nearly impossible not to have heard of or encountered bots. They can be found everywhere and perform a variety of human tasks ranging from flight booking to medical assistance. Since bots are increasingly being used in more critical tasks, their testing is more challenging. Traditionally, chatbots are evaluated on clean and small test sets using global metrics like accuracy, which measures the rate of correct bot responses from all responses, resulting in impressive performance. Accompanying this outstanding results, many chatbots are failing miserably to connect with their users or to perform simple actions [10]. These approaches are limited to the best and average case scenarios, which leads to overconfidence. However, because bots interact with humans, more worst-case scenarios are required to assess the real performance, such as naturally causing input perturbation by typos.

To that end, the scientific community and businesses are constantly investing in the development and implementation of new approaches and frameworks for chatbot testing. Chatbots are composed of three modules, detailed in 1.1, including the Dialogue State

Tracking (DST). Although DST is important in generating correct bot responses and is responsible of multiple erroneous behaviors, it is neglected when it comes to robustness assessment.

2.2 Problem Statement

Chatbots are in direct natural language communication with humans, who typically use rich language expressions and make typos. This natural behavior may reveal many chatbot robustness weaknesses [15]. Although the DST module is important in generating relevant responses, to the best of our knowledge, they have not been tested yet for robustness. The chatbot testing community was particularly interested in Language Understanding (LU) testing using Natural Language Processing (NLP) adversarial attacks [16] which are simple text transformations with random parameters setup.
For all these reasons, our contributions will be aimed towards designing and implementing a **R**einforcement **L**earning **Test**ing framework for **Chatbot**s (RLTest4Chatbot) to assess chatbot DST state-of-the-art models performance. Our framework aims to evaluate chatbots under real scenarios by simulating human interaction with chatbot using systematically generated user queries.
To do RLTest4Chatbot (i) extends the NLP actions search space using parametric, compound and semantically-preserving textual transformations, then (ii) enhances the exploration of the search space using Reinforcement Learning (RL). Finally, (iii) the testing success rate is measured and compared to the results of random sampler and baseline strategies.

3 Internship Timeline

This section is dedicated to present the internship timeline. We begin by introducing the management methodology, Kanban, and then we present the task management tool, MeisterTask. The project timeline is shown in 1.1.

Table 1.1: Internship Timeline

Task	Start date	End date	Duration
Understanding and exploring chatbots	01-03	10-03	10
Get familiar with state of the art software testing approaches	11-03	16-03	5
Adapt a testing approach (Dial Test) to DST chatbot models	17-03	04-04	15
Design our framework: RLTest4Chatbot	05-04	18-04	13
Get familiar with RL and the different applications	19-04	30-04	11
Implement and design parametric compound NLP transformations and the action space	01-05	21-05	21
Implement the RL testing environment	22-05	30-05	8
Implement the RL testing environment and reward shaping	01-06	07-06	7
Put all together: agent, actions, reward, environment	08-06	15-06	7
Implement chatbot models' wrappers	16-06	30-06	15
Reward engineering and parameters fine-tuning	01-07	10-07	10
Empirical experiment and evaluate results	11-07	11-08	30
Writing internship report and a conference paper	12-08	23-09	42

3.1 Kanban

In this project, we are using Kanban as our work method. Kanban [17], which is a Lean-inspired work method developed by Japanese industrial engineer Taiichi Ono in 1950. Kanban's goal is to track the status of completed tasks and achieve continuous delivery, lowering the risk of overproduction and waste while also shortening lead times and costs. The Kanban method is a system that employs cards to represent the tasks that must be completed in order to fulfil a customer's request. These cards are frequently

classified as "to do," "in progress," and "completed." This way, everyone knows what to do, when to do it, and how to do it.

3.2 MeisterTask

MeisterTask [18] is a cloud-based collaborative task-management tool that adheres to the Kanban paradigm. MeisterLabs GmbH, a software company headquartered in Vienna with offices in Munich, founded it in 2015. MeisterTask is a SaaS platform that allows customers to organise and manage projects and tasks. The web- and app-based interfaces improve communication and remote collaboration by allowing team members to organise and automate workflow, discuss and assign tasks, set due dates, and track progress on configurable project boards in real-time. Other technologies that are integrated with this software include GitHub, Microsoft Teams, Google Drive, Slack, Dropbox, and others.

Conclusion

In this chapter, we framed the project context by introducing the hosting laboratory SWAT. Following that, the problem and its motivation were introduced, as well as a general overview of our proposed approach RLTest4Charbot. Finally, we present the project timeline as well as the Kanban work methodology that we used in order to manage this project. In the following chapter, we will introduce the fundamental concepts and backgrounds required to understand the functioning of our testing framework, RLTest4Chatbot and previous works on chatbot testing.

2

Background and Related Work

Introduction

In this chapter, we present the required and relevant concepts necessary to understand the content of our project. It is divided into two sections: first, we provide the fundamental principles that are referenced in subsequent chapters, and then we analyze related work in the literature.

1 Background

This section introduce several key concepts and definitions required for understanding the main points of this study. Because chatbots are a special subset of software, we start by describing their different types and key components. Then, we go over various software testing fundamentals. Finally, because our testing framework is based on reinforcement learning, we provide an overview to RL.

1.1 Chatbots

A chatbot, also known as a chatterbot, conversational system, or simply a bot, is "a computer program designed to simulate conversation with human users" [19] with a specific goal in mind. Chatbots are divided into two types based on their purpose: chit chat and Task Oriented Dialogue (TOD) chatbots [20].

Chit chat, tea-chat or social chatbot, aims to appropriately answer the sentence they are given by imitating human behavior when engaging in a conversation [21]. This type of chatbot converses seamlessly and appropriately with humans acting as chat companions while maintaining a sense of humour.

While TOD chatbots try to assist users in completing specific tasks, such as booking a restaurant table or purchasing movie tickets, in a continuous and uninterrupted conversation with as few steps as possible. They can be implemented using either hand-crafted and carefully designed rules or a corpus, which allows the chatbot to be trained with a relatively large corpus of conversational data [20].

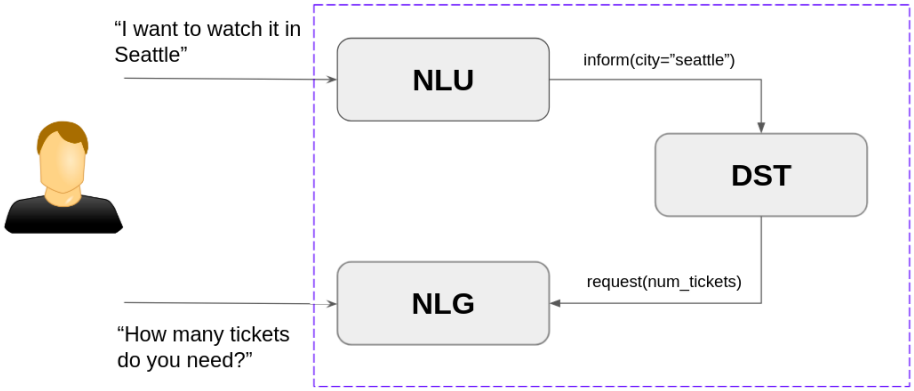


Figure 2.1: Chatbot modules.

As presented in the figure 2.1, trained TOD chatbots are composed mainly of three major components. First, the natural language understanding (NLU) module which recognizes user intents and extracts task related information. Second, the dialogue state tracking (DST) module tracks the state during the conversation and keeps the dialog context in memory. It is responsible of the stateful aspect of the conversation. Finally, given the intent and state, the natural language generation (NLG) module generates an appropriate natural language response [22].

To implement trained TOD chatbots, two approaches are used: either following the

pipelined manner where each module is implemented and evaluated separately or using an unified model following the end-to-end style.

1.2 Software Testing

Software testing is the process of running an application in order to look for software bugs, errors or other defects, related to functional and non-functional requirements. Additionally, it is also used to analyze the software in terms of usability, reliability, efficiency, security ,robustness etc [23]. We mainly categorize testing techniques into two categories: static, also known as manual testing and dynamic also known as automated testing.
Static testing is the process of testing software without running any code. Code reviews and specification documents, for example, are used to detect errors. While under dynamic testing [24], a code is executed. It examines the system's functional/nonfunctional behaviors as well as its overall performance. Dynamic testing runs the software and compares the output to the expected result. It can be done at all levels of testing and in either back, white, or gray box mode [25]. When the systems internal structure and architecture are apparent to the tester, we call it white box, mainly used in functional testing. While black box is when the tester cannot access the inside system internals. It is used for behavioral testing. Finally the gray box allows the tester to see only limited internal system structure. It is a combination of behavioral and functional testing.
One of the dynamic software testing most difficult tasks is the oracle problem [26]. An oracle is a mechanism against which testers can decide whether the outcome of the program on test cases is correct. In most situations, the oracle is either unavailable or too expensive to get. In machine learning, usually there is no oracle without human participation through data labeling. Several solutions to address the oracle problem have been proposed, including metamorphic testing.
A metamorphic testing (MT) approach has been proposed [6] with a view to making use of the valuable information in successful test cases. It does not depend on the availability of an oracle. It proposes to generate followup test cases based on metamorphic relations (MRs), or properties among inputs and outputs of the target function. The violation of MRs indicates potential errors.
This approach, as shown in Figure 2.2, constructs MR that connect inputs in such a way that the relationship between their corresponding outputs is known in advance, allowing the desired outputs to be expected.

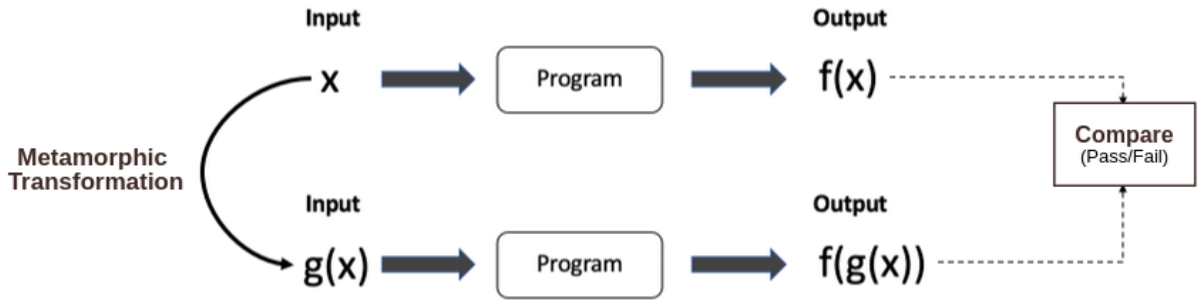
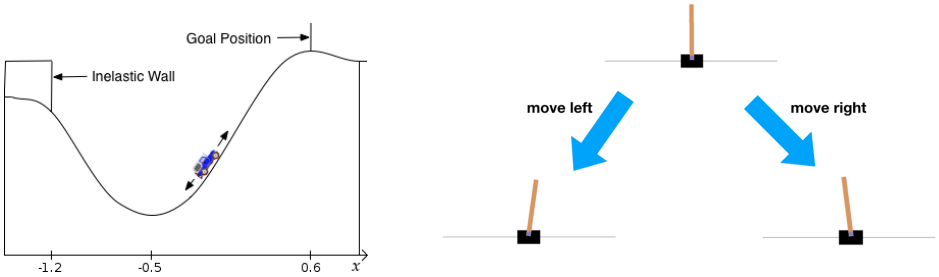


Figure 2.2: Overview of the metamorphic testing process [6]

For example, a metamorphic relation to test the implementation of the function sin(x) can be the transformation of input x to π-x, which allows reviewing the results by checking if sin(x) is equal to sin(π-x).

1.3 Reinforcement Learning

Reinforcement Learning (RL) is a feedback-based machine learning technique in which an agent learns how to behave in a given environment by performing actions and observing the results of those actions.
Deep Reinforcement Learning (DRL) is the combination of RL and Deep Learning (DL) that has demonstrated its ability to resolve high complexity tasks such as autonomous driving cars [27]. In a specific RL environment , the set of all possible actions is known action space and it can fall into one of the three categories: discrete, continuous and hybrid [28]. For discrete actions, we can clearly count the number of actions e.g. in the Mountain Car environment of Atari 2.3a, there are three actions: push the car to the left, push the car to the right, and stand still. Whereas in the continuous action space problem we can not count the number of actions, instead we can describe their range. For example, in the pendulum environment 2.3b, the possible action is to apply a force between 2.0 and -2.0 to keep the pendulum frictionless. Hybrid or parameterized actions combine discrete action k from a discrete set [k] with the associated continuous parameters x_k. Hybrid environments, are designed using Parametric Markov Decision Process (PMDP) [29], as follows M=(S,A,P,R,γ) where S denotes the set of all possible states, A represents the parameterized action space, P(s'|s,k, x_k) represents the Markov state transition probability function, R(s', k, x_k, s) is the reward function, and $\gamma \in [0, 1]$ represents the reward discount factor. The action policy $\pi \quad S \mapsto (k,x_k)$ maps states to parameterized actions.



(a) (b)

Figure 2.3: Game environments design: (a) The mountain car and (b) Pendulum.

There are two main techniques to learn these policies. The first option is hybrid-action reshaping [30], which may be used to reshape actions to either a completely discrete or a continuous action space. If we adopt this method we will face either action space explosion or losing the ability to slightly finetune continuous action parameters. The second approach is to use parameterized-action specific algorithms that do not use approximation or relaxation, for example: H-PPO (Hybrid Proximal Policy Optimization) [31], Hybrid MPO (Maximum Posteriori Policy Optimisation) [32] P-DQN (Parametric Deep Q-networks) [7]. In the context of our project we used P-DQN model which we introduce just bellow.

P-DQN is a hybrid learning framework. The model architecture, as illustrated in figure 2.4, extends the spirit of DQN (Deep Q-networks) [33], which deals with discrete action space, by use DDPG (Deep Deterministic Policy Gradient) [34] for the continuous part of actions.

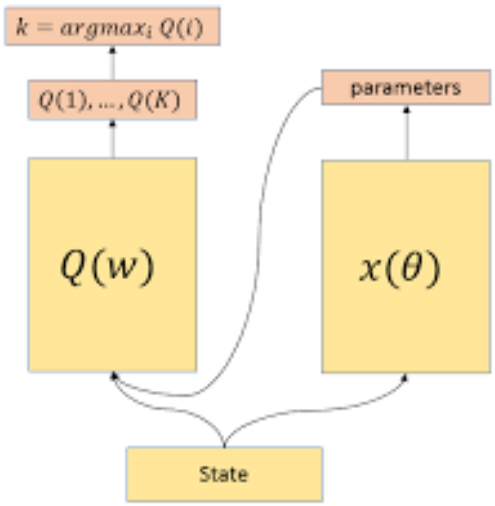


Figure 2.4: Overview of P-DQN model architecture [7]

P-DQN is trained in the following way. First action parameters are approximated with a deterministic policy network. Then, like DQN , deep neural networks are used to approximate $Q(\varpi$,k,$x_k)$ where ϖ are the DQN model weights. The hybrid action is,then, determined by aggregating the discrete action k with the highest $Q(s, k$,x_k, ϖ) and the corresponding w_k continuous parameters. Finally, the two networks, DQN and DDPG, are updated using stochastic gradients.

2 Related Work

Over the last few years, researchers have adapted concepts and methods from software testing to propose advanced chatbot testing frameworks for both functional and non-functional requirements. The DL Testing community has been investigating the dark sides of conversational systems by looking into functional errors, security vulnerabilities, output manipulation, and perturbation robustness.

J. Bozic et al. [35] proposed a functional testing framework based on AI planning to generate communication sequences systematically. To proceed with the plan, each action is assumed to be the user query, and the chatbot response should make the action post-condition true. Re-planning is required when the actual chatbot behavior differs from the expected one.

Since chatbots are connected to databases and perform SQL queries, then vulnerable to related attacks. Bozic, Josip et al. [36], propose a black-box security testing approach against XSS/SQLI attacks. Malicious inputs from test sets are passed, as an HTTP request, to the chatbot under test, then outputs are checked against test verdict.

When it comes to manipulating chatbot outputs and triggering a specific target, different testing methods are proposed [37] [38]. Liu, Haochen et al. [38] crafted a reverse dialogue generator framework based on RL where given a target response as an input, the framework returns the respective question.

Finally, chatbot robustness acquired researchers attention and multiple works were proposed. Pick-n-Plug [39] is a black box metamorphic testing framework, proposed to assess Question/Answering (Q/A) chatbots real performance. The testing process consists of two steps: first picking a block of sentences from another context and then plugging them in clean inputs. Model outputs have to be kept unchanged. Pick-n-plug tests all chatbot models in the same way, ignoring differences, and no search method is used to optimize the selection of plug box neither position. While, LAUG [16] aims to

test different NLU robustness aspects: language variety, noise perturbation and speech characteristics using different NLP approaches: text paraphrasing, word-perturbation, speech recognition, i.e. text to speech then back to text, and speech disfluency. LAUG framework does not use any guidelines to generate pertinent test cases. Then, Liu et al. proposed DialTest [40], a gray-box systematic robustness testing framework. The generation process is guided by the gini impurity metric that measures the likelihood of detecting faults. Dial Test uses the same test sample relevance criteria for different NLU models ignoring the difference between tested chatbots.
Both Dial Test and LAUG, use non-parametric NLP transformations, i.e. dropping the first word in the sentence is considered the same transformation as dropping the last word, and transformations can not be enchained. The table below 2.1 gives a summary about previous robustness testing framework and their corresponding limitations.

Framework	Description	Limitations
Pick-n-Plug	Black-box metamorphic testing framework to assess context memorization robustness.	- Ignoring differences between models - Non-optimized plugged block and position
LAUG	Black-box metamorphic testing tool to test NLU robustness different aspects.	- Ignoring differences between models - Non-parametric NLP transformation
DialTest	Gray-box metamorphic testing framework to test NLU robustness using gini impurity as a search guidance metric.	- Ignoring differences between models - Non-parametric NLP transformation - Fixed gini impurity difference threshold

Table 2.1: Summary of chatbot robustness testing: description and limitations

Conclusion

This chapter provides important theoretical and technical details that serve as the foundation for the work carried out during this internship and that help to clarify the project and its contributions. In the following chapter, we introduce and go into detail about our framework: RLTest4Chatbot.

3

RLTest4Chatbot: A Robustness Chatbot Testing Framework

Introduction

In this chapter we detail our framework for DST assessment. We start with an overview of RLTest4Chatbot and then we discuss each of its components in detail.

1 RLTest4Chatbot: Description

This section introduce our framework, RLTest4Chabotn that aims to assess chatbot DST performance. We start with introducing the proposed framework testing workflow. An overview of the approach is then presented.

1.1 RLTest4Chatbot: Workflow

Figure 3.1 demonstrates the main behavior of RLTest4Chatbot, which is composed of three key components: a Generator which generates an NLP action, a Transformer which applies the NLP action on the clean input, and an Analyzer which returns global performance metrics at the end of the testing process.
We begin our workflow with a seed input, i.e. a set of dialogues. For each turn in the dialogue, the generator predicts the action that will most likely cause the model to mispredict. Next, given the action and clean user query, the transformer systematically develops the adversial DST model input. In 2.1.1 additional information regarding the transformations are provided, and in 2.2 the internal functioning of the generator is discussed.

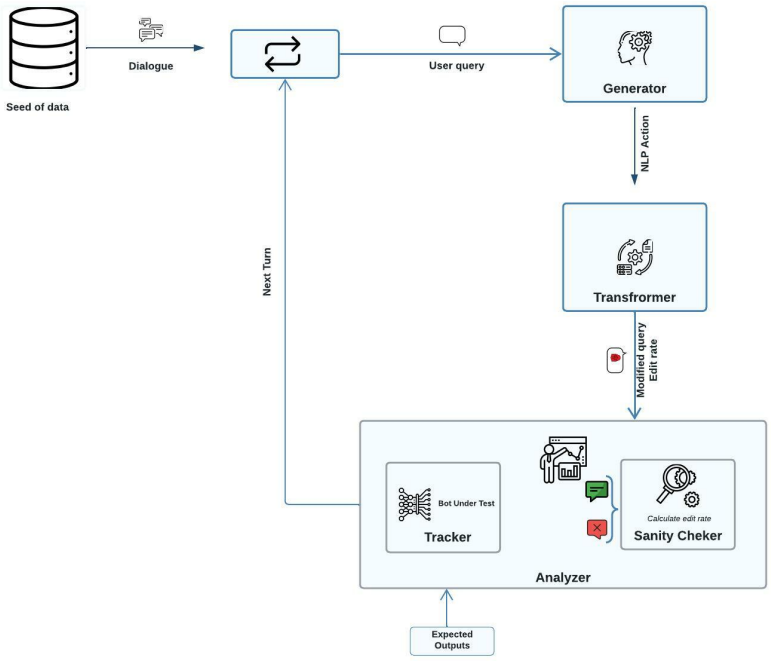


Figure 3.1: RLTest4Chatbot Workflow

The generated model input will pass through the analyzer component. First, a sanity check in order to test the naturalness of generated adversial model inputs. Only realistic perturbed inputs are passed to the chatbot under test. Outputs then are compared following the predefined metamorphic testing criteria. The same process is repeated until finalizing dialogues in the input seed. Finally, the analyzer uses tracked results to provide statistics for overall performance evaluation.

1.2 RLTest4Chatbot: Approach

RLTest4Chatbot's protocol for DST model assessment is designed to systematically search and expose erroneous behavior. Despite the outstanding chatbot performance, when solving best-case scenarios with relevant data, those systems behave differently on naturally occurring noised inputs.
We propose to elaborate NLP metamorphic transformations, in order to systematically generate input test cases. Those transformations will be introduced in 2.1.1. This allows worst-case DST models robustness assessment using perturbed inputs that arise naturally because of common typos or diversification of natural language. The systematic aspect of our approach is established by the variation of NLP transformations, and respective parameters within permitted range, with the purpose of enhancing vulnerability coverage. We adopt an RL-based approach to optimally leverage the data transformation towards the production of naturally misprediction-inducting inputs. Our NLP actions are considered to be multi-parametrized, i.e. an association of hybrid discrete/continuous actions. We have designed the reward to encourage difference-inducting transformations and penalize unrealistic transformations. More details are mentionned in 2.2. We evaluate our framework, according to two key metrics: the ability of generating accurate synthetic data and the capability of identifying DST model weaknesses.
RLTest4Chatbot supports multiple design attacks, cumulative, non-cumulative and hybrid of previous attacks. We think of this design because of contextuality, that does not come naturally as in case with humans. One way to present context is to concatenate all previous queries and answers. Both the context and the adversarial queries are fed into DST models. So, cumulative attack is when all the context is replaced with the attacked one from all previous turns. While non-cumulative is when clean context is used. A third design attack is hybrid, to form the context, randomly propagate or ignore previous attacks.

2 RLTest4Chatbot: Components

After introducing our framework, RLTest4Charbot, we go over and elaborate on each of its components. The transformer is presented first, followed by the generator and finally the analyzer component.

2.1 RLTest4Chatbot Components: Transformer

This part details the transformer component. We begin by introducing our NLP transformations, then our actions representation, and the transformer workflow.

2.1.1 Transformer: Textual adversial transformations

To design our transformer actions we opt to use textual adversial transformations. This technique gained a lot of interest from the NLP community and it is frequently applied in various NLP applications to augment data or to investigate robustness [41]. Textual adversial attacks can be classified based on the granularity of the transformation into three main groups [42] which are character level, word level and sentence level.

- **Character level:** which modifies several letters in words to produce misspellings in order to simulate noise perturbation
- **Word level:** that makes use of unseen words by manipulating the entire word rather than just a few characters to reflect language richness
- **Sentence level:** which actually changes the entire sentence while utilizing linguistic complex operations like paraphrasing and back translation

In our work, we employ only low-level transformations , i.e. character and word level, while discarding the use of sentence level. This design decision was made because there are few possible configurations for high-level, or sentence-level, transformations while there are many for low-level transformations.
Textual transformations can also be categorized based on the employed technique which are rule-based, dictionary-based and model-based:

- **Rule-based:** in which the transformed sentence is obtained by applying a rule, such as dropping a character at the 5th position
- **Dictionary-based:** where a predefined dictionary defines the transformation result, e.g. replace a word by it's commonly misspelled from *friend* to *freind*
- **Model-based:** in which the transformation includes a pre-trained model, e.g. the use of Bert, pre-trained masked language model, to insert a word

We have designed and implemented six textual transformation categories grouping multiple sub-operations. In contrast with prior works that use random configuration attributes (e.g. the inserted position), we propose transformations that are fully parameterized, for example when inserting a character the insertion position and the inserted character are the corresponding transfromation parameters.
In our metamorphic testing based framework, those textual transformations represent the metamorphic operators, where the corresponding relation preserves the DST model output.
Human behavior in informal text conversations serves as the motivation for our designed textual transformations. Our main goal was to mimic the natural noise made by typos while also attempting to use more vocabulary words. Our transformations are: *Char Insert, Char Drop, Char Replace, Word Insert, Word Drop and Word Replace* .

1. ***Char Insert*** A character is inserted at the appropriate position within the given sentence. Operator attributes include the inserted character and position. The char insert englobes: repeating a character to emphasize meaning, mistakenly insert a random character or insert punctuation or known as intruders in community to get around censorship.
2. ***Char Replace*** In the given sentence, a character will is replace position. Char replace encomposases: employing diacritical letter style for a visual transformation, mistakenly substituting due to small mobile integrated keyboards, and letter swapping mimicking speedy writing style known as word scrambling in the NLP literature.
3. ***Char Drop*** This operation causes a character to be removed from a sentence at a particular location, that represents the transformation attribute. Char drop includes: dropping a blank space accidentally (also known as segmentation in literature), or simply missing a character.

When applying character level transformations, we determine the edit rate between two sentences using the Jaro distance [43], which is defined as minimum number of single-character substitutions needed to transform one sentence into another. Jaro distance between sentences s1 and s2 is formulated as:

$$Jaro\ distance\ =\ 1 - \begin{cases} 0 & if\ m\ =\ 0 \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{|m|}\right) & otherwise \end{cases}$$

with

- m is the number of matching characters
- t is the number of transpositions

Because Jaro is a bounded distance between 0 and 1, it is useful for validity checks and takes into account transpositions of matching characters. This definition is appropriate for situations where we might encounter deletion.

4. ***Word Insert*** Within the given sentence, a word is added at the appropriate place. Transformation attributes include the inserted word and position. Word insert englobes inserting an emoji following the textual conversational trends, mistakenly repeating a word, inserting a stop word or enriching the sentence by the insertion of word using Bert which is a masked language pre-trained model.
5. ***Word Replace*** Where appropriate, a word is substituted in the given sentence. The replaced word and the replaced position are the transformation attributes. The word replace transformation englobes mistakenly misspelling a word and employ a commonly misspelled form, substituting by a synonym or a neighbor in the embedding space.
6. ***Word Drop*** This operation causes a word to be removed from a sentence at a particular location, which is a transformation attribute. Word drop includes getting rid of stop words since they add no meaning or mistakenly omitting a word.

When applying word level transformations, we determine the edit rate between two sentences using Jaccard distance [44] which measures the similarity between two sets of words to see which are shared and distinct. Jaccard distance between the two sentences s1 and s2 is formally defined as:

$$Jaccard\ distance\ =\ 1 - \frac{|A \cap B|}{A \cup B}$$

with

- A is set of words of sentence s1
- B is set of words of sentence s2

Jaccard is a bounded distance between 0 and 1, which is convenient when checking transformation natureless. Additionally, when it comes to word-level only transformations, it

is relevant to determine the rate of shared words between two sentences.
The table 3.1 summarizes our designed transformations, corresponding description transformation granularity and transformation technique.

Table 3.1: Textual Transformations summary

NLP Transformation	**Granularity**	**Type**
Char Insert	Character Level	Rule-based
Char Drop	Character Level	Rule-based
Char Replace	Character Level	Rule-based
Word Insert	Word Level	Rule-based Model-based
Word Drop	Word Level	Rule-based
Word Replace	Word Level	Dictionary-based Model-based

2.1.2 Transformer: Action Space

In practice, users can combine multiple textual transformations, such as missing a character in two different words and replacing one word with its frequently misspelled form. We decide to combine various textual transformations, allowing repetition, in order to mimic this natural behavior.
Designing a vectorized action space is essential, it will serve as a search space for the RL-based action generator. Additionally, we choose to normalize action vectors, to speed up generator training, and prevent its divergence. Finally, We got inspired from [45], to set the maximum allowed repetition to 25% of words within the sentence.
To encode the repeated textual transformation vector, we stack the corresponding operation parameters, such as inserted word index, as well as a binary variable that serves as a trigger to signal whether or not the following transformation is active.

Figure 3.2 represents a textual transformation encoding where n is the number of corresponding parameters, and m is the maximum number of allowed repetitions, where each action parameter is a real number in the range of [0,1].
We can associate multiple NLP adversial transformations together, as indicated in figure 3.3, to encode an action in the space we stack together all repeated textual transformation

Figure 3.2: Textual Transformation Encoding

Figure 3.3: NLP Action Encoding

encodings with a real bit [0,1] indicating the importance for each, thus presenting how much each is repeated. Formally an action is represented as:

$$Act(sentence) = T^k(sentence)$$

where $1 \leqslant k \leqslant 6$ is the number of applied textual transformations T, and 6 present the number of our designed textual transformations.

2.1.3 Transformer: Workflow

As represented in figure 3.4, the inputs to the transformer component are the clean user query and the NLP action that will be applied in order to get the transformed question and the corresponding edit rates. As stated in 2.1.2, the actions are encoded and it is necessary to define, as well, the decoding, so the transformer uses them. As motivated in 2.1.2, we can associate multiple textual transformations and apply them sequentially. We choose to define the operations order, so word level are applied first than character level transformations. This decision was made because word level transformations are only applied to grammatically correct words. For example, if we insert a character in a word, we will create a typo preventing it to be potentially replaced with one the corresponding synonym. Figure 3.4 defines the workflow for the transformer component. We apply a loop over all of the transformations after first figuring out how many times each will be repeated according to its importance. After word level operations, the Jaccard distance is determined. We calculate the Jaro distance for all char level transformations in the final

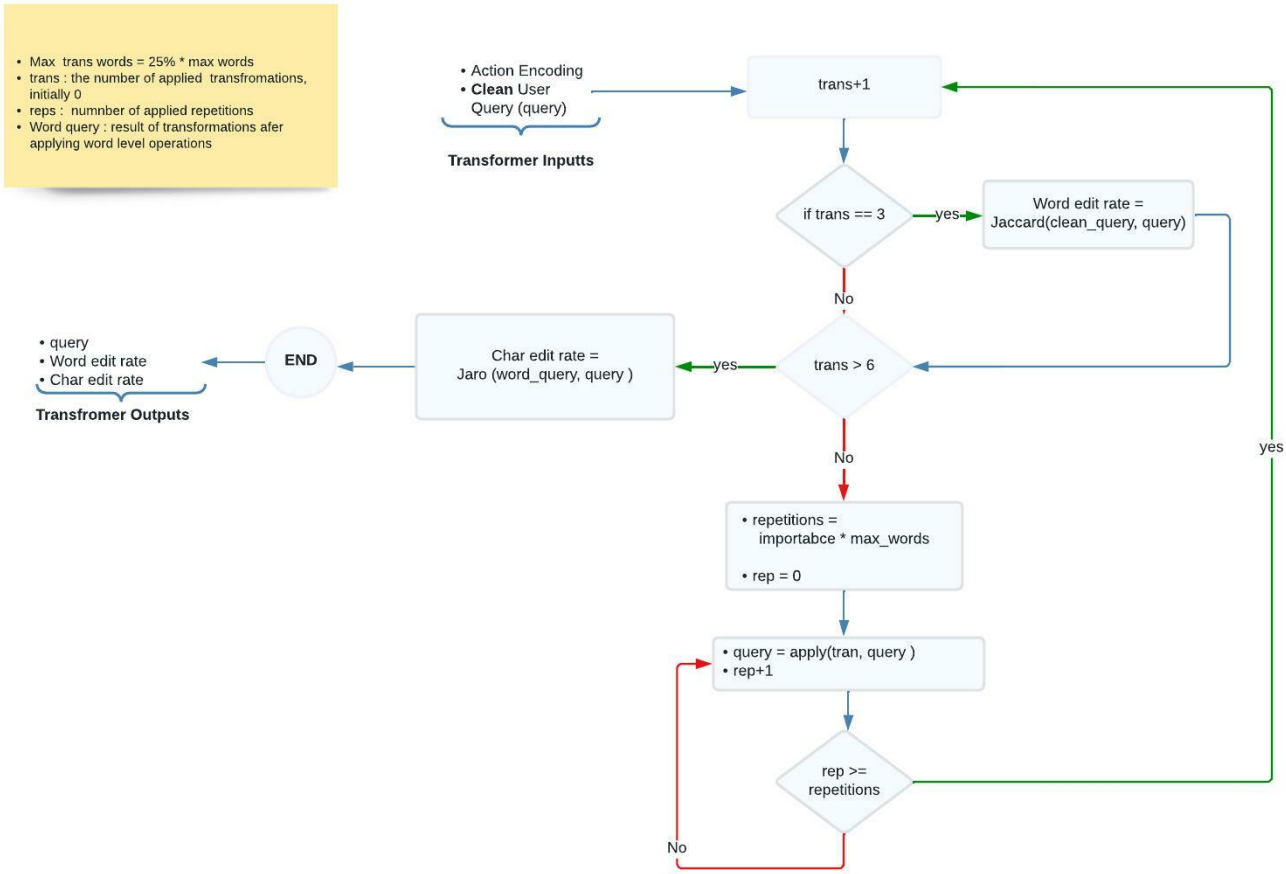


Figure 3.4: Transformer workflow

step. Finnaly the perturbated query and the two distance measures are returned.

2.2 RLTest4Chatbot Components: Generator

In order to evaluate how the DST model behaves when interacting with the chatbot, we want to use a systematic tester. This is exactly what RL is all about, learning by interacting with the environment. The latter motives our choice to adopt the reinforcement learning to generate the NLP actions that probably result in a model misprediction.
In this section, we go into more detail about the RL-based generator. We start by formulating chatbot DST testing as a RL task. Then we present the agent architecture. And finally, we detail the reward function which is the RL recompensation mechanism.

2.2.1 Generator: RL Task Formulation

The conversation between the automated tester and the chatbot consists of a series of turns, with each representing a user question and a system response. At each step of the conversation, the environment, i.e. the chatbot model being tested, is in the state of the current turn. After the NLP action, the state is updated to be in the next turn. Up until the dialogue is finished, this testing process is repeated. As shown in the figure 3.5, the

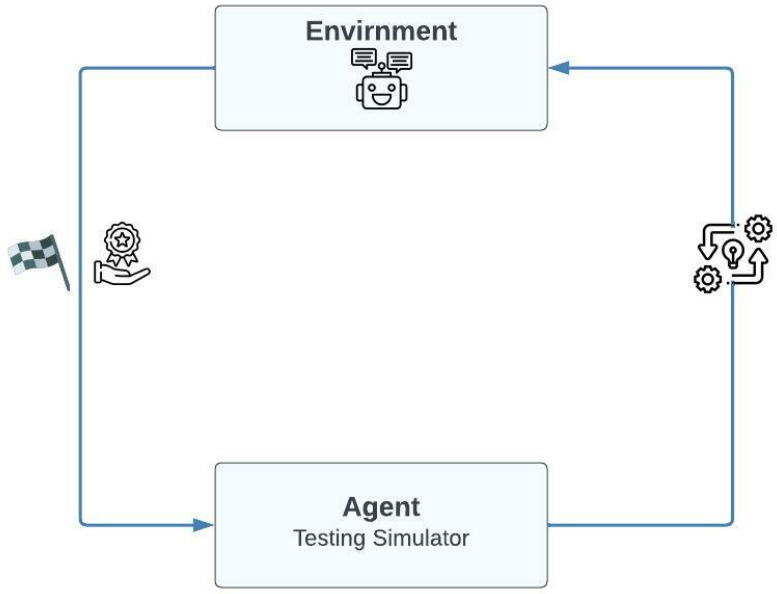


Figure 3.5: Chatbot testing task design

agent, or testing simulator, chooses an NLP action that modifies the user query before feeding it to the environment, or chatbot under test. As an output, we have the reward value that is covered in detail in section 2.2.3; and if the current turn is final.

2.2.2 Generator: Agent

Our actions, as explained in 2.1.2, are a combination of various textual transformations that permit repetitions. Using RL taxonomy, we are dealing with an association of parametrized, or hybrid, actions where the continuous attribute corresponds to the repeated textual operation parameters and the discrete represents the textual transformation.

To the best of our knowledge, no prior work has proposed an algorithm to deal with multiple parameterized action spaces. The RL community is working progressively only

on discrete and continuous. Even for parameterized actions, only few works have been proposed. Thus, we must choose whether to reshape the action space to conform to a conventional design or adjust one of the RL algorithms to suit our needs. When the action space is reshaped, the already huge space will expanded or we will lose the ability to slightly fine-tune parameters. A RL agent can be adjusted as a second solution. Three requirements must be met by the selected agent. First and foremost, it must be proven to achieve state-of-the-art results. Second, the internal functional and architecture of the model must remain unchanged after the agent is adopted. Finally, having the code accessible to facilitate adaptation is more preferable.

P-DQN [7] meets all the stated requirements. Figure 3.6 illustrates the key differences (the red box) between P-DQN and our modified version, Multi-PDQN, which has been suggested to work with association of parametrized actions. Only the last layer is changed from argmax to top_k, where k is the number of associated textual transformations.

Multi-PDQN architecture is composed of two main actors, much like PDQN. The second

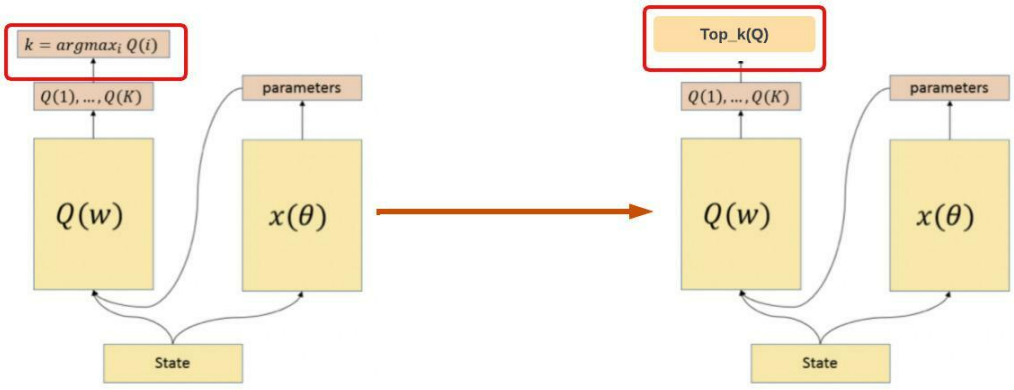


Figure 3.6: Main differences between PDQN and Multi-PDQN

is primarily a Q-actor, while the first is based on e-greedy. The figure 3.7 represents how actions are generated. All action parameters for each discrete action are first generated by the first component given the current conversation state. The Q-actor uses parameters and the environment state to predict Q values that will be used to determine the importance of the top k discrete actions. In the third and final step, the importance and parameter vectors for all possible transformations are combined together to create the NLP action vector.

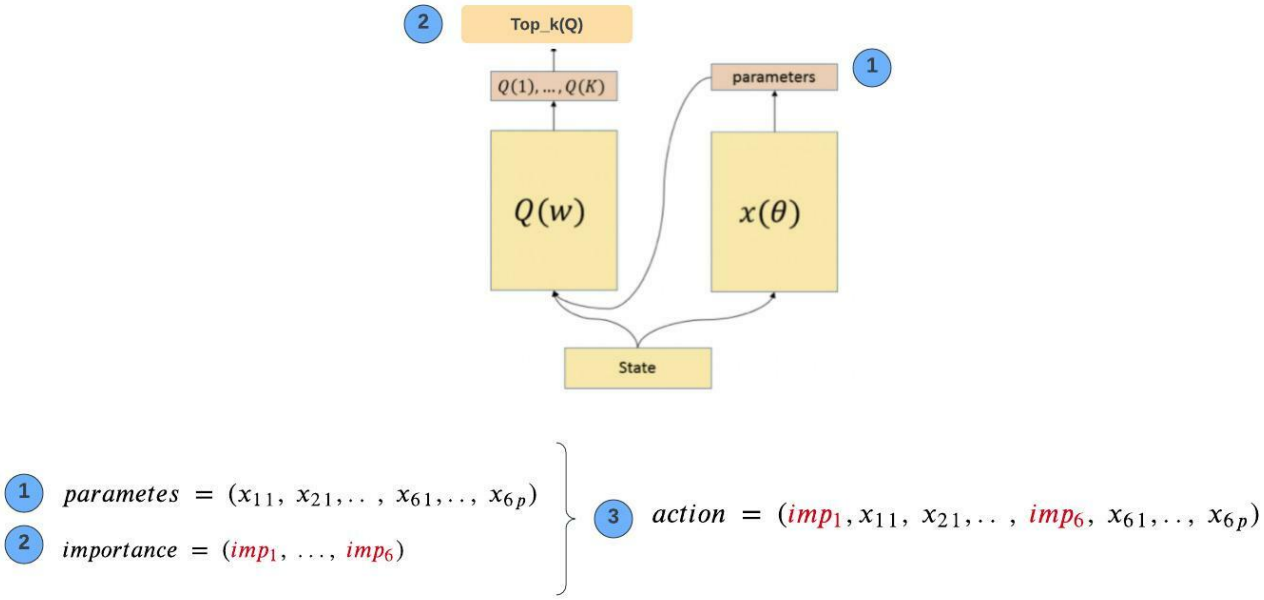


Figure 3.7: Process of NLP action composition from Multi-PDQN

2.2.3 Generator: Reward Function

The reward is an incentive mechanism that instructs the agent, i.e. the generator, what is correct and what is wrong through either reward or punishment. The agent's objective is to maximize the overall rewards.

The agent's goals must be expressed in the reward function. The RLTest4Chatbot generator seeks to encourage both natural behavior and the model uncertainty . Following the Dial Test [40] idea, we first use the difference between the gini impurities of the clean query and the attacked one to measure the generated perturbation. The less certain the model is about the predicted answer the more different are the two queries thus greater gini difference is. Since the gini difference is mostly in the [0.001: 0.01] range, we used exponential to prevent its vanishing. Second, the more natural the transformed questions are, the lower the modification rate is. Therefore, we want to minimize modification rates and penalize unrealistic actions, i.e., when one of the modification rates, either the character or word, is higher than 25%.

The reward function is formally presented as follows to achieve all of the mentioned goals:

$$Reward = \frac{\exp(gini_{dif})}{\sqrt{edit_w^2 + edit_c^2}} + \beta$$

$$\beta = \begin{cases} -100 & if\ edit_w > 25\%\ or\ edit_c > 25\% \\ 0 \end{cases}$$

Where:
$gini_{dif}$: $Gini\ difference$
$edit_w$: $Word\ edit\ rate$
$edit_c$: $Char\ \ edit\ rate$

Optimizing perturbations and ensuring naturalness are obviously mutually exclusive. Where the two goals are evenly balanced must be determined by the RL agent. The training then becomes even more difficult.

2.3 RLTest4Chatbot Components: Analyzer

It is now time to assess the robustness of our DST model's performance. Both batch and single data inputs are supported by our framework.
The generator and transformer receive all user queries from dialogue within the data inputs, as the testing workflow shows in the figure 3.1. The analyzer component follows. The sanity checker first examines the natureless of the transformed query. It must demonstrate that the character and word edit rates are both lower than 25%. We choose to use separate checking constraints because there is no distance that allows us to combine both character and word level transformations. Alzantot, Moustafa Et al. [45] are the first to propose the use of a 25% as edit rate threshold, which the NLP research community later adopted. We manually asserted in our work that this threshold should continue to be adaptable for brief sentences in chatbot contexts.
The transformed query will pass through a tracker sub-component only if it has been marked as valid. There, it will be fed to the tested DST model and outputs will be all tracked. After the systematic tester has been finished with evaluation data inputs, adversial outputs, from the tracker, will be compared with correct ones. We use failure rate, which is defined as the percentage of validly transformed queries that successfully caused the model to fail, to provide insights about the robustness of the model.

Conclusion

We went into detail in this chapter to describe how the RLTest4Chatbot approach assesses the robustness of DST models. The testing workflow was first explained utilizing components. Then,we presented each of the components by outlining our designed NLP actions, our modified RL agent, and how to assess the chatbot model. The experiments performed to assess the effectiveness of the RLTest4Chatbot approach will be presented in the following chapter.

4

Empirical Evaluation

Introduction

This chapter describes the empirical evaluation of our framework. Through three research questions, we will attempt to assess the effectiveness of RLTest4Chatbot. Later in this chapter, we will present our in-production test results. Finally we will discuss limitations and threats to validity.

1 Research Questions

RLTest4Chatbot, is developed in Python based on the Pytorch framework [13] . Further, to implement the NLP adversarial attacks, we used NLTK [46] and gensim [47] libraries. We evaluate the effectiveness of our framework, RLTest4Chatbot, through the following research questions (RQs):

- **RQ1** How effective semantically-preserving transformations are at perturbing DST

models?

- **RQ2** How effective is RLTest4Chatbot at enhancing the coverage of chatbot faults?
- **RQ3** How does RLTest4Chatbot compared to state-of-the-art DialTest?

2 Evaluation

In this section, we will detail conducted experiments. We will start by introducing the experience setup and subjects. The findings of the empirical evaluation will then be presented.

2.1 Experience Setup

We are training a reinforcement learning agent, which interacts with heavy chatbot models, with multiple experience configurations. Such training consumes both resources and time. In order to perform all of our experiences, we used simultaneously a variety of computing ressources from various sources.

- **Google Cloud Platform (GCP)**: which is a collection of cloud computing services. We used 5 E2 virtual machine instances from GCP, each has a 12-core CPU and 8GB of RAM and runs Ubuntu 18.04.
- **Compute Canada (CC)**: which is an advanced research computing platform sponsored by the Canadian Digital Alliance Research. We used 4 computing clusters from CC: Beluga, Cedar, Graham, and Narval.
- **Polytechnique Montréal cluster of machines**

For the internal settings, we adopted the default configuration of P-DQN hyperparameters. Due to time and resource constraints, we choose to keep all k combinations of NLP transformation while limiting our experiments only to the hybrid attack design.

2.2 Experience Subject

In this section, we discuss the subject of our experience. We begin by introducing Multiwoz benchmark. We go over the DST chatbot models later. Finally, we explain how

we formed our dataset.

2.2.1 Multiwoz Benchmark

The Multi-domain Wizard-of-Oz (MultiWOZ) dataset is a large-scale human-human conversational corpus spanning seven domains: restaurant, hotel, attraction, taxi, train, hospital, and police, with a total of 10k multi-turn dialogues averaging 14 turns per dialogue. It has 30 (domain, slot) pairs and over 4,500 possible values, making it at least one order of magnitude larger than previous annotated task-oriented corpora like Wizard-of-Oz 2 (WOZ2) [48] and Dialogue State Tracking Corpus 2 (DSTC2) [49], which have less than 10 slots and a few hundred values. It is also a more challenging dataset due to the mixed domain property. A user can begin the conversation by asking to reserve a restaurant, then ask for a nearby attraction, and finally request to book a taxi.
Multiwoz is a public crowd-sourced labeled dataset with dialogue belief states, dialogue actions, and dialogue domains. It is regarded as a benchmark for various multi-domain conversational tasks such as dialogue state tracking, dialogue acts, and response generation. Multiwoz is constantly being improved and refined. It is available in five different versions, each with a unique improvement. The table 4.1 summarizes the main differences between.

2.2.2 Models

Since there are two primary implementation techniques for chatbots: modular and pipelined. We choose to test two state-of-the-art DST models, Trade-DST and Simple-TOD, following respectively the modular and pipelined appraoches. These two models are introduced in this subsection.

- **Trade-DST** The TRAnsferable Dialogue StatE Generator for Dialogue State Tracking (Trade-DST) [50] uses a copy mechanism to generate dialogue states from utterances. TRADE achieves cutting-edge results in one of the zero shot domains and is able to adapt to few-shot cases without forgetting the domains that have already been trained by using the transferring ability. Since there are many slots among various domains that share all or some of their values, the model shares tracking knowledge across domains to enable its transfer when predicting values that were not encountered during training.
- **Simple-TOD** Understanding user input, choosing actions, and producing a response

Table 4.1: Multiwoz benchmark

Mutiwoz version	**Description and improvements**
Multiwoz 2.0 [1]	• Fully labeled written human to human conversation • 7 domains • 10k dialogues
Multiwoz 2.1 [2]	• Adding dialogue acts • Resolving noise issues with crowdsourcing
Multiwoz 2.2 [3]	• Fixing dialogue state annotation on top of 2.1 • Redefining the ontology
Multiwoz 2.3 [4]	• Unifying annotation for dialogue acts and slot values • Enhancing annotation with coreference
Multiwoz 2.4 [5]	• Refining annotation for only test and dev sets on top of 2.1

are the three tasks that task-oriented dialogue is frequently decomposed into. With a single, causal language model trained on all subtasks and reformulated as a single sequence prediction problem, Simple-TOD [51] proposes a simple and unified approach. This allows Simple-TOD to fully leverage transfer learning from pre-trained, open domain, causal language models such as Generative Pre-trained Transformer 2 (GPT-2) [52]. The proposed Simple Task-Oriented Dialogue (Simple-TOD) approach enables modeling the inherent dependencies between the sub-tasks of task-oriented dialogue, by optimizing for all tasks in an end-to-end manner. In this study, we only evaluated Simple-TOD's DST capability.

To train our testing simulator agents, we use only fully correctly classified dialogues from Mutiwoz test and validation sets. The used dialogues are fully correct which means at each turn the model correctly predicts jointly all the dialogue states. The generated dataset is then splitted into a training and testing set with percentages of 80% and 20%,

respectively. The table 4.2 summarizes the overall statistics of our formed dataset per DST model respectively.

Table 4.2: Our formed dataset

Model	Dataset	Size (# dialogues)
Trade	Multiwoz 2.1	**Train** 315 **Test** 78
	Multiwoz 2.2	**Train** 310 **Test** 77
Simple-TOD	Multiwoz 2.1	**Train** 410 **Test** 102
	Multiwoz 2.2	**Train** 349 **Test** 86

3 Results

In this section we evaluate multiple aspects of our approach. First we begin by (RQ1) investigating the effectiveness of the proposed NLP actions in revealing DST models erroneous behavior. Then, (RQ2) we assess the capability of our RL-based generator in enhancing coverage. Finally (RQ3) we compare our framework results to DialTest, which is the most recent state-of-the-art robustness testing framework, results.

3.1 RQ1: How effective semantically-preserving transformations are at perturbing DST models?

Motivation: RLTest4Chatbot uses textual transformations to systematically generate difference-inducing inputs. In our first RQ, we attempt to check the effectiveness of these transformations.

Method: In order to check the effectiveness of our NLP transformations, we apply them with random parameters to generate adversial inputs that will be passed to the chatbot DST state-of-the-art models. Through this experiment, we keep track of the failure rate, i.e the rate of generated inputs that cause the model to mispredict.

Results: Tables [4.3, 4.4] report the failure rate revealed by the synthetic generated

inputs using random textual transformations respectively for Trade-DST and Simple-TOD.

Table 4.3: Trade-DST Results

Dataset	Faillure Rate
MultiWoz 2.1	7.51 %
MultiWoz 2.2	7.14 %

Table 4.4: Simple-TOD Results

Dataset	Faillure Rate
MultiWoz 2.1	7.79 %
MultiWoz 2.2	5.29 %

By applying textual transformations with random parameters, Trade-DST joint accuracy decreases from 100% to 92.5%, i.e. 7.51% failure rate, and to 92.86%, i.e. 7.14% failure rate, respectively using Multiwoz 2.1 and Multiwoz 2.2. Simple-TOD demonstrates a similar behavior. Its joint accuracy is reduced by 7.8% and by 5.3% using respectively Multiwoz 2.1 and Multiwoz 2.1.
Non-zeros failure rates for all the studied subjects show that the employed transformations, using average-case robustness settings, are able to reveal weaknesses in the chatbot DST models.

Finding 1: Our designed textual transformations are able to uncover model vulnerabilities.

In the light of these results, the analysis of DST chatbot model robustness against optimized textual transformations will be continued towards the worst-case scenarios.

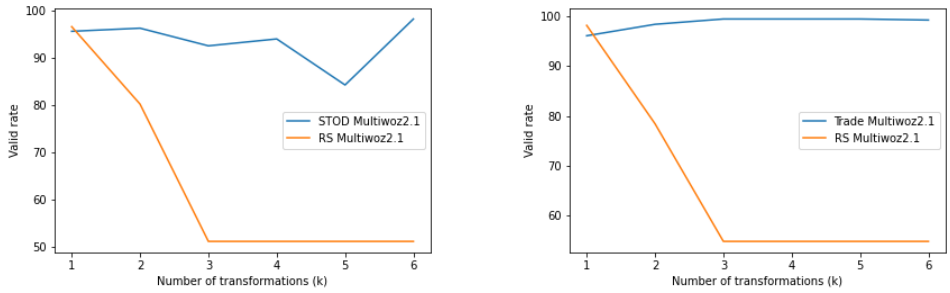
3.2 RQ2 : How effective is RLTest4Chatbot at enhancing the coverage of chatbot faults?

Motivation: RLTest4Chatbot uses a RL-based action generator to enhance worst-case scenarios coverage. In our second RQ, we try to evaluate the contribution of reinforcement learning in optimizing NLP action parameters.

Method: We first test our framework, RLTest4Chatbot, using the experience subjects mentioned in 2.2. Then, we tried a random sampler search strategy instead of the RL-based action generator. For both, we utilized the same transformer component 2.1 and the action space design 2.1.2. Through these experiences, we keep track of various metrics including (i) failure rate, i.e. the rate of valid difference-inducing inputs, (ii) the valid rate , i.e.the percentage of semantically-preserving generated inputs in all generated ones,

(iii) action parameters inertia, i.e. the variance using the statiscal notion which measures the spread of action parameters in the action space.

Results: Figure 4.1 shows the valid rates of generated sentences for both models, Trade-DST and Simple-TOD, on Multiwoz 2.1, using the two search strategies Reinforcement Learning (RL) and Random Sampling (RS). Please refer to appendix A for futher detailled and supplementray results.
The goal of tracking the valid rate is to investigate the effect of combining multiple textual transformations and to assess both strategies' semantic-preserving capability.



(a) (b)

Figure 4.1: Valid Rate using RL vs RS strategies : (a) Simple-TOD (b) Trade-DST

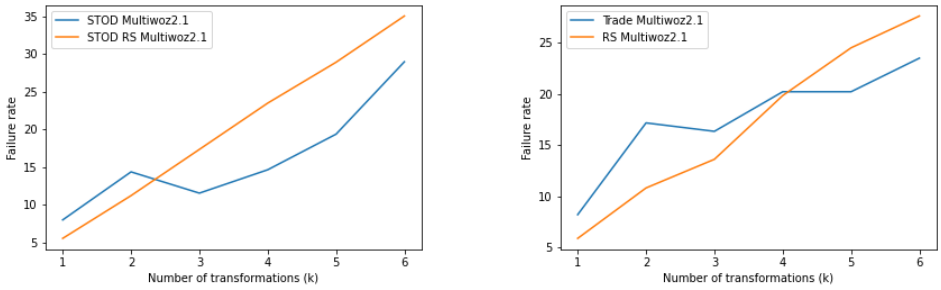
Using RS, the more textual transformations we combine the lower the valid rate we got. Whereas using our trained RL-based generator, the valid rate is always greater than 85% regarding the model, the dataset and the number of associated transformations, while it's much lower using RL and attempts the 50%.

Finding 2: Our RL-based search strategy ensures the naturalness of adversial inputs, by generating semantically-preserving transformations.

The results, shown in figure 4.2, prove that combining our designed textual translations results in a high failure rate for both strategies: RL and RS.

Finding 3: The more textual transformations we associate, the more successful adversial inputs we generate.

When more than 3 textual transformations are combined, the RS strategy generates more successful adversarial sentences, i.e. a higher failure rate, than our RL approach.



(a) (b)

Figure 4.2: Failure Rate using RL vs RS strategies : (a) Simple-TOD (b) Trade-DST

But, RS, as shown in the figure 4.1, performs poorly in terms of the naturalness of generated perturbations. Although encouraging both a high valid rate (favouring the fewest number of transformations) and a high failure rate (favouring perturbations) are mutually exclusive goals, RLTest4Chatbot was successful in learning a compromise between the two objectives.

Finding 4: Our framework, RLTest4Chatbot, maintain a balance between valid and failure rates.

Through this experience, we keep track of action parameters inertia values, which measure the distribution of action points in the action space. The higher inertia is, the more distributed action points are. The tables [4.6, 4.5] summarize all inertia values for possible k, number of associated transformations, respectively for Simple-TOD and Trade-DST.

Table 4.5: Simple-TOD Inertia

k	RL Inertia	RS Inertia
1	1.482	3.36
2	1.601	3.35
3	1.768	3.36
4	1.65	3.36
5	2.18	3.34
6	1.49	3.35

Table 4.6: Trade-DST Inertia

k	RL Inertia	RS Inertia
1	1.88	3.35
2	1.57	3.35
3	1.49	3.36
4	1.66	3.37
5	2.017	3.36
6	2.057	3.36

Results show that our RL approach can generate diverse actions, as demonstrated by non-zero inertia values, and learns pat-

terns confirmed by lower inertia values than corresponding RS values.

Finding 5: RLTest4Chatbot learns actions parameters relevant patterns while ensuring their diversity.

3.3 RQ3 : How does RLTest4Chatbot compared to state-of-the-art DialTest?

Motivation: The goal is to compare RLTest4Chatbot performance in discovering difference-inducting inputs with DialTest, the state-of-the-art framework robustness assessment.

Method: DialTest was first proposed to test chatbot NLU module robustness. We implemented DST support by making a slight modification to it. We decided to use the same evaluation subjects, i.e. DST models and Multiwoz benchmark, and the same edit distance for a fair comparison. We chose to adopt the same hybrid design attack to run both frameworks in order to provide the most accurate comparison. The failure and the valid rates are the comparison metrics we used.
DialTest, is a robustness framework that makes use of the following NLP transformations: Synonym Replace, Word insert and Back Translate. We executed DialTest only on two of the three proposed transformations: Synonym Replace and Word Insert. We discard the use of Back Translation because of three reasons. First authors did not propose an implementation in their official git repository, second we did not define an edit rate for sentence level transformations in the scope of our project and finally back translation APIs are costly.
Two hyperparameters must be defined for DialTest: the maximum number of permitted transformations and a gini difference threshold to accept generated perturbated test inputs. None of these hyperparameters were suggested by the authors, either in their paper or in the source code. We limited the maximum number of transformations to 5, which represents 25% of a query's words. We fine-tune the gini difference threshold, and the best results are obtained using a value of 0.001.

Results: Tables show a comparison between best configuration of DialTest (DT) and of our RL approach using Multiwoz 2.1 for Simple-TOD ans Trade-DST respectively.

The results show that our RL approach outperforms DialTest in terms of generating

Table 4.7: DialTest(DT) **vs** RLTest4Chatbot(RL)

Model	**RL Failure Rate**	**RL Failure Rate**	**DT Valid Rate**	**DT Valid Rate**
Trade-DST	23.5%	33.33%	99.14%	70.08%
Simple-TOD	23.34%	45%	98.21%	71.76%

semantically-preserving inputs, whereas DialTest fails to ensure perturbation naturalness. Because DialTest adversial generated inputs are less natural, the failure rate is higher than that released by our RL approach.

Finding 6: RLTest4Chatbot outperforms DialTest in terms of generating semantically-preserved adversial inputs.

Nevertheless, time and resources consuming are the weaknesses of our framework. To test a model we have to train an RL-based action generator first. Compared to DialTest, which does not require a training phase.

4 Threats to validity

In this section, we discuss potential threats to the validity of our work and emphasize our preventative measures. We categorize these threats into groups for a better clarification.

- **Generalization (selection of subjects)**
  The choice of experimental subjects like dataset and models, can be a threat to how generic our findings are. By using two variants of each evaluation subject, we attempt to reduce this threat. These variants are (i) state-of-the-art like the tested models, (ii) widely-used by the community like the Multiwoz benchmark, (iii) state-of-the-practice like the NLP transformations. Also, we use the official implementation of each model and we rely on open-source libraries.
- **Transformation rate**
  The estimation of edit rates poses a threat to both our project and all NLP applications.

Regardless of the importance of meaning distances between sentences, the research community has not yet developed a reliable method, particularly when it comes to the potential for typos. To overcome this threat, we suggest combining two edit distances, one for typo and another for typo-free sentences that are widely-used by the community: Jaccard and Jaro distances. Finally, we determine whether the two sentences have the same meaning by using state-of-the-art thresholds.

- **Exclusive application on DST**

  All of the earlier studies only tested NLU robustness, so we chose another chatbot module as a study case because of its importance and error propagation characteristics. Nevertheless, the use of the DST as an evaluation to RLTest4Chatbot may constitute a risk to its application on other modules. We try to prevent this threat by designing a generic approach and a plug-and-play architecture regardless of the module.

Conclusion

In this chapter, we went through the empirical evaluation of our proposed framework. In our assessment, we attempt to address three research questions to prove the effectiveness of RLTest4Chatbot. Our results show that our RL approach can successfully generate meaningful test inputs that cause the DST chatbot model to mispredict. It also outperforms by far DialTest, the state-of-the-art framework in systematically generating semantically-preserved model inputs.

Conclusion and perspectives

The goal of this graduation project at the SoftWare Analytics and Technologies (SWAT) Laboratory was to create a new advanced chatbot robustness assessment framework. To be more specific, we suggest RLTest4Chatbot, an RL-based software testing framework that assesses DST chatbot performance by systematically generating new meaningful adversial model inputs using NLP attacks.

Our research shows that conventional testing methods are ineffective. They only evaluate chatbots based on global metrics such as accuracy, on small and clean testsets. More advanced robustness assessment frameworks have been proposed. They use non-parametric NLP transformations with either a random strategy or by applying static acceptance test input criteria, and they ignore some chatbot tasks like DST. Thus,the goal of our project was to develop a more dedicated and effective approach for assessing state tracking chatbot performance using RL-based Software Testing. To do so, RLTest4Chatbot (i) extends the search input space using an association of semantically-preserved parametric NLP transformations, then (ii) explores the search space following a reinforcement learning approach.

RLTest4Chatbot is evaluated on state-of-the-art chatbot DST models: Trade-DST [50] and Simple-TOD [51] using Multiwoz benchmark [1–5]. Results show that RLTest4Chatbot succeeds in generating human-like perturbated test input that induce behavioral disagreements. RLTest4Chatbot also succeeds in outperforming the state-of-the-art testing framework, DialTest [40], in generating valid adversial inputs.

For future work, we intend to extend our framework, RLTest4Chatbot, in many ways. First, we want it to be a general framework for testing chatbots that provides assessments for all conversational systems modules: NLU, DST and NLG. Second, we aim to improve the flexibility of the action space by allowing testers to select used NLP transformations.

Last but not least, even though our suggested NLP actions are effective in exposing vulnerabilities, we want to add support for sentence level transformations to boost attack variety.

A

Supplementary Results

This section contains further details about our experiments for the second research question (RQ2): How effective is RLTest4Chatbot at enhancing the coverage of chatbot faults?

Table A.1: Simple-TOD valid and faillure rates respectivally for Multiwoz 2.1 and 2.2 using the Reinforcement Learning approach

Table A.2: Multiwoz 2.1

#**Trans**	**Failure Rate**	**Valid Rate**
1	5.62 %	95.609 %
2	12.5 %	96.26%
3	18.8 %	92.52 %
4	11.41 %	93.98 %
5	23.93 %	84.22 %
6	23.34 %	98.21 %

Table A.3: Multiwoz 2.2

#**Trans**	**Failure Rate**	**Valid Rate**
1	7.97 %	99.41 %
2	14.34 %	98.45 %
3	11.52 %	99.03 %
4	14.62 %	88.58 %
5	19.34 %	98.83 %
6	28.94 %	80.35 %

Table A.4: Simple-TOD valid and faillure rates respectivally for Multiwoz 2.1 and 2.2 using the Random Sampling approach

Table A.5: Multiwoz 2.1

#**Trans**	**Failure Rate**	**Valid Rate**
1	7.91 %	96.58 %
2	13.59 %	80.18%
3	16.24 %	51.05 %
4	22.92 %	51.05 %
5	30.57 %	51.05 %
6	33.12 %	51.05 %

Table A.6: Multiwoz 2.2

#**Trans**	**Failure Rate**	**Valid Rate**
1	5.52 %	98.06 %
2	11.19 %	81.23 %
3	17.32 %	53.57 %
4	23.46 %	53.57 %
5	28.88 %	53.57 %
6	35.01 %	53.57 %

Table A.7: Trade-DST valid and faillure rates respectivally for Multiwoz 2.1 and 2.2 using the Reinforcement Learning approach

Table A.8: Multiwoz 2.1

#**Trans**	**Failure Rate**	**Valid Rate**
1	8.2 %	96 %
2	17.17 %	98.29 %
3	16.34 %	99.35 %
4	20.215 %	99.35 %
5	20.215 %	99.35 %
6	23.5 %	99.14 %

Table A.9: Multiwoz 2.2

#**Trans**	**Failure Rate**	**Valid Rate**
1	10.022 %	99.54 %
2	14.49 %	92.29 %
3	19.66 %	93.424 %
4	18.06 %	89.115 %
5	23.51 %	99.31 %
6	30.05 %	98.86 %

Bibliography

[1] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[2] Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 422–428, Marseille, France, May 2020. European Language Resources Association.

[3] Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. MultiWOZ 2.2 : A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online, July 2020. Association for Computational Linguistics.

[4] Ting Han, Ximing Liu, Ryuichi Takanobu, Yixin Lian, Chongxuan Huang, Dazhen Wan, Wei Peng, and Minlie Huang. Multiwoz 2.3: A multi-domain task-oriented dialogue dataset enhanced with annotation corrections and co-reference annotation. In *NLPCC*, 2021.

[5] Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. Multiwoz 2.4: A multi-domain task-oriented dialogue dataset with essential annotation corrections to improve state tracking evaluation. *ArXiv*, abs/2104.00773, 2021.

[6] Tsong Yueh Chen. Metamorphic testing: A simple method for alleviating the test oracle problem. In *Proceedings of the 10th International Workshop on Automation of Software Test*, AST '15, page 53â54. IEEE Press, 2015.

[7] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, T. Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *ArXiv*, abs/1810.06394, 2018.

[8] Eliza. https://en.wikipedia.org/wiki/ELIZA.

[9] Chatbot statistics. https://startupbonsai.com/chatbot-statistics/.

[10] 9 epic chatbot/conversational bot failures. https://research.aimultiple.com/chatbot-fail/.

[11] The good, the bad and the ugly of chatbots. https://venturebeat.com/ai/the-good-the-bad-and-the-ugly-of-chatbots/.

[12] Advantages, risks and use cases of chatbots in health care. https://varfix.ai/blog/advantages-risks-use-cases-chatbots-health-care/.

[13] Polytechnique montréal website. http://admission.polymtl.ca/.

[14] Swat lab website. http://swat.polymtl.ca/.

[15] Nicole Radziwill and Morgan Benton. Evaluating quality of chatbots and intelligent conversational agents. 04 2017.

[16] Jiexi Liu, Ryuichi Takanobu, Jiaxin Wen, Dazhen Wan, Hongguang Li, Weiran Nie, Cheng Li, Wei Peng, and Minlie Huang. Robustness testing of language understanding in task-oriented dialog. 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 2467–2480, Online, August 2021. Association for Computational Linguistics.

[17] Kanban : une méthode agile de gestion de projet visuelle et continue. https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1443832-kanban-une-methode-agile-de-gestion-de-projet-visuelle-et-continue/.

[18] Mesitertask. https://https://www.meistertask.com/.

[19] *chatbot | Definition of chatbot in English.* Lexico Dictionaries.

[20] Eleni Adamopoulou and Lefteris Moussiades. An overview of chatbot technology. pages 373–383, 05 2020.

[21] Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, Eric King, Kate Bland, Amanda Wartick, Yi Pan, Han Song, Sk Jayadevan, Gene Hwang, and Art Pettigrue. Conversational ai: The science behind the alexa prize. 01 2018.

[22] Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[23] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing.* Wiley Publishing, 3rd edition, 2011.

[24] Avinash Hedaoo and Abha Khandelwal. Study of dynamic testing techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7:322–330, 04 2017.

[25] Divyani Taley. Comprehensive study of software testing techniques and strategies: A review. *International Journal of Engineering Research and*, V9, 09 2020.

[26] Elaine J. Weyuker. On Testing Non-Testable Programs. *The Computer Journal*, 25(4):465–470, 11 1982.

[27] Abdur R. Fayjie, Sabir Hossain, Doukhi Oualid, and Deok-Jin Lee. Driverless car: Autonomous driving using deep reinforcement learning in urban environment. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 896–901, 2018.

[28] Jie Zhu, Fengge Wu, and Junsuo Zhao. An overview of the action space for deep reinforcement learning. *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence*, 2021.

[29] Warwick Masson, Pravesh Ranchod, and George Konidaris. Reinforcement learning with parameterized actions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016.

[30] Anssi Kanervisto, Christian Vibe Scheller, and Ville Hautamäki. Action space shaping in deep reinforcement learning. *2020 IEEE Conference on Games (CoG)*, pages 479–486, 2020.

[31] Zhou Fan, Rui Su, Weinan Zhang, and Yong Yu. Hybrid actor-critic reinforcement learning in parameterized action space. In *IJCAI*, pages 2279–2285, 2019.

[32] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Jost Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. 01 2020.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[34] Chengrun Qiu, Yang Hu, Yan Chen, and Bing Zeng. Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications. *IEEE Internet of Things Journal*, 6:8577–8588, 10 2019.

[35] Josip Bozic, Oliver A. Tazl, and Franz Wotawa. Chatbot testing using ai planning. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 37–44, 2019.

[36] Josip Bozic and Franz Wotawa. Security testing for chatbots. In Inmaculada Medina-Bulo, Mercedes G. Merayo, and Robert Hierons, editors, *Testing Software and Systems*, pages 33–38, Cham, 2018. Springer International Publishing.

[37] Haochen Liu, Zhiwei Wang, Tyler Derr, and Jiliang Tang. Chat as expected: Learning to manipulate black-box neural dialogue models, 05 2020.

[38] Haochen Liu, Tyler Derr, Zitao Liu, and Jiliang Tang. Say what i want: Towards the dark side of neural dialogue models, 09 2019.

[39] Alvin Chan, Lei Ma, Felix Juefei-Xu, Yew-Soon Ong, Xiaofei Xie, Minhui Xue, and Yang Liu. Breaking neural reasoning architectures with metamorphic relation-based adversarial examples. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–7, 2021.

[40] Zixi Liu, Yang Feng, and Zhenyu Chen. Dialtest: Automated testing for recurrent-neural-network-driven dialogue systems. ISSTA 2021, page 115â126, New York, NY, USA, 2021. Association for Computing Machinery.

[41] John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp, 2020.

[42] Basemah Alshemali and Jugal Kalita. Improving the reliability of deep neural networks in nlp: A review. *Know.-Based Syst.*, 191(C), mar 2020.

[43] Jaro distance. https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance.

[44] Jaccard distance. https://en.wikipedia.org/wiki/Jaccard_index.

[45] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2890–2896, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[46] Nltk. https://www.nltk.org/.

[47] Gensim. https://radimrehurek.com/gensim/.

[48] Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[49] Matthew Henderson, Blaise Thomson, and Jason D. Williams. The second dialog state tracking challenge. In *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pages 263–272, Philadelphia, PA, U.S.A., June 2014. Association for Computational Linguistics.

[50] Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the As-*

sociation for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2019.

[51] Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20179–20191. Curran Associates, Inc., 2020.

[52] Gpt-2 description from wikipedia. https://en.wikipedia.org/wiki/GPT-2.

اختبار متانة روبوت الدردشة باستخدام التعلم المعزز

روبوتات المحادثة هي برامج كمبيوتر مصممة لمحاكاة المحادثات مع المستخدمين البشريين. يتم اعتماد هذه الأنظمة بشكل متزايد من قبل العديد من الصناعات من أجل الأتمتة بعض جوانب خدمات العملاء. مع تزايد الطلب باستخدام روبوتات المحادثة في مهام أكثر أهمية ، أصبح ضمان موثوقية هذه الأنظمة أمرًا بالغ الأهمية وذات أهمية قصوى. طرق الاختبار التقليدية ، باستخدام عينات بيانات صغيرة ونظيفة ، ليست فعالة لأنها لا تعكس التنوع والحجم الكبير لمساحة الإدخال. نتيجة لذلك ، فإن اقتراح استراتيجيات اختبار متقدمة هو حاجة ملحة. لتحقيق هذه الغاية ، نقترح نهج اختبار التعلم المعزز، RLTest4Chatbot ، بهدف تقييم قدرة تتبع حالة الحوار. يقوم RLTest4chatbot بتوسيع مساحة إدخال الاختبار باستخدام رابطة من تحولات البرمجة اللغوية العصبية (NLP) للمحافظة على الدلالة وتستخدم RL لتعزيز تغطية الضعف.

الكلمات الرئيسية: اختبار البرمجيات ، روبوتات المحادثة ، تتبع حالة الحوار ، التعلم المعزز ، تحويلات NLP

Test de Robustesse du Chatbot à l'Aide de l'Apprentissage par Renforcement

Les chatbots sont des programmes informatiques conçus pour simuler des conversations avec des utilisateurs humains.Ces systèmes sont de plus en plus adoptés par de multiples industries afin d'automatiser certains aspects des services à la clientèle. Avec la demande croissante et l'utilisation des chatbots dans des tâches plus critiques, assurer la fiabilité de ces systèmes est devenu crucial et d'une importance primordiale. Les méthodes de test traditionnelles, utilisant des échantillons de données petits et propres, ne sont pas efficaces car elles ne reflètent pas la diversité et la grande taille de l'espace d'entrée. Par conséquent, proposer des stratégies de test avancées est un besoin urgent. À cette fin, nous proposons une approche de test d'apprentissage par renforcement pour Chatbot, RLTest4Chatbot, visant à évaluer la capacité de suivi de l'état du dialogue. RLTest4chatbot étend l'espace d'entrée de test en utilisant
une association de transformations NLP paramétriques préservant sémantiquement en utilisant RL pour améliorer la couverture des vulnérabilités.

Mots-clés: Tests logiciels, Chatbots, Suivi de l'état du dialogue, Apprentissage par renforcement, Transformations NLP

Chatbot Robustness Testing using Reinforcement Learning

Chatbots are computer programs designed to simulate conversations with human users. These systems are increasingly being adopted by multiple industries in order to automate some aspects of customer services. With the rising demand and the use of chatbots in more critical tasks, ensuring the reliability of these systems have become crucial and of paramount importance. Traditional testing methods, using small and clean data samples, are not effective because they don't reflect the diversity and the large size of input space. As a result, proposing advanced testing strategies is an urgent need. To that end we propose a Reinforcement Learning Testing approach for Chatbot, RLTest4Chatbot, aiming to evaluate the Dialogue State Tracking capability. RLTest4chatbot extends the testing input space using an association of parametric semantically-preserving NLP transformations and employs RL to enhance vulnerability coverage.

Keywords: Software Testing, Chatbots, Dialogue State Tracking, Reinforcement Learning, NLP Transformations

Intitule et adresse complète de l'entreprise :

Entreprise : SoftWare Analytics and Technologies (SWAT) Lab, DGIGL, École Polytechnique de Montréal
Adresse : Room: M 4123, C.P. 6079, succ. Centre-Ville, Montréal, QC, H3C 3A7, Canada
Tél. : (+1) 514-340-4711 Ext. 4233
Fax : (+1) 514-340-5139
Email : foutse.khomh@polymtl.ca